



# 免費電子書

## 學習

# C++

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

#C++

	1
<b>1: C ++</b>	2
.....	2
.....	2
Examples	2
.....	2
.....	2
.....	3
.....	3
<b>C/</b>	3
.....	4
.....	4
.....	4
.....	5
.....	5
.....	5
.....	5
.....	6
-	6
.....	6
<b>C ++</b>	7
.....	8
<b>2: Arithmetic Metaprogramming</b>	9
.....	9
Examples	9
Olog n	9
<b>3: C ++ 11</b>	11
.....	11
.....	11
.....	11

.....	11
-	11
-	12
.....	12
<b>Examples</b> .....	12
.....	12
.....	13
<b>4: C ++ Streams</b> .....	15
.....	15
<b>Examples</b> .....	15
.....	15
.....	17
.....	17
.....	17
.....	17
.....	17
.....	17
.....	18
.....	18
<b>iostream</b> .....	18
.....	18
.....	18
.....	19
.....	19
.....	19
<b>STL</b> .....	19
.....	19
.....	20
<b>5: C ++</b> .....	21
.....	21
<b>Examples</b> .....	21

	.....	21
<b>6: C ++</b>	.....	<b>22</b>
Examples	.....	22
.....	.....	22
.....	.....	22
.....	.....	22
.....	.....	22
.....	.....	22
.....	.....	23
/	.....	23
.....	.....	23
.....	.....	24
.....	.....	24
.....	.....	25
<b>7: C ++</b>	.....	<b>26</b>
.....	.....	26
Examples	.....	26
.....	.....	26
.....	.....	26
.....	.....	26
<b>8: C ++</b>	.....	<b>29</b>
.....	.....	29
Examples	.....	29
C ++	.....	29
<b>9: c ++/</b>	.....	<b>31</b>
.....	.....	31
Examples	.....	31
.....	.....	31
<b>10: C ++</b>	.....	<b>32</b>
Examples	.....	32
C ++	.....	32
<b>11: C ++</b>	.....	<b>34</b>



<b>15: constexpr</b>	<b>48</b>
.....	48
.....	48
.....	48
<b>Examples</b>	<b>48</b>
<b>constexpr</b>	<b>48</b>
<b>constexpr</b>	<b>50</b>
<b>if</b>	<b>51</b>
<b>16: Const</b>	<b>53</b>
.....	53
.....	53
<b>Examples</b>	<b>53</b>
.....	53
<b>Const</b>	<b>54</b>
<b>Const</b>	<b>57</b>
<b>Const</b>	<b>58</b>
<b>const CV-Qualified Member Functions</b>	<b>58</b>
<b>const</b>	<b>59</b>
<b>17: const</b>	<b>62</b>
.....	62
.....	62
<b>Examples</b>	<b>62</b>
<b>Const</b>	<b>62</b>
<b>Const</b>	<b>62</b>
<b>Const</b>	<b>63</b>
<b>constconst getter</b>	<b>63</b>
<b>18: C</b>	<b>65</b>
.....	65
<b>Examples</b>	<b>65</b>
.....	65
.....	65
.....	65

<b>19: decltype</b> .....	<b>66</b>
.....	66
<b>Examples</b> .....	<b>66</b>
.....	66
.....	66
<b>20: ISO C ++</b> .....	<b>67</b>
.....	67
.....	67
<b>Examples</b> .....	<b>67</b>
.....	67
<b>C ++ 11</b> .....	<b>67</b>
.....	67
.....	67
.....	68
.....	68
.....	68
.....	68
.....	68
.....	68
.....	68
.....	68
.....	68
.....	68
.....	68
.....	69
<b>C ++ 14</b> .....	<b>69</b>
.....	69
.....	69
<b>/</b> .....	<b>69</b>
<b>C ++ 17</b> .....	<b>69</b>
.....	69
.....	70
<b>C ++ 03</b> .....	<b>70</b>
.....	70

C ++ 98.....	70
<b>C89 / C90.....</b>	<b>70</b>
.....	70
C ++ 20.....	71
.....	71
.....	71
<b>21: Lambda.....</b>	<b>72</b>
.....	72
.....	72
.....	72
Examples.....	72
lambda.....	72
.....	74
.....	74
.....	75
.....	76
.....	77
lambdas.....	77
.....	78
lambda.....	79
lambdaC ++ 03.....	80
lambdas.....	82
std::function.....	82
.....	82
Y.....	83
lambdas.....	83
<b>22: Pimpl.....</b>	<b>86</b>
.....	86
Examples.....	86
Pimpl.....	86
<b>23: RAII.....</b>	<b>88</b>
.....	88

Examples.....	88
.....	88
/ ScopeExit.....	89
ScopeSuccessc ++ 17.....	89
ScopeFailc ++ 17.....	91
<b>24: RTTI.....</b>	<b>94</b>
Examples.....	94
.....	94
dynamic_cast.....	94
typeid.....	94
c ++.....	95
<b>25: SFINAE.....</b>	<b>96</b>
Examples.....	96
enable_if.....	96
.....	96
void_t.....	97
decltype.....	98
SFINAE.....	98
enable_if_all / enable_if_any.....	99
is_detected.....	100
.....	101
<b>26: static_assert.....</b>	<b>103</b>
.....	103
.....	103
.....	103
Examples.....	103
static_assert.....	103
<b>27: std :: function.....</b>	<b>104</b>
Examples.....	104
.....	104
std :: functionstd :: bind.....	104
std :: function with lambdastd :: bind.....	105

`function` .....	105
std :: function .....	106
std :: tuple .....	110
<b>28: std :: setstd :: multiset .....</b>	<b>113</b>
..... .....	113
..... .....	113
Examples .....	113
..... .....	113
..... .....	114
..... .....	115
..... .....	116
..... .....	116
Lambda .....	116
..... .....	117
setmultiset .....	117
..... .....	118
<b>29: Typedef .....</b>	<b>119</b>
..... .....	119
..... .....	119
Examples .....	119
typedef .....	119
typedef .....	119
typedef .....	120
`"	120
<b>30: ODR .....</b>	<b>121</b>
Examples .....	121
..... .....	121
..... .....	121
ODR .....	122
<b>31: .....</b>	<b>124</b>
Examples .....	124
..... .....	124

.....	125
.....	126
.....	128
<b>32: C ++C ++ 11C ++ 14C ++ 17C ++</b>	<b>129</b>
Examples.....	129
.....	129
<b>33:</b>	<b>130</b>
.....	130
<b>std :: shared_mutexstd :: shared_timed_mutex</b>	<b>130</b>
std :: shared_mutexMSVC14.1.....	130
std :: shared_timed_mutexMSVC14.1.....	131
std :: shared_mutex/std :: shared_timed_mutex2 .....	133
Examples.....	135
std :: unique_lockstd :: shared_lockstd :: lock_guard.....	135
std :: try_to_lockstd :: adopt_lockstd :: defer_lock.....	135
std ::.....	138
std :: scoped_lockC ++ 17.....	138
.....	138
std ::.....	139
<b>34:</b>	<b>140</b>
.....	140
.....	140
Examples.....	140
.....	140
<b>35:</b>	<b>142</b>
.....	142
Examples.....	142
.....	142
<b>C</b>	<b>142</b>
<b>std :: bitset</b>	<b>142</b>
.....	142

C	142
std :: bitset	142
.....	143
C	143
std :: bitset	143
.....	143
C	143
std :: bitset	143
nx	143
C	143
std :: bitset	143
.....	144
C	144
std :: bitset	144
.....	144
C	144
.....	144
2	145
.....	145
36:	147
.....	147
Examples	147
- AND	147
- OR	147
^ -	148
- NOT	149
<< -	150
>> -	151
37: OpenMP	152
.....	152
.....	152

Examples.....	152
OpenMP.....	152
OpenMP.....	153
OpenMPParallel For Loop.....	153
OpenMP/.....	154
<b>38: std :: unordered_map</b> .....	<b>155</b>
.....	155
.....	155
Examples.....	155
.....	155
.....	155
<b>39:</b> .....	<b>156</b>
Examples.....	156
.....	156
.....	156
<b>40:</b> .....	<b>158</b>
.....	158
.....	158
.....	158
Examples.....	158
.....	158
.....	158
.....	158
<b>41:</b> .....	<b>160</b>
Examples.....	160
.....	160
Rethrow.....	161
Try Blocks.....	162
.....	162
.....	162
const.....	163
.....	163

std :: uncaught_exceptions.....	166
.....	168
<b>42: .....</b>	<b>172</b>
.....	172
<b>Examples.....</b>	<b>172</b>
C ++ 11.....	172
.....	172
<b>43: .....</b>	<b>174</b>
<b>Examples.....</b>	<b>174</b>
.....	174
.....	176
<b>44: .....</b>	<b>177</b>
<b>Examples.....</b>	<b>177</b>
.....	177
<b>prvalue.....</b>	<b>177</b>
X.....	177
.....	178
<b>glvalue.....</b>	<b>178</b>
.....	178
<b>45: .....</b>	<b>180</b>
.....	180
<b>Examples.....</b>	<b>180</b>
/.....	180
.....	180
<b>46: .....</b>	<b>182</b>
.....	182
.....	182
<b>Examples.....</b>	<b>182</b>
.....	182
.....	185
<b>std :: integer_sequence.....</b>	<b>186</b>
.....	187

.....	187
C ++ 11.....	189
T.....	189
IF-THEN-ELSE.....	190
/.....	191
<b>47:</b> .....	<b>192</b>
.....	192
.....	192
Examples.....	192
.....	192
.....	193
.....	193
<b>48:</b> .....	<b>195</b>
.....	195
.....	195
.....	195
.....	195
.....	195
.....	195
.....	195
Examples.....	195
.....	195
.....	195
.....	195
.....	196
<b>49:</b> .....	<b>197</b>
.....	197
Examples.....	197
.....	197
<b>50:</b> .....	<b>198</b>
Examples.....	198
.....	198
switch.....	198

.....	198
Scoped enums.....	200
C ++ 11.....	201
<b>51:</b> .....	<b>202</b>
Examples.....	202
gccgprof.....	202
gperf2dot.....	202
gccGoogle Perf ToolsCPU.....	204
<b>52:</b> .....	<b>206</b>
.....	206
Examples.....	206
.....	206
<b>53:</b> .....	<b>207</b>
.....	207
.....	207
Examples.....	207
.....	207
.....	208
cv-qualifier.....	208
<b>54:</b> .....	<b>211</b>
.....	211
.....	211
Examples.....	211
.....	211
<b>55:</b> .....	<b>213</b>
Examples.....	213
.....	213
<b>56:</b> .....	<b>214</b>
Examples.....	214
.....	214
.....	214

<b>57:</b>	<b>215</b>
Examples.....	215
.....	215
C ++.....	215
<b>58:</b>	<b>217</b>
.....	217
.....	217
Examples.....	217
.....	217
operatorFunction.....	217
<b>59:</b>	<b>219</b>
Examples.....	219
.....	219
lambdas.....	219
<b>60:</b>	<b>221</b>
.....	221
.....	221
Examples.....	221
.....	221
.....	222
.....	223
.....	223
.....	224
ADL.....	224
.....	224
/.....	226
.....	226
.....	226
.....	227
.....	227
<b>61:</b>	<b>229</b>

.....	229
Examples.....	229
.....	229
.....	230
Singeton.....	231
.....	231
<b>62:</b> .....	<b>232</b>
Examples.....	232
INT.....	232
.....	232
.....	232
char16_t.....	232
char32_t.....	232
.....	232
.....	233
.....	233
.....	233
.....	233
wchar_t.....	233
<b>63:</b> .....	<b>235</b>
Examples.....	235
.....	235
.....	235
.....	236
<b>64: CRTP</b> .....	<b>238</b>
.....	238
Examples.....	238
CRTP.....	238
CRTP.....	240
<b>65:</b> .....	<b>243</b>
.....	243
Examples.....	243

243	243
.....	243
nullptr	243
.....	243
.....	244
66:	246
.....	246
.....	246
Examples	246
.....	246
.....	246
.....	247
.....	248
EXTERN	248
67:	249
.....	249
Examples	249
.....	249
68:	250
Examples	250
TCP	250
TCP	255
69:	258
Examples	258
Char	258
.....	258
char	258
.....	258
char16_tchar32_t	259
bool	259
wchar_t	259

<b>260</b>	.....	260
.....	.....	260
.....	.....	261
.....	.....	262
.....	.....	262
.....	.....	262
<b>70:</b>	.....	263
.....	.....	263
Examples	.....	263
.....	.....	263
.....	.....	264
.....	.....	264
.....	.....	264
.....	.....	264
.....	.....	264
<b>71:</b>	.....	265
.....	.....	265
.....	.....	265
Examples	.....	265
.....	.....	265
.....	.....	265
<b>72:</b>	.....	267
.....	.....	267
Examples	.....	267
[[[]]	.....	267
[[[]]]	.....	268
[[[deprecated]][[deprecated“reason”]]]	.....	268
[[nodiscard]]	.....	269
[[maybe_unused]]	.....	269
<b>73:</b>	.....	271

271	
Examples.....	271
.....	271
.....	271
.....	272
<b>74:</b> .....	<b>273</b>
.....	273
.....	273
.....	273
Examples.....	273
regex_matchregex_search.....	273
regex_replace.....	274
regex_token_iterator.....	274
regex_iterator.....	274
.....	275
.....	275
.....	276
<b>75: /GCC</b> .....	<b>277</b>
Examples.....	277
****.....	277
.....	277
.....	277
****.....	278
***.....	278
<b>76:</b> .....	<b>280</b>
.....	280
Examples.....	280
.....	280
<b>77:</b> .....	<b>281</b>
.....	281
Examples.....	281
.....	281

std :: tuple.....	281
std :: array.....	282
std :: pair.....	282
struct.....	283
.....	284
.....	285
std :: vector.....	286
.....	286
<b>78:</b> .....	<b>288</b>
.....	288
.....	288
.....	288
.....	288
<b>Examples</b> .....	<b>288</b>
.....	288
.....	290
.....	292
.....	292
<b>Do-while</b> .....	<b>293</b>
.....	294
-	295
<b>79:</b> .....	<b>298</b>
.....	298
<b>Examples</b> .....	<b>298</b>
.....	298
.....	298
.....	298
<b>80:</b> .....	<b>300</b>
.....	300
<b>Examples</b> .....	<b>300</b>
.....	300
.....	300
.....	301

301	
81:	303
.....	303
.....	303
.....	303
.....	303
Examples	303
.....	303
.....	303
.....	304
.....	304
.....	305
.....	306
.....	307
/	307
/	307
.....	307
82:	309
.....	309
Examples	309
.....	309
.....	309
.....	310
lambdaC ++ 11	311
.....	312
std :: map	313
.....	314
83:	316
Examples	316
.....	316
84:	317
.....	317

Examples.....	317
.....	317
.....	318
std :: vector.....	320
2D.....	321
std :: vector.....	322
.....	322
<b>85: I / O.</b> .....	<b>324</b>
.....	324
Examples.....	324
.....	324
.....	325
.....	326
.....	326
.....	327
.....	327
ASCIIstd :: string.....	328
.....	329
`struct`.....	329
.....	331
.....	332
.....	332
<b>86:</b> .....	<b>335</b>
.....	335
.....	335
Examples.....	335
std :: shared_ptr.....	335
std :: weak_ptr.....	336
std :: unique_ptr.....	338
C.....	339
auto_ptr.....	340
shared_ptr.....	341

std :: shared_ptr.....	341
value_ptr.....	342
<b>87: .....</b>	<b>345</b>
.....	345
Examples.....	345
.....	345
.....	346
.....	346
<b>88: .....</b>	<b>348</b>
.....	348
Examples.....	348
std :: futurestd :: promise.....	348
.....	348
std :: packaged_taskstd :: future.....	349
std :: future_errorstd :: future_errc.....	349
std :: futurestd :: async.....	350
.....	351
<b>89: .....</b>	<b>352</b>
Examples.....	352
.....	352
.....	352
.....	352
.....	353
<b>90: .....</b>	<b>354</b>
.....	354
.....	354
Examples.....	354
.....	354
voidreturn.....	354
.....	355
.....	355
.....	355

355	
.....	356
.....	356
.....	357
.....	358
.....	358
.....	358
.....	359
.....	359
`std` `posix`	359
.....	360
.....	360
.....	360
const	360
.....	361
.....	361
.....	361
.....	362
[[noreturn]]	362
.....	362
.....	363
<b>91:</b>	<b>364</b>
.....	364
<b>Examples</b>	<b>364</b>
TU	364
.....	364
*	365
reinterpret_cast	365
.....	365
.....	366
.....	366
.....	367
<b>92:</b>	<b>369</b>

.....	369
.....	369
Examples.....	369
CMake.....	369
GNU make.....	370
.....	370
.....	370
.....	371
.....	373
SCons.....	373
.....	373
.....	373
NMAKEMicrosoft.....	373
.....	373
AutotoolsGNU.....	373
.....	373
93: .....	375
Examples.....	375
std :: for_each.....	375
std :: next_permutation.....	375
std :: .....	376
std :: .....	377
std :: .....	377
std :: count_if.....	378
std :: find_if.....	378
std :: min_element.....	379
std :: nth_element.....	379
94: .....	381
.....	381
.....	381
.....	381

Examples.....	382
.....	382
.....	383
.....	383
.....	384
.....	384
.....	386
.....	386
.....	386
auto.....	387
unique_ptr.....	387
.....	388
.....	388
.....	389
<b>95:</b> .....	<b>391</b>
.....	391
Examples.....	391
.....	391
.....	392
autoconstrefrences.....	392
.....	392
lambdaC ++ 14.....	393
.....	393
<b>96:</b> .....	<b>395</b>
.....	395
.....	395
Examples.....	396
.....	396
.....	401
.....	402
<b>97:</b> .....	<b>404</b>
.....	404

Examples.....	404
.....	404
.....	404
.....	404
.....	405
.....	405
.....	405
.....	406
.....	406
.....	407
.....	407
ifif..else.....	407
breakcontinuegotoexit.....	408
<b>98:</b> .....	<b>411</b>
Examples.....	411
.....	411
<b>99:</b> .....	<b>412</b>
Examples.....	412
.....	412
.....	413
.....	413
.....	415
.....	418
<b>100: C ++</b> .....	<b>422</b>
.....	422
.....	422
Examples.....	422
.....	422
.....	426
.....	430
Fluent APIBuilder.....	431
.....	433

<b>434</b>	
<b>101:</b>	.....
Examples.....	435
.....	435
.....	435
.....	436
.....	437
.....	437
<b>102: std :: integer_sequence</b> .....	439
.....	439
Examples.....	439
std :: tuple .....	439
.....	440
.....	440
<b>103: std :: string</b> .....	441
.....	441
.....	441
.....	441
Examples.....	442
.....	442
.....	442
.....	442
.....	443
.....	444
.....	444
[]n.....	445
n.....	445
.....	445
.....	445
.....	445
constchar *.....	446

<b>446</b>	
/.....	447
.....	448
std :: wstring.....	449
std :: string_view.....	450
.....	451
/.....	451
.....	452
.....	453
std :: string.....	453
<b>104: std ::iomanip.....</b>	<b>455</b>
Examples.....	455
std ::.....	455
std :: setprecision.....	455
std :: setfill.....	456
std :: setiosflags.....	456
<b>105: std ::.....</b>	<b>458</b>
.....	458
Examples.....	458
.....	458
<b>106: std ::Modifiers.....</b>	<b>459</b>
.....	459
.....	459
Examples.....	459
.....	459
.....	460
<b>107: std ::.....</b>	<b>462</b>
Examples.....	462
.....	462
<b>108: std ::.....</b>	<b>465</b>
Examples.....	465
.....	465

465	465
vs.....	465
vs Sentinel.....	465
vs std::pair<bool, T>.....	465
.....	465
.....	466
.....	467
value_or.....	467
<b>109: std ::</b>	<b>468</b>
.....	468
Examples.....	468
.....	468
std :: mapstd :: multimap.....	469
.....	469
.....	469
std :: mapstd :: multimap.....	470
std :: mapstd :: multimap.....	471
.....	472
.....	472
.....	472
.....	472
std :: map.....	473
.....	473
<b>110: std ::</b>	<b>474</b>
Examples.....	474
.....	474
.....	474
<b>111: std ::</b>	<b>476</b>
.....	476
.....	476

<b>Examples</b>	476
<b>std :: vector</b>	476
<b>.....</b>	477
<b>std :: vector</b>	478
<b>.....</b>	478
<b>.....</b>	478
<b>const</b>	479
<b>.....</b>	480
<b>std :: vectorC</b>	481
<b>/</b>	481
<b>.....</b>	482
<b>.....</b>	483
<b>.....</b>	483
<b>lambda</b>	483
<b>.....</b>	483
<b>.....</b>	483
<b>std :: vector</b>	484
<b>std :: vector</b>	485
<b>.....</b>	485
<b>.....</b>	486
<b>.....</b>	487
<b>.....</b>	488
<b>.....</b>	489

490	491
.....	491
.....	491
<b>112: std ::</b>	<b>493</b>
.....	493
Examples	493
std :: variant	493
.....	493
`std :: variant`	496
<b>113: std ::</b>	<b>498</b>
.....	498
.....	498
Examples	498
std :: array	498
.....	498
.....	501
.....	501
.....	501
<b>114:</b>	<b>503</b>
Examples	503
.....	503
.....	503
.....	504
std :: moveOn?On	504
.....	509
.....	509
<b>115:</b>	<b>511</b>
.....	511
.....	511
.....	511
Examples	511
.....	511

.....	511
std :: thread.....	512
.....	514
std :: asyncstd :: thread.....	515
.....	515
.....	515
.....	516
.....	516
.....	517
.....	517
.....	519
.....	521
<b>116:</b> .....	<b>522</b>
.....	522
Examples.....	522
std :: shared_lock.....	522
std :: call_oncestd :: once_flag.....	522
.....	523
std :: condition_variable_anystd :: cv_status.....	525
<b>117:</b> .....	<b>526</b>
.....	526
.....	526
Examples.....	526
GCC.....	526
.....	527
Visual C ++.....	527
Visual Studio - Hello World.....	530
Clang.....	537
.....	537
C ++.....	538
Code :: Blocks.....	539
<b>118:</b> .....	<b>544</b>

.....	544
.....	544
.....	544
Examples.....	544
Unix.....	544
CC ++.....	544
<b>119:</b> .....	<b>546</b>
.....	546
.....	546
Examples.....	546
C ++ 11override with virtual.....	546
.....	547
.....	548
.....	549
.....	550
<b>120:</b> .....	<b>552</b>
Examples.....	552
.....	552
vec.hhstd :: vector.....	553
File expr.hhvector plusvector inner product.....	553
main.ccsrc.....	554
.....	554
<b>121: Elision</b> .....	<b>561</b>
Examples.....	561
.....	561
.....	561
.....	562
.....	562
.....	563
elision.....	563
<b>122:</b> .....	<b>564</b>
.....	564

.....	564
.....	564
Examples.....	564
.....	564
.....	565
.....	565
<b>123:</b> .....	<b>568</b>
Examples.....	568
.....	568
decltype.....	568
.....	569
.....	569
.....	569
<b>124:</b> .....	<b>570</b>
.....	570
Examples.....	570
.....	570
.....	570
<b>125:</b> .....	<b>572</b>
Examples.....	572
.....	572
.....	573
.....	573
.....	574
<b>126:</b> .....	<b>575</b>
.....	575
Examples.....	575
1.....	575
2.....	576
3.....	576
<b>127:</b> .....	<b>578</b>
Examples.....	578

.....	.578
.....	.578
.....	.578
.....	.578
.....	.578
.....	.579
<b>128:</b>	<b>580</b>
Examples.....	580
C.....	580
.....	.580
.....	.582
.....	.582
.....	.582
.....	.582
.....	.583
.....	.583
.....	.583
.....	.583
.....	.583
.....	.584
.....	.584
Map Iterator.....	584
.....	.585
.....	.585
<b>129:</b>	<b>587</b>
.....	.587
.....	.587
Examples.....	587
.....	.587
Lambda.....	587
<b>130:</b>	<b>588</b>
.....	.588
Examples.....	588

588	590
.....	590
.....	590
Pointer CV-Qualifiers	592
.....	594
<b>131:</b>	<b>596</b>
.....	596
Examples	596
.....	596
ANDOR	597
&&	597
.....	598
<b>132:</b>	<b>599</b>
.....	599
.....	599
Examples	599
.....	599
.....	600
.....	601
.....	602
.....	602
.....	603
.....	603
NOT	604
I / O	604
.....	605
.....	609
<b>133:</b>	<b>611</b>
.....	611
Examples	611
.....	611
.....	611

612	612
.....	613
.....	615
.....	615
constnessvolatility	616
<b>134:</b>	<b>618</b>
Examples	618
std :: recursive_mutex	618
<b>135:</b>	<b>619</b>
.....	619
Examples	619
.....	619
.....	621
<b>136:</b>	<b>623</b>
.....	623
.....	623
.....	623
Examples	625
ASM	625
.....	625
noexcept	626
.....	627
sizeof	627
.....	627
<b>137:</b>	<b>631</b>
.....	631
Examples	631
.....	631
.....	631
.....	632
<b>138:</b>	<b>633</b>

.....	633
.....	633
Examples.....	633
.....	633
.....	634
.....	634
.....	636
Const.....	640
<b>139:</b> .....	<b>642</b>
.....	642
.....	642
Examples.....	642
.....	642
.....	643
.....	645
.....	649
.....	649
X-.....	650
#pragma.....	653
.....	653
<b>140:</b> .....	<b>655</b>
Examples.....	655
.....	655
.....	655
<b>141:</b> .....	<b>657</b>
.....	657
Examples.....	657
.....	657
.....	657
.....	658
.....	659
<b>142: /</b> .....	<b>661</b>

.....	661
.....	661
<b>Examples</b> .....	<b>661</b>
.....	661
.....	662
.....	663
.....	664
.....	665
.....	665
.....	666
.....	666
.....	668
.....	668
/.....	669
.....	671
.....	673
.....	681
/.....	682
<b>143:</b> .....	<b>684</b>
.....	684
<b>Examples</b> .....	<b>684</b>
.....	684
.....	684
.....	684
<b>144:</b> .....	<b>687</b>
.....	687
.....	687
<b>Examples</b> .....	<b>687</b>
.....	687
<b>Lambda</b> .....	<b>687</b>
.....	688
<b>145:</b> .....	<b>689</b>
.....	689

Examples.....	689
.....	689
vtable.....	691
`std :: function`.....	695
T.....	698
std :: any.....	699
<b>146:</b> .....	<b>703</b>
.....	703
Examples.....	703
.....	703
.....	703
.....	703
.....	703
.....	703
std :: is_same.....	704
.....	704
.....	705
<b>147:</b> .....	<b>707</b>
.....	707
.....	707
.....	707
Examples.....	708
.....	708
.....	708
.....	709
.....	709
.....	710
.....	710
.....	710
.....	711
*T*.....	711
C.....	712



---

You can share this PDF with anyone you feel could benefit from it, download the latest version from: [cplusplus](#)

It is an unofficial and free C++ ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official C++.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

# 1: C ++

“Hello World”。 C ++<iostream> std::cout。

C ++。 Compiling and Building C ++。

C ++ Standard		
C ++ 98	ISO / IEC 148821998	1998-09-01
C ++ 03	ISO / IEC 148822003	2003-10-16
C ++ 11	ISO / IEC 148822011	2011-09-01
C ++ 14	ISO / IEC 148822014	
C ++ 17	TBD	201711
C ++ 20	TBD	202011

## Examples

Hello World!

```
#include <iostream>

int main()
{
    std::cout << "Hello World!" << std::endl;
}
```

Coliru。

- #include <iostream> C ++iostream。
- iostream。 std。
- I/O。
- int main() { ... }main。 main。 C ++mainint。
- int。 main。
- 0EXIT\_SUCCESS。 。
- returnmain0。 return 0; 。

```
void°
```

- std::cout << "Hello World!" << std::endl;"Hello World"
  - std :: °
  - ::stdcout ° - Microsoft °
  - std::coutiosstream stdout °
  - << °
  - <<° stream << contentcontent° std::cout << "Foo" << " Bar";"FooBar"°
  - "Hello World!" "° iostream°
  - std::endl I/O iostream°
  - std::endl° ° "°°

```
#include <iostream>

int main()
{
    std::cout << "Hello World!" << std::endl;
}
```

\n °

- ; ° C ++/°

C ++° °

C ++

---

//

```
int main()
{
    // This is a single-line comment.
    int a; // this also is a single-line comment
    int i; // this is another single-line comment
}
```

---

CI

/\*\*/° C ++° "C" C ++C

```
int main()
{
    // This is a single-line comment.
    int a; // this also is a single-line comment
    int i; // this is another single-line comment
}
```

◦ \*/

```
int main()
{
    // This is a single-line comment.
    int a; // this also is a single-line comment
    int i; // this is another single-line comment
}
```

C ++C◦ /\*◦

◦

```
int main()
{
    // This is a single-line comment.
    int a; // this also is a single-line comment
    int i; // this is another single-line comment
}
```

- 
- /
  - 
  - 
  - /◦

- 
- 

◦ ◦ ◦

---

◦ ◦ “”◦

◦

◦

◦ ◦

- ◦

◦

- C ++◦

◦ ◦ ◦

● ◦ ◦

—

◦

```
int add2(int i); // The function is of the type (int) -> (int)
```

```
int add2(int i)
```

- int ◦
- add2 ◦
- 1
  - int◦
  - i◦

;

```
int add2(int i); // The function is of the type (int) -> (int)
```

## C ++C ++◦ -◦ C ++

```
int add2(int i); // The function is of the type (int) -> (int)
```

void ◦ ◦

```
int add2(int i); // The function is of the type (int) -> (int)
```

◦ main2 add2

```
int add2(int i); // The function is of the type (int) -> (int)
```

add2(2)◦

—  
\*◦

add2

```
int add2(int i); // The function is of the type (int) -> (int)
```

◦

```
int add2(int i); // The function is of the type (int) -> (int)
```

add2。 C ++。 。  
—  
。

```
int add2(int i); // The function is of the type (int) -> (int)
```

multiply()。 b7.。  
—  
—

```
int add2(int i); // The function is of the type (int) -> (int)
```

C ++name\_of\_function(value1, value2, value3)。 。  
! + - \* %<<。 +C ++。 。  
C ++;。 C ++

```
int add2(int i); // The function is of the type (int) -> (int)
```

```
int add2(int i); // The function is of the type (int) -> (int)
```

operator。 。  
C ++CC ++。 "C ++ C ++。 。  
C ++。 。  
int main()  
{  
 foo(2); // error: foo is called, but has not yet been declared  
}  
  
void foo(int x) // this later definition is not known in main  
{  
}

main() foo()。 。  
int main()  
{  
 foo(2); // error: foo is called, but has not yet been declared  
}

```
void foo(int x) // this later definition is not known in main
{
}
```

“main”

```
int main()
{
    foo(2); // error: foo is called, but has not yet been declared
}

void foo(int x) // this later definition is not known in main
{
}
```

void foo int °

```
int main()
{
    foo(2); // error: foo is called, but has not yet been declared
}

void foo(int x) // this later definition is not known in main
{
}
```

```
int main()
{
    foo(2); // error: foo is called, but has not yet been declared
}

void foo(int x) // this later definition is not known in main
{
}
```

foo.omain.o

```
int main()
{
    foo(2); // error: foo is called, but has not yet been declared
}

void foo(int x) // this later definition is not known in main
{
}
```

“”° °

**C ++**

**C ++°**

° °

C ++" "C。 C ++

1. C ++#define。
  2. C ++。
  3. 。
  4. 。

C ++。 C ++。

- [1] <http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html>

○

```
#define ZERO 0
```

```
#define ZERO 0
```

- something ◦ **hello world**

```
#define ZERO 0
```

1

CC ++ C ++C。

```
#define ZERO 0
```

- something something else ◦ C++◦

something else somethingsomething.

```
#if #else#endif。
```

```
#define ZERO 0
```

truefalse。 8

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/206/c-plusplus>

## 2: Arithmetic Metaprogramming

C++.

### Examples

Olog n

◦

```
template <int base, unsigned int exponent>
struct power
{
    static const int halfvalue = power<base, exponent / 2>::value;
    static const int value = halfvalue * halfvalue * power<base, exponent % 2>::value;
};

template <int base>
struct power<base, 0>
{
    static const int value = 1;
    static_assert(base != 0, "power<0, 0> is not allowed");
};

template <int base>
struct power<base, 1>
{
    static const int value = base;
};
```

```
template <int base, unsigned int exponent>
struct power
{
    static const int halfvalue = power<base, exponent / 2>::value;
    static const int value = halfvalue * halfvalue * power<base, exponent % 2>::value;
};

template <int base>
struct power<base, 0>
{
    static const int value = 1;
    static_assert(base != 0, "power<0, 0> is not allowed");
};

template <int base>
struct power<base, 1>
{
    static const int value = base;
};
```

C++ 14

```

template <int base, unsigned int exponent>
struct power
{
    static const int halfvalue = power<base, exponent / 2>::value;
    static const int value = halfvalue * halfvalue * power<base, exponent % 2>::value;
};

template <int base>
struct power<base, 0>
{
    static const int value = 1;
    static_assert(base != 0, "power<0, 0> is not allowed");
};

template <int base>
struct power<base, 1>
{
    static const int value = base;
};

```

Arithmitic Metaprogramming <https://riptutorial.com/zh-TW/cplusplus/topic/10907/arithmitic-metaprogramming>

# 3: C++ 11

◦ ◦ ◦

◦ C++ 11◦

- std::atomic<t>◦ t◦
- ◦ std::atomic<unsigned>std::atomic<std::vector<foo> \*>std::atomic<std::pair<bool,char>>◦
- ◦
  - ◦
  - ◦ ◦
  - - - ◦ ◦
  - std::memory\_order◦

## std :: memory\_order

std::memory\_order\_relaxed

std::memory\_order\_release →  
std::memory\_order\_acquire

load-acquirestore-releasestore-releasestore-  
release store-release

std::memory\_order\_consume

memory\_order\_acquire

std::memory\_order\_acq\_rel

load-acquirestore-release

std::memory\_order\_seq\_cst

- ◦

◦ std::memory\_order\_seq\_cst◦

◦ ◦ ◦ ◦

◦ std::memory\_order\_relaxed◦ ◦ ◦

-

std::memory\_order\_releasestd::memory\_order\_acquire◦ **load-acquire**◦

**load-acquirestore-releasestore-release load-acquire**◦

- - std::memory\_order\_acq\_rel◦ ◦

◦ - - ◦

◦ ◦

-

*release-acquire* `std::memory_order_consume` ◦ ◦

—  
Fences ◦ ◦

◦ ◦

## Examples

```
int x, y;
bool ready = false;

void init()
{
    x = 2;
    y = 3;
    ready = true;
}
void use()
{
    if (ready)
        std::cout << x + y;
}
```

init()use()◦ use()5◦

- CPU<sub>init()</sub>

```
int x, y;
bool ready = false;

void init()
{
    x = 2;
    y = 3;
    ready = true;
}
void use()
{
    if (ready)
        std::cout << x + y;
}
```

- CPU<sub>use()</sub>

```
int x, y;
bool ready = false;

void init()
```

```

{
    x = 2;
    y = 3;
    ready = true;
}
void use()
{
    if (ready)
        std::cout << x + y;
}

```

- C++。

init() use() o use() ready==true xy o

## C++。

```

int x, y;
bool ready = false;

void init()
{
    x = 2;
    y = 3;
    ready = true;
}
void use()
{
    if (ready)
        std::cout << x + y;
}

```

init() o true ready o

use() - o ready o

## CPU。

init() S use() o use() ready==true x==2 y==3 o

x CPU y use() o

```

int x, y;
std::atomic<bool> ready{false};

void init()
{
    x = 2;
    y = 3;
    atomic_thread_fence(std::memory_order_release);
    ready.store(true, std::memory_order_relaxed);
}
void use()
{
    if (ready.load(std::memory_order_relaxed))

```

```
{  
    atomic_thread_fence(std::memory_order_acquire);  
    std::cout << x + y;  
}  
}
```

xstd::cout°

◦

```
int x, y;  
std::atomic<bool> ready{false};  
  
void init()  
{  
    x = 2;  
    y = 3;  
    atomic_thread_fence(std::memory_order_release);  
    ready.store(true, std::memory_order_relaxed);  
}  
void use()  
{  
    if (ready.load(std::memory_order_relaxed))  
    {  
        atomic_thread_fence(std::memory_order_acquire);  
        std::cout << x + y;  
    }  
}
```

block\_and\_use() ready° °

C ++ 11 <https://riptutorial.com/zh-TW/cplusplus/topic/7975/c-plusplus-11>

# 4: C ++ Streams

```
std::istream_iterator<int> ifs, end;
std::copy(ifs, end, std::ostream_iterator<int>(cout));
cout << endl;
```

## Examples

```
std::ostringstream operator<< o
```

```
#include <sstream>
#include <string>

using namespace std;

int main()
{
    ostringstream ss;
    ss << "the answer to everything is " << 42;
    const string result = ss.str();
}
```

```
#include <sstream>
#include <string>

using namespace std;

int main()
{
    ostringstream ss;
    ss << "the answer to everything is " << 42;
    const string result = ss.str();
}
```

◦

```
#include <sstream>
#include <string>

using namespace std;

int main()
{
    ostringstream ss;
    ss << "the answer to everything is " << 42;
    const string result = ss.str();
}
```

```
#include <sstream>
#include <string>
```

```
using namespace std;

int main()
{
    ostringstream ss;
    ss << "the answer to everything is " << 42;
    const string result = ss.str();
}
```

result"the answer to everything is 42" .

◦

```
#include <sstream>
#include <string>

using namespace std;

int main()
{
    ostringstream ss;
    ss << "the answer to everything is " << 42;
    const string result = ss.str();
}
```

foo

```
#include <sstream>
#include <string>

using namespace std;

int main()
{
    ostringstream ss;
    ss << "the answer to everything is " << 42;
    const string result = ss.str();
}
```

```
#include <sstream>
#include <string>

using namespace std;

int main()
{
    ostringstream ss;
    ss << "the answer to everything is " << 42;
    const string result = ss.str();
}
```

resultfoo◦

ifstream o C++ o

o

ifstreamoperator bool() true o ifstream::operator >> EOF

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

ifstream::operator >> o std::getline ifstream::operator >> o getline

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

std::getline o

o

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

std::back\_inserter

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

ifs

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
```

```
    std::cout << s << std::endl;
}
```

---

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

## ◦ Boost.Asio

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

## STL◦

```
std::ifstream ifs("1.txt");
std::string s;
while(ifs >> s) {
    std::cout << s << std::endl;
}
```

## ◦ 100◦

## iostream

---

```
std::ostream_iterator<STL> std::ostream_iterator<std::ostream>
```

```
std::vector<int> v = {1,2,3,4};
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

```
std::vector<int> v = {1,2,3,4};
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

## std::ostream\_iterator◦ std::cout◦ 3

```
std::vector<int> v = {1,2,3,4};
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

```
float std::ostream_iterator int
```

```
std::vector<int> v = {1,2,3,4};  
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

```
std::vector<int> v = {1,2,3,4};  
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

std::vector<int> v = {1,2,3,4};  
std::copy(v.begin(), v.end(), std::ostream\_iterator<int>(std::cout, " ! "));

std::generate std::generate\_n std::transform  
std::vector<int> v = {1,2,3,4};  
std::copy(v.begin(), v.end(), std::ostream\_iterator<int>(std::cout, " ! "));

“x is even”

```
std::vector<int> v = {1,2,3,4};  
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

```
std::vector<int> v = {1,2,3,4};  
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

N

```
std::vector<int> v = {1,2,3,4};  
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

◦

```
std::vector<int> v = {1,2,3,4};  
std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " ! "));
```

**STL**

istream\_iterator STL◦

```
std::vector<int> v(100);  
std::copy(std::istream_iterator<int>(ifs), std::istream_iterator<int>(),  
v.begin());
```

```
std::vector<int> v(100);  
std::copy(std::istream_iterator<int>(ifs), std::istream_iterator<int>(),  
v.begin());
```

◦

```
istream::operator>>while◦
```

```
std::vector<int> v(100);
std::copy(std::istream_iterator<int>(ifs), std::istream_iterator<int>(),
v.begin());
```

```
std::vector<int> v(100);
std::copy(std::istream_iterator<int>(ifs), std::istream_iterator<int>(),
v.begin());
```

---

std::istream\_iterator◦ std::transform◦ **3.14**

```
std::vector<int> v(100);
std::copy(std::istream_iterator<int>(ifs), std::istream_iterator<int>(),
v.begin());
```

C ++ Streams <https://riptutorial.com/zh-TW/cplusplus/topic/7660/c-plusplus-streams>

# 5: C ++

C ++。

## Examples

◦ ◦

### §12.7.1◦

```
struct W { int j; };
struct X : public virtual W { };
struct Y {
    int *p;
    X x;
    Y() : p(&x.j) { // undefined, x is not yet constructed
    }
};
```

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/9885/c-plusplus>

# 6: C ++

## Examples

1. `sizeof(T) >= 1` ◦

```
class Base {};  
  
class Derived : public Base  
{  
public:  
    int i;  
};
```

Derived Base ◦ `sizeof(Derived) == sizeof(int)` ◦ C ++ ◦

Derived ◦ ◦

CC ++ - ◦

◦

- ◦ ◦
- 199
- 
- 
- /
- 
- 
- -
- API

◦ ◦

◦

```
void func(const A *a); // Some random function  
  
// useless memory allocation + deallocation for the instance  
auto a1 = std::make_unique<A>();  
func(a1.get());  
  
// making use of a stack object prevents  
auto a2 = A{};  
func(&a2);
```

C ++ 14

## C++ 14.

```
void func(const A *a); // Some random function

// useless memory allocation + deallocation for the instance
auto a1 = std::make_unique<A>();
func(a1.get());

// making use of a stack object prevents
auto a2 = A{};
func(&a2);
```

unique

```
void func(const A *a); // Some random function

// useless memory allocation + deallocation for the instance
auto a1 = std::make_unique<A>();
func(a1.get());

// making use of a stack object prevents
auto a2 = A{};
func(&a2);
```

/

**std::set** **std::vector** °

```
void func(const A *a); // Some random function

// useless memory allocation + deallocation for the instance
auto a1 = std::make_unique<A>();
func(a1.get());

// making use of a stack object prevents
auto a2 = A{};
func(&a2);
```

°

```
// This variant of stableUnique contains a complexity of N log(N)
// N > number of elements in v
// log(N) > insert complexity of std::set
std::vector<std::string> stableUnique(const std::vector<std::string> &v) {
    std::vector<std::string> result;
    std::set<std::string> checkUnique;
    for (const auto &s : v) {
        // See Optimizing by executing less code
        if (checkUnique.insert(s).second)
            result.push_back(s);
    }
    return result;
}
```

## N.std :: string。

```
// This variant of stableUnique contains a complexity of N log(N)
// N > number of elements in v
// log(N) > insert complexity of std::set
std::vector<std::string> stableUnique(const std::vector<std::string> &v) {
    std::vector<std::string> result;
    std::set<std::string> checkUnique;
    for (const auto &s : v) {
        // See Optimizing by executing less code
        if (checkUnique.insert(s).second)
            result.push_back(s);
    }
    return result;
}
```

std::string /o .

◦ ◦

---

```
#include <cstring>

class string final
{
    constexpr static auto SMALL_BUFFER_SIZE = 16;

    bool _isAllocated{false};                                ///<-- Remember if we allocated memory
    char *_buffer{nullptr};                                 ///<-- Pointer to the buffer we are using
    char _smallBuffer[SMALL_BUFFER_SIZE] = {'\0'};          ///<-- Stack space used for SMALL OBJECT
OPTIMIZATION

public:
    ~string()
    {
        if (_isAllocated)
            delete [] _buffer;
    }

    explicit string(const char *cStyleString)
    {
        auto stringSize = std::strlen(cStyleString);
        _isAllocated = (stringSize > SMALL_BUFFER_SIZE);
        if (_isAllocated)
            _buffer = new char[stringSize];
        else
            _buffer = &_smallBuffer[0];
        std::strcpy(_buffer, &cStyleString[0]);
    }

    string(string &&rhs)
        : _isAllocated(rhs._isAllocated)
        , _buffer(rhs._buffer)
        , _smallBuffer(rhs._smallBuffer) //< Not needed if allocated
    {
        if (_isAllocated)
        {
            // Prevent double deletion of the memory
    }
```

```
    rhs._buffer = nullptr;
}
else
{
    // Copy over data
    std::strcpy(_smallBuffer, rhs._smallBuffer);
    _buffer = &_smallBuffer[0];
}
// Other methods, including other constructors, copy constructor,
// assignment operators have been omitted for readability
};
```

new delete。。

buffer**bool** isAllocatedintel 648. . .

.. C++11 standard library `std::basic_string<>std::function<>`

2

## ◦ POD◦

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/4474/c-plusplus>

# 7: C ++

1

1

## Examples

## Google TestGoogleC ++。 qtest。

```
// main.cpp

#include <gtest/gtest.h>
#include <iostream>

// Google Test test cases are created using a C++ preprocessor macro
// Here, a "test suite" name and a specific "test name" are provided.
TEST(module_name, test_name) {
    std::cout << "Hello world!" << std::endl;
    // Google Test will also provide macros for assertions.
    ASSERT_EQ(1+1, 2);
}

// Google Test can be run manually from the main() function
// or, it can be linked to the gtest_main library for an already
// set-up main() function primed to accept Google Test test cases.
int main(int argc, char** argv) {
    ::testing::InitGoogleTest(&argc, argv);

    return RUN_ALL_TESTS();
}

// Build command: g++ main.cpp -lgtest
```

# CatchTDDBDD.

## Catch

```
SCENARIO( "vectors can be sized and resized", "[vector]" ) {
    GIVEN( "A vector with some items" ) {
        std::vector v( 5 );

        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 5 );

    WHEN( "the size is increased" ) {
        v.resize( 10 );

    THEN( "the size and capacity change" ) {
```

```

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }
}

WHEN( "the size is reduced" ) {
    v.resize( 0 );

    THEN( "the size changes but not capacity" ) {
        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );
    }
}

WHEN( "more capacity is reserved" ) {
    v.reserve( 10 );

    THEN( "the capacity changes but not the size" ) {
        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 10 );
    }
}

WHEN( "less capacity is reserved" ) {
    v.reserve( 0 );

    THEN( "neither size nor capacity are changed" ) {
        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 5 );
    }
}
}
}

```

```

SCENARIO( "vectors can be sized and resized", "[vector]" ) {
    GIVEN( "A vector with some items" ) {
        std::vector v( 5 );

        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 5 );

        WHEN( "the size is increased" ) {
            v.resize( 10 );

            THEN( "the size and capacity change" ) {
                REQUIRE( v.size() == 10 );
                REQUIRE( v.capacity() >= 10 );
            }
        }

        WHEN( "the size is reduced" ) {
            v.resize( 0 );

            THEN( "the size changes but not capacity" ) {
                REQUIRE( v.size() == 0 );
                REQUIRE( v.capacity() >= 5 );
            }
        }

        WHEN( "more capacity is reserved" ) {
            v.reserve( 10 );

            THEN( "the capacity changes but not the size" ) {
                REQUIRE( v.size() == 5 );

```

```
    REQUIRE( v.capacity() >= 10 );
}
}
WHEN( "less capacity is reserved" ) {
    v.reserve( 0 );

    THEN( "neither size nor capacity are changed" ) {
        REQUIRE( v.size() == 5 );
        REQUIRE( v.capacity() >= 5 );
    }
}
}
```

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/9928/c-plusplus>

# 8: C ++

C ++UTF-8UTF-16。ANSI / ASCII。

◦

Unicode◦

## Examples

### C ++

```
#include <iostream>
#include <string>

int main()
{
    const char * C_String = "This is a line of text w";
    const char * C_Problem_String = "This is a line of text Σ";
    std::string Std_String("This is a second line of text w");
    std::string Std_Problem_String("This is a second line of ΣεΣ Σ");

    std::cout << "String Length: " << Std_String.length() << '\n';
    std::cout << "String Length: " << Std_Problem_String.length() << '\n';

    std::cout << "CString Length: " << strlen(C_String) << '\n';
    std::cout << "CString Length: " << strlen(C_Problem_String) << '\n';
    return 0;
}
```

WindowsOSXGCCMSVC◦

Microsoft MSVC

```
31
31
CString24
CString24
```

MSVC◦

Unicode◦

GNC / GCC

```
31
36
CString24
CString26
```

LinuxGCC◦

Unicode“”。

- CC ++ **C ++char**。

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/5270/c-plusplus>

## 9: C++/

```
<iostream>
```

istream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

```
cin cin cout .
```

istream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

```
cin. ENTER.
```

istream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

```
cout
```

```
std::stream std::stream stream.
```

```
std::endl . . '\n' . .
```

## Examples

```
#include <iostream>

int main()
{
    int value;
    std::cout << "Enter a value: " << std::endl;
    std::cin >> value;
    std::cout << "The square of entered value is: " << value * value << std::endl;
    return 0;
}
```

C++/ <https://riptutorial.com/zh-TW/cplusplus/topic/10683/c-plusplus->

# 10: C ++

## Examples

### C ++

```
class listNode
{
public:
    int data;
    listNode *next;
    listNode(int val):data(val),next(NULL) {}
};
```

### List

```
class listNode
{
public:
    int data;
    listNode *next;
    listNode(int val):data(val),next(NULL) {}
};
```

```
class listNode
{
public:
    int data;
    listNode *next;
    listNode(int val):data(val),next(NULL) {}
};
```

```
class listNode
{
public:
    int data;
    listNode *next;
    listNode(int val):data(val),next(NULL) {}
};
```

```
class listNode
{
public:
    int data;
    listNode *next;
    listNode(int val):data(val),next(NULL) {}
};
```

```
class listNode
{
public:
```

```
int data;
listNode *next;
listNode(int val):data(val),next(NULL) {}
};
```

```
class listNode
{
public:
int data;
listNode *next;
listNode(int val):data(val),next(NULL) {}
};
```

```
class listNode
{
public:
int data;
listNode *next;
listNode(int val):data(val),next(NULL) {}
};
```

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/7485/c-plusplus>

# 11: C ++

## Examples

### Fibonacci

#### FibonacciN

```
int get_term_fib(int n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return get_term_fib(n - 1) + get_term_fib(n - 2);
}
```

n ◦ ◦

```
int get_term_fib(int n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return get_term_fib(n - 1) + get_term_fib(n - 2);
}
```

#### Fibonacci<sub>n◦</sub>

#### memoization

◦ ◦

#### memoizationFibonacci

```
#include <map>

int fibonacci(int n)
{
    static std::map<int, int> values;
    if (n==0 || n==1)
        return n;
    std::map<int,int>::iterator iter = values.find(n);
    if (iter == values.end())
    {
        return values[n] = fibonacci(n-1) + fibonacci(n-2);
    }
    else
    {
        return iter->second;
    }
}
```

```
}
```

\$ On\$。 \$ O1\$。

◦ ◦

```
#include <map>

int fibonacci(int n)
{
    static std::map<int, int> values;
    if (n==0 || n==1)
        return n;
    std::map<int,int>::iterator iter = values.find(n);
    if (iter == values.end())
    {
        return values[n] = fibonacci(n-1) + fibonacci(n-2);
    }
    else
    {
        return iter->second;
    }
}
```

◦ ◦ ;◦

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/5693/c-plusplus>

# 12: C ++“”””

- 
- 
- 

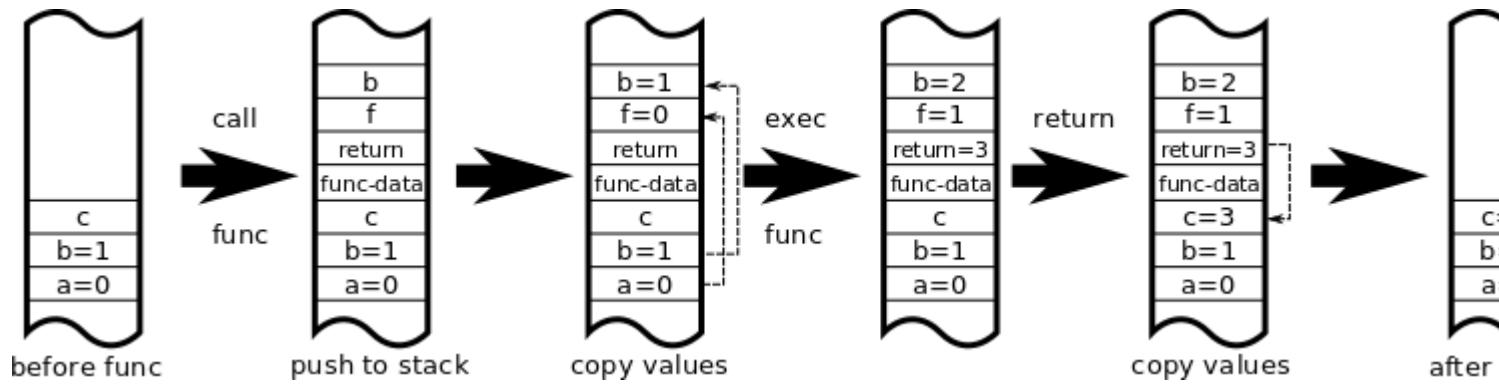
## Examples

- ◦
- ◦ ◦
- ◦

```
int func(int f, int b) {  
    //new variables are created and values from the outside copied  
    //f has a value of 0  
    //inner_b has a value of 1  
    f = 1;  
    //f has a value of 1  
    b = 2;  
    //inner_b has a value of 2  
    return f+b;  
}  
  
int main(void) {  
    int a = 0;  
    int b = 1; //outer_b  
    int c;  
  
    c = func(a,b);  
    //the return value is copied to c  
  
    //a has a value of 0  
    //outer_b has a value of 1     <--- outer_b and inner_b are different variables  
    //c has a value of 3  
}
```

main◦◦ finner\_b b◦ a<->fb<->b◦

varibale b◦◦



“”。

C ++“””” <https://riptutorial.com/zh-TW/cplusplus/topic/10669/c-plusplus---->

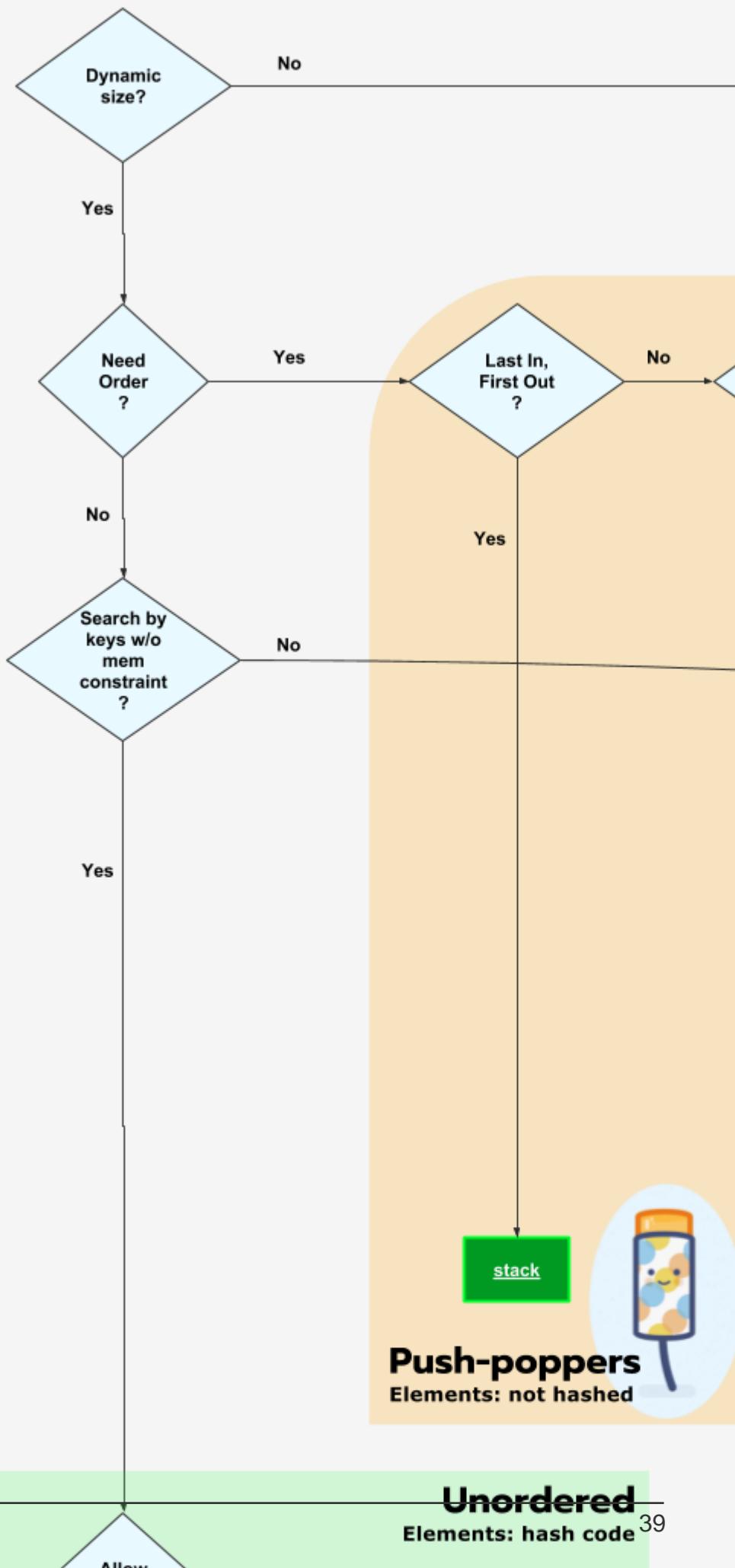
# 13: C ++

C ++。 C ++int MyClass。

## Examples

C ++

## C ++Container。



**Push-poppers**  
Elements: not hashed

**Unordered**  
Elements: hash code

# 14: C ++

C ++◦◦◦

/

- 
- UBSanTSanMSanESan .....
- CFI .....
- 

## Examples

### C ++segfault - valgrind

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

-g

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
```

```

int *p3 = p1;
if (p3) {
    std::cout << *p3 << std::endl;
}
}

int main() {
    fail();
}

```

## valgrind

```

#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}

```

```

#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}

```

## 4. main.cpp8 fail() main.cpp main 13°

### main.cpp8

```

#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

```

```
int main() {
    fail();
}
```

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

7

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

p3p2◦ p3

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

Valgrind--track-origins=yes

```
#include <iostream>
```

```
void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

valgrindvalgrind..

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

73

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p3 << std::endl;
    }
}

int main() {
    fail();
}
```

# GDBSegfault

```
#include <iostream>
```

```
void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p2 << std::endl;
    }
}

int main() {
    fail();
}
```

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p2 << std::endl;
    }
}

int main() {
    fail();
}
```

## gdb

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p2 << std::endl;
    }
}

int main() {
    fail();
}
```

## gdb shell。 run。

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p2 << std::endl;
    }
}
```

```
int main() {
    fail();
}
```

11。 p2。。

```
#include <iostream>

void fail() {
    int *p1;
    int *p2(NULL);
    int *p3 = p1;
    if (p3) {
        std::cout << *p2 << std::endl;
    }
}

int main() {
    fail();
}
```

p20x0NULL。 NULL。。

。

```
int main() {
    int value;
    std::vector<int> vectorToSort;
    vectorToSort.push_back(42); vectorToSort.push_back(13);
    for (int i = 52; i; i = i - 1)
    {
        vectorToSort.push_back(i * 2);
    }
    /// Optimized for sorting small vectors
    if (vectorToSort.size() == 1);
    else
    {
        if (vectorToSort.size() <= 2)
            std::sort(vectorToSort.begin(), std::end(vectorToSort));
    }
    for (value : vectorToSort) std::cout << value << ' ';
    return 0;
}
```

```
int main() {
    int value;
    std::vector<int> vectorToSort;
    vectorToSort.push_back(42); vectorToSort.push_back(13);
    for (int i = 52; i; i = i - 1)
    {
        vectorToSort.push_back(i * 2);
    }
    /// Optimized for sorting small vectors
    if (vectorToSort.size() == 1);
    else
    {
        if (vectorToSort.size() <= 2)
            std::sort(vectorToSort.begin(), std::end(vectorToSort));
```

```
    }
    for (value : vectorToSort) std::cout << value << ' ';
    return 0;
}
```

◦

◦ ◦

◦ main() 3◦

bug◦

/

◦ "◦ ◦

◦ IDE◦

/◦ v.begin() VS std::end(v) ◦

◦

- optimized
- sortVector()
- std::ref() sortVector()

◦ ◦ PS2◦

◦ ◦

if if (var); ◦

- clang++ -Wall -Weverything -Werror ...
- g++ -Wall -Weverything -Werror ...
- cl.exe /W4 /WX ...

◦ while (staticAtomicBool);while (localBool);while (localBool);◦

◦ ◦

◦ C ++ 98

- clang++ -Wall -Weverything -Werror -Wno-error-to-accept ...
- g++ -Wall -Weverything -Werror -Wno-error-to-accept ...
- cl.exe /W4 /WX /wd<no of warning>...

◦ ◦ ◦

◦ clang-tidy

- ○
- ○
- ○
- ○
- ○ make\_unique
- ○ nullptr
- ○ ○

Clang○

- visual studio
- clazy QtClang

C ++○ ○

bug○ ○

○ ○ ○

○ ○

/○ ○

bug○ CPI ○ ○

CPU<1○

clang -fsanitize=safe-stack○ GCC○

○ ○

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/9814/c-plusplus>

# 15: constexpr

constexpr C++ 17 if。

constexpr C++ 11 C++ 11。 C++ 11。 C++ 14 constexpr。

## Examples

### constexpr

constexpr const。

#define

constexpr#define。 constexpr。

### C++ 11

```
int main()
{
    constexpr int N = 10 + 2;
    cout << N;
}
```

```
int main()
{
    constexpr int N = 10 + 2;
    cout << N;
}
```

◦

```
int main()
{
    constexpr int N = 10 + 2;
    cout << N;
}
```

```
int main()
{
    constexpr int N = 10 + 2;
    cout << N;
}
```

cout << 10 + 2; ◦ ◦ ◦

#define

```
int main()
```

```
{  
    constexpr int N = 10 + 2;  
    cout << N;  
}
```

```
int main()  
{  
    constexpr int N = 10 + 2;  
    cout << N;  
}
```

constexpr24 °

### const

const ° constexpr° constexpr° const°

```
int main()  
{  
    constexpr int N = 10 + 2;  
    cout << N;  
}
```

GCC° ° size210 °

const° const° #define°

### C++ 11

```
int main()  
{  
    constexpr int N = 10 + 2;  
    cout << N;  
}
```

constexpr°

### C++ 11

```
int main()  
{  
    constexpr int N = 10 + 2;  
    cout << N;  
}
```

abs constexpr °

constexpr°

### C++ 11

```
int main()  
{
```

```
    constexpr int N = 10 + 2;
    cout << N;
}
```

auto

## C++ 11

```
int main()
{
    constexpr int N = 10 + 2;
    cout << N;
}
```

## constexpr

constexpr

## C++ 11

```
constexpr int Sum(int a, int b)
{
    return a + b;
}
```

constexpr

## C++ 11

```
constexpr int Sum(int a, int b)
{
    return a + b;
}
```

constexpr SSconstexpr**const** Sumconstexpr °

constexpr

## C++ 11

```
constexpr int Sum(int a, int b)
{
    return a + b;
}
```

Sumconstexpr ° °

**const**

## C++ 11

```
constexpr int Sum(int a, int b)
{
```

```
    return a + b;  
}
```

constexpr Sumconstexpr Rconstexpr Lconstexpr .

---

constexpr.

## C++ 11

```
constexpr int Sum(int a, int b)  
{  
    return a + b;  
}
```

a1constexpr constexpr constexpr-a

## C++ 11

```
constexpr int Sum(int a, int b)  
{  
    return a + b;  
}
```

## C++ 11

```
constexpr int Sum(int a, int b)  
{  
    return a + b;  
}
```

abs(a) abs(10) absconstexpr int

## C++ 11

```
constexpr int Sum(int a, int b)  
{  
    return a + b;  
}
```

Absconstexpr Abs . Sum constexpr . Sum(-10, 20) 30 .

if

## C++ 17

if constexpr . . .

```
template<class T, class ... Rest>  
void g(T &&p, Rest &&...rs)  
{  
    // ... handle p
```

```
if constexpr (sizeof...(rs) > 0)
    g(rs...); // never instantiated with an empty argument list
}
```

```
return 0;
```

```
if constexpr#ifndef  #ifdef  #ifdef  if constexpr#ifndef
```

```
template<class T, class ... Rest>
void g(T &&p, Rest &&...rs)
{
    // ... handle p
    if constexpr (sizeof...(rs) > 0)
        g(rs...); // never instantiated with an empty argument list
}
```

constexpr <https://riptutorial.com/zh-TW/cplusplus/topic/3899/constexpr>

# 16: Const

- class ClassOne {public bool non\_modifying\_member\_functionconst /\* ... \*/;}
- int ClassTwo :: non\_modifying\_member\_functionconst /\* ... \*/
- void ClassTwo :: modification\_member\_function/\* ... \*/
- char non\_param\_modding\_funcconst ClassOneoneconst ClassTwo \* two/\* ... \*/
- float parameter\_modifying\_functionClassTwooneClassOne \* two/\* ... \*/
- short ClassThree :: non\_modding\_non\_param\_modding\_fconst ClassOneconst /\* ... \*/

const. Const Correct Function Parameters Const Correct Function Parameters.

constconst. const const.

volatileconst.

## ISO\_CPP

### C++

## Examples

const. . constconst.

const CV-qualifiers /const.

```
class ConstCorrectClass {
    int x;

    public:
        int getX() const { return x; } // Function is const: Doesn't modify instance.
        void setX(int i) { x = i; } // Not const: Modifies instance.
};

// Parameter is const: Doesn't modify parameter.
int const_correct_reader(const ConstCorrectClass& c) {
    return c.getX();
}

// Parameter isn't const: Modifies parameter.
void const_correct_writer(ConstCorrectClass& c) {
    c.setX(42);
}

const ConstCorrectClass invariant; // Instance is const: Can't be modified.
ConstCorrectClass variant; // Instance isn't const: Can be modified.

// ...

const_correct_reader(invariant); // Good. Calling non-modifying function on const instance.
const_correct_reader(variant); // Good. Calling non-modifying function on modifiable
instance.
```

```
const_correct_writer(variant); // Good. Calling modifying function on modifiable instance.  
const_correct_writer(invariant); // Error. Calling modifying function on const instance.
```

**const;constconstconstconstconst** **CV**◦

## Const

const -correct this **CV**const **mutable**const ;const **cv-qualified**const 。 **CV**const T\*T\*const T\* 。 const 。

constconstConst Correct Function Parameters◦

this **cv-qualifiers**

```
// Assume class Field, with member function "void insert_value(int);".  
  
class ConstIncorrect {  
    Field fld;  
  
public:  
    ConstIncorrect(Field& f); // Modifies.  
  
    Field& getField(); // Might modify. Also exposes member as non-const reference,  
                       // allowing indirect modification.  
    void setField(Field& f); // Modifies.  
  
    void doSomething(int i); // Might modify.  
    void doNothing(); // Might modify.  
};  
  
ConstIncorrect::ConstIncorrect(Field& f) : fld(f) {} // Modifies.  
Field& ConstIncorrect::getField() { return fld; } // Doesn't modify.  
void ConstIncorrect::setField(Field& f) { fld = f; } // Modifies.  
void ConstIncorrect::doSomething(int i) { // Modifies.  
    fld.insert_value(i);  
}  
void ConstIncorrect::doNothing() {} // Doesn't modify.  
  
  
class ConstCorrectCVQ {  
    Field fld;  
  
public:  
    ConstCorrectCVQ(Field& f); // Modifies.  
  
    const Field& getField() const; // Doesn't modify. Exposes member as const reference,  
                                  // preventing indirect modification.  
    void setField(Field& f); // Modifies.  
  
    void doSomething(int i); // Modifies.  
    void doNothing() const; // Doesn't modify.  
};  
  
ConstCorrectCVQ::ConstCorrectCVQ(Field& f) : fld(f) {}  
Field& ConstCorrectCVQ::getField() const { return fld; }  
void ConstCorrectCVQ::setField(Field& f) { fld = f; }  
void ConstCorrectCVQ::doSomething(int i) {  
    fld.insert_value(i);  
}
```

```

void ConstCorrectCVQ::doNothing() const {}  

// This won't work.  

// No member functions can be called on const ConstIncorrect instances.  

void const_correct_func(const ConstIncorrect& c) {  

    Field f = c.getField();  

    c.do_nothing();  

}  

// But this will.  

// getField() and doNothing() can be called on const ConstCorrectCVQ instances.  

void const_correct_func(const ConstCorrectCVQ& c) {  

    Field f = c.getField();  

    c.do_nothing();  

}

```

Const Correct Function Parameters **const -correct**.

```

// Assume class Field, with member function "void insert_value(int);".  

class ConstIncorrect {  

    Field fld;  

public:  

    ConstIncorrect(Field& f); // Modifies.  

    Field& getField();           // Might modify. Also exposes member as non-const reference,  

                                // allowing indirect modification.  

    void setField(Field& f);   // Modifies.  

    void doSomething(int i);    // Might modify.  

    void doNothing();          // Might modify.  

};  

ConstIncorrect::ConstIncorrect(Field& f) : fld(f) {} // Modifies.  

Field& ConstIncorrect::getField() { return fld; } // Doesn't modify.  

void ConstIncorrect::setField(Field& f) { fld = f; } // Modifies.  

void ConstIncorrect::doSomething(int i) {           // Modifies.  

    fld.insert_value(i);  

}  

void ConstIncorrect::doNothing() {}                  // Doesn't modify.  

class ConstCorrectCVQ {  

    Field fld;  

public:  

    ConstCorrectCVQ(Field& f); // Modifies.  

    const Field& getField() const; // Doesn't modify. Exposes member as const reference,  

                                // preventing indirect modification.  

    void setField(Field& f);   // Modifies.  

    void doSomething(int i);    // Modifies.  

    void doNothing() const;    // Doesn't modify.  

};  

ConstCorrectCVQ::ConstCorrectCVQ(Field& f) : fld(f) {}  

Field& ConstCorrectCVQ::getField() const { return fld; }  

void ConstCorrectCVQ::setField(Field& f) { fld = f; }

```

```

void ConstCorrectCVQ::doSomething(int i) {
    fld.insert_value(i);
}
void ConstCorrectCVQ::doNothing() const {} 

// This won't work.
// No member functions can be called on const ConstIncorrect instances.
void const_correct_func(const ConstIncorrect& c) {
    Field f = c.getField();
    c.do_nothing();
}

// But this will.
// getField() and doNothing() can be called on const ConstCorrectCVQ instances.
void const_correct_func(const ConstCorrectCVQ& c) {
    Field f = c.getField();
    c.do_nothing();
}

```

**constconst;containersconst**。

```

// Assume class Field, with member function "void insert_value(int);".

class ConstIncorrect {
    Field fld;

public:
    ConstIncorrect(Field& f); // Modifies.

    Field& getField();          // Might modify. Also exposes member as non-const reference,
                                // allowing indirect modification.
    void setField(Field& f);   // Modifies.

    void doSomething(int i);    // Might modify.
    void doNothing();          // Might modify.
};

ConstIncorrect::ConstIncorrect(Field& f) : fld(f) {} // Modifies.
Field& ConstIncorrect::getField() { return fld; }      // Doesn't modify.
void ConstIncorrect::setField(Field& f) { fld = f; } // Modifies.
void ConstIncorrect::doSomething(int i) {               // Modifies.
    fld.insert_value(i);
}
void ConstIncorrect::doNothing() {}                     // Doesn't modify.

class ConstCorrectCVQ {
    Field fld;

public:
    ConstCorrectCVQ(Field& f); // Modifies.

    const Field& getField() const; // Doesn't modify. Exposes member as const reference,
                                // preventing indirect modification.
    void setField(Field& f);    // Modifies.

    void doSomething(int i);    // Modifies.
    void doNothing() const;    // Doesn't modify.
};

```

```

ConstCorrectCVQ::ConstCorrectCVQ(Field& f) : fld(f) {}
Field& ConstCorrectCVQ::getField() const { return fld; }
void ConstCorrectCVQ::setField(Field& f) { fld = f; }
void ConstCorrectCVQ::doSomething(int i) {
    fld.insert_value(i);
}
void ConstCorrectCVQ::doNothing() const {}

// This won't work.
// No member functions can be called on const ConstIncorrect instances.
void const_correct_func(const ConstIncorrect& c) {
    Field f = c.getField();
    c.do_nothing();
}

// But this will.
// getField() and doNothing() can be called on const ConstCorrectCVQ instances.
void const_correct_func(const ConstCorrectCVQ& c) {
    Field f = c.getField();
    c.do_nothing();
}

```

const .

## Const

const -correct const. const CV -this const T\*T“。

```

struct Example {
    void func()          { std::cout << 3 << std::endl; }
    void func() const { std::cout << 5 << std::endl; }
};

void const_incorrect_function(Example& one, Example* two) {
    one.func();
    two->func();
}

void const_correct_function(const Example& one, const Example* two) {
    one.func();
    two->func();
}

int main() {
    Example a, b;
    const_incorrect_function(a, &b);
    const_correct_function(a, &b);
}

// Output:
3
3
5
5

```

const const -correct const -incorrect const -correct const -incorrect const const. [ const -incorrect const

```

const .

struct Example {
    void func()          { std::cout << 3 << std::endl; }
    void func() const { std::cout << 5 << std::endl; }
};

void const_incorrect_function(Example& one, Example* two) {
    one.func();
    two->func();
}

void const_correct_function(const Example& one, const Example* two) {
    one.func();
    two->func();
}

int main() {
    Example a, b;
    const_incorrect_function(a, &b);
    const_correct_function(a, &b);
}

// Output:
3
3
5
5
5

```

## Const

const. const const.

## ~~const~~ CV-Qualified Member Functions

- const
  - ◦ mutable◦
  - mutable◦
- const
  - ◦
  - ◦

```

// ConstMemberFunctions.h

class ConstMemberFunctions {
    int val;
    mutable int cache;
    mutable bool state_changed;

public:
    // Constructor clearly changes logical state. No assumptions necessary.
    ConstMemberFunctions(int v = 0);

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val(). It may or may not call squared_calc() or bad_func().

```

```

int calc() const;

// We can assume this function doesn't change logical state, and doesn't call
// set_val(). It may or may not call calc() or bad_func().
int squared_calc() const;

// We can assume this function doesn't change logical state, and doesn't call
// set_val(). It may or may not call calc() or squared_calc().
void bad_func() const;

// We can assume this function changes logical state, and may or may not call
// calc(), squared_calc(), or bad_func().
void set_val(int v);
};

const.

```

```

// ConstMemberFunctions.h

class ConstMemberFunctions {
    int val;
    mutable int cache;
    mutable bool state_changed;

public:
    // Constructor clearly changes logical state. No assumptions necessary.
    ConstMemberFunctions(int v = 0);

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val(). It may or may not call squared_calc() or bad_func().
    int calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val(). It may or may not call calc() or bad_func().
    int squared_calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val(). It may or may not call calc() or squared_calc().
    void bad_func() const;

    // We can assume this function changes logical state, and may or may not call
    // calc(), squared_calc(), or bad_func().
    void set_val(int v);
};

const.

```

- const
  - ◦
  - /◦
- const
  - ◦
  - /◦

```

// ConstMemberFunctions.h

class ConstMemberFunctions {
    int val;
    mutable int cache;
    mutable bool state_changed;

public:
    // Constructor clearly changes logical state.  No assumptions necessary.
    ConstMemberFunctions(int v = 0);

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call squared_calc() or bad_func().
    int calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call calc() or bad_func().
    int squared_calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call calc() or squared_calc().
    void bad_func() const;

    // We can assume this function changes logical state, and may or may not call
    // calc(), squared_calc(), or bad_func().
    void set_val(int v);
};


```

const.

```

// ConstMemberFunctions.h

class ConstMemberFunctions {
    int val;
    mutable int cache;
    mutable bool state_changed;

public:
    // Constructor clearly changes logical state.  No assumptions necessary.
    ConstMemberFunctions(int v = 0);

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call squared_calc() or bad_func().
    int calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call calc() or bad_func().
    int squared_calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call calc() or squared_calc().
    void bad_func() const;

    // We can assume this function changes logical state, and may or may not call
    // calc(), squared_calc(), or bad_func().
    void set_val(int v);
};


```

---

const

Machiavelli。

```
// ConstMemberFunctions.h

class ConstMemberFunctions {
    int val;
    mutable int cache;
    mutable bool state_changed;

public:
    // Constructor clearly changes logical state.  No assumptions necessary.
    ConstMemberFunctions(int v = 0);

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call squared_calc() or bad_func().
    int calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call calc() or bad_func().
    int squared_calc() const;

    // We can assume this function doesn't change logical state, and doesn't call
    // set_val().  It may or may not call calc() or squared_calc().
    void bad_func() const;

    // We can assume this function changes logical state, and may or may not call
    // calc(), squared_calc(), or bad_func().
    void set_val(int v);
};

const const。
```

Const <https://riptutorial.com/zh-TW/cplusplus/topic/7217/const>

# 17: const

- `const myVariable = initial; //const;`
- `const Type myReference = myVariable; //const`
- `const Type * myPointer = myVariable; //const`
- `* const myPointer = myVariable; //const`
- `const Type * const myPointer = myVariable; //constconst`

`const1 ° const°`

`1const_cast`

## Examples

### Const

◦

```
// a is const int, so it can't be changed
const int a = 15;
a = 12;           // Error: can't assign new value to const variable
a += 1;           // Error: can't assign new value to const variable
```

```
// a is const int, so it can't be changed
const int a = 15;
a = 12;           // Error: can't assign new value to const variable
a += 1;           // Error: can't assign new value to const variable
```

### Const

```
int a = 0, b = 2;

const int* pA = &a; // pointer-to-const. `a` can't be changed through this
int* const pB = &a; // const pointer. `a` can be changed, but this pointer can't.
const int* const pC = &a; // const pointer-to-const.

//Error: Cannot assign to a const reference
*pA = b;

pA = &b;

*pB = b;

//Error: Cannot assign to const pointer
pB = &b;

//Error: Cannot assign to a const reference
*pC = b;

//Error: Cannot assign to const pointer
```

```
pC = &b;
```

## Const

const

```
class MyClass
{
private:
    int myInt_;
public:
    int myInt() const { return myInt_; }
    void setMyInt(int myInt) { myInt_ = myInt; }
};
```

const thisconst MyClass \*MyClass \* . ;° setMyIntconst °

const ° const MyClassconst °

staticconst ° ;° staticconst °

## constconst getter °

C ++const ° getter °

FooBar

```
class Foo
{
public:
    Bar& GetBar(/* some arguments */)
    {
        /* some calculations */
        return bar;
    }

    const Bar& GetBar(/* some arguments */) const
    {
        /* some calculations */
        return bar;
    }

    // ...
};
```

constconstconstconst °

° constconst ° constconst ° 'const\_cast'const °

```
class Foo
{
public:
    Bar& GetBar(/* some arguments */)
```

```
{  
    /* some calculations */  
    return bar;  
}  
  
const Bar& GetBar(/* some arguments */) const  
{  
    /* some calculations */  
    return bar;  
}  
  
// ...  
};
```

**constconst** GetBar**const**GetBar const\_cast<const Foo\*>(this) 。 **constconstconstconst**。

```
class Foo  
{  
public:  
    Bar& GetBar(/* some arguments */)  
    {  
        /* some calculations */  
        return bar;  
    }  
  
    const Bar& GetBar(/* some arguments */) const  
    {  
        /* some calculations */  
        return bar;  
    }  
  
    // ...  
};
```

**const** <https://riptutorial.com/zh-TW/cplusplus/topic/2386/const>

# 18: C

CC ++。

## Examples

C ++CC ++。

```
int class = 5
```

◦

C\_void\* C ++。 C ++C

```
void* ptr;
int* intptr = ptr;
```

◦

C ++gotoswitch。 CC ++

```
goto foo;
int skipped = 1;
foo;
```

◦

C <https://riptutorial.com/zh-TW/cplusplus/topic/9645/c>

# 19: decltype

decltype◦

## Examples

◦

```
int a = 10;

// Assume that type of variable 'a' is not known here, or it may
// be changed by programmer (from int to long long, for example).
// Hence we declare another variable, 'b' of the same type using
// decltype keyword.
decltype(a) b; // 'decltype(a)' evaluates to 'int'
```

“a”

float a=99.0f;

bfloat ◦

```
std::vector<int> intVector;
```

◦ auto ◦ ◦

```
std::vector<int> intVector;
```

decltype intVector◦

```
std::vector<int> intVector;
```

begin vector<int>::iterator ◦

const\_iterator cbegin

```
std::vector<int> intVector;
```

decltype <https://riptutorial.com/zh-TW/cplusplus/topic/9930/decltype>

# 20: ISO C ++

1998C ++。 C ++C ++。 。 。

C ++“”。

C ++。 Bjarne Stroustrup90。 C ++C ++。 。

C 。 ANSI1989。 ISOisos。 。

C ++。 ISO22WG21。 1995。 C ++。 1995STLWG21C ++。 3。 ISOC ++ISO / IEC 14882。

148821998。

◦ C ++。 2003148822003。 C ++;◦

C ++ 0x20082009。 - 148822011。

WG21。 C ++ 11C ++。 3148822014。

◦ C ++ 17C ++ 20◦

## Examples

ISOISO <http://www.iso.org>◦ C ++◦

- C ++ 20C ++ 2a [HTML](#)
- C ++ 17C ++ 1z [20173N4659](#)◦
- C ++ 14C ++ 1y [201411N4296](#)
- C ++ 11C ++ 0x [20112N3242](#)
- C ++ 03
- C ++ 98

## C ++ 11

C ++ 11C ++◦ [isocpp](#)◦

- 
- - [decltype](#)
  - 
  - 
  - 
  - [Rvalue](#)
  - [Lambda](#)
  - [no](#)
  - [constexpr](#)

- `nullptr` -
  - 
  - 
  - 
  - 
  - `= default= delete`
  - 
  - 
  - 
  - 
  - 
  - `final`
  - 
  - 
  - 
  - `long long` -
  - 
  - 
  - `POD`
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - 
  - `__cplusplusC ++ 11`
  - 
  - 
  - 
  - 
  - `static_assert`
  - 
  - 
  - 
  - 
  - `C99`
- 

- `unique_ptr`
- `shared_ptr`
- `weak_ptr`
- `ABI`
- 
- 
- 
- 
- 
- 
- 
- 
-

- 
- `unordered_*`
- `std::`
- `Modifiers`

•  
•  
•  
•  
•  
•  
•  
•

## C++ 14

C++ 14 C++ 11。C++ 11。isocpp。

---

- 
- 
- `decltype`
- `lambda`
- `lambdas`
- 
- `constexpr`
- [ [deprecated] ]
- 

- 
- - `std::types`
  - `std::make_unique`
  - `_t`
  - `get<string>(t)`
  - `greater<>(x)`
  - `std::quoted`



- `std::gets` C++ 11 C++ 14
- `std::random_shuffle`

## C++ 17

C++ 17。C++ 1z。

---

•

- `auto`
  - 
  - 
  - 
  - 
  - 
  - `[[fallthrough]] [[nodiscard]] [[maybe_unused]]`
  - `static_assert`
  - `ifswitchswitch`
  - 
  - `if constexpr`
  - 
  -

- std:
- std:
- std:
- merge
- <fill>
- <alg>
- <cma>
- map

C ++ 03C ++ 98。。

C ++ 98

C ++ 98C ++。 CC ++C。

# C89 / C90

- const
  - 
  - C-languagueC99
  - 
  - 
  - C-languagueC99
  - 
  - 
  - 
  -

## C ++ 20

C ++ 20C ++C ++ 17。 ISO cpp。

2020C ++。



◦

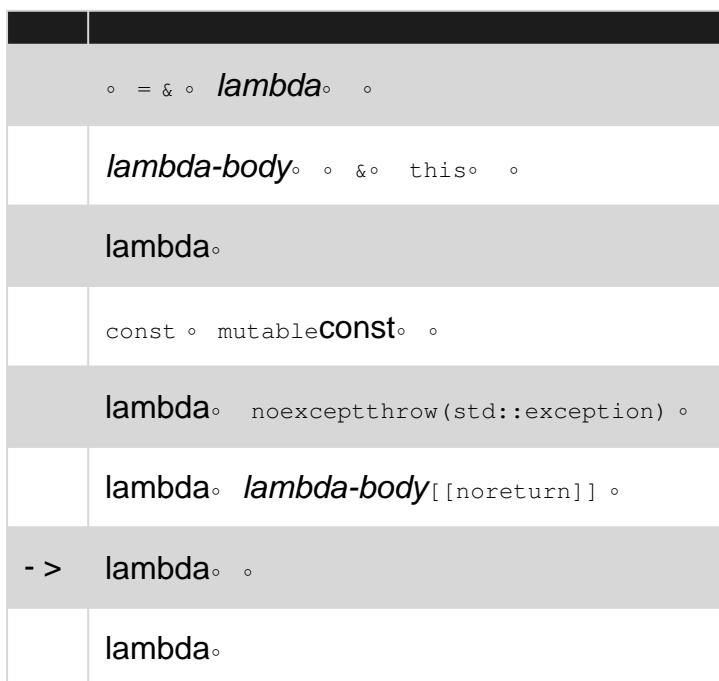


◦

ISO C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/2742/iso-c-plusplus>

# 21: Lambda

- [ default-capture capture-list ] argument-list mutable throw-specification -> return-type { lambda-body } // lambda.
- [ capture-list ] argument-list { lambda-body } //lambda.
- [=] argument-list { lambda-body } //.
- [] argument-list { lambda-body } //.
- [ capture-list ] { lambda-body } //.



C ++ 17 constexpr lambdas lambdas lambda constexpr lambda constexpr constexpr

```
//Explicitly define this lambdas as constexpr
[]() constexpr {
    //Do stuff
}
```

## Examples

### lambda

lambda lambda prvalue .

"lambda" lambda Alonzo Church 2030 Lambda LISP LISP lambda LISPC ++ lambda .

lambda . . lambda .

lambda [] () {}

```
[] () {} // An empty lambda, which does and returns nothing
```

---

## [] . **lambda** . **lambda** . **lambda**;**lambda**.

```
[] () {} // An empty lambda, which does and returns nothing
```

## () . **lambda****lambda** **mutable** . **lambda**s

```
[] () {} // An empty lambda, which does and returns nothing
```

## C++ 14

### auto . **lambda**s

```
[] () {} // An empty lambda, which does and returns nothing
```

```
{ } .
```

---

## **lambda**

### **lambda** **operator()**

```
[] () {} // An empty lambda, which does and returns nothing
```

---

## **lambda**.

```
[] () {} // An empty lambda, which does and returns nothing
```

```
bool .
```

```
[] () {} // An empty lambda, which does and returns nothing
```

---

## **Lambda**

### **lambda**. **operator()****const** .

```
[] () {} // An empty lambda, which does and returns nothing
```

```
mutablecloseoperator()const
```

```
[] () {} // An empty lambda, which does and returns nothing
```

```
mutable.
```

```
[] () {} // An empty lambda, which does and returns nothing
```

## lambda

C ++ 11

C ++ 11

```
[] () {} // An empty lambda, which does and returns nothing
```

C ++ 11

C ++ 11

```
[] () {} // An empty lambda, which does and returns nothing
```

## return lambdas return

```
// Returns bool, because "value > 10" is a comparison which yields a Boolean result
auto l = [](int value) {
    return value > 10;
}
```

## return lambda

```
// Returns bool, because "value > 10" is a comparison which yields a Boolean result
auto l = [](int value) {
    return value > 10;
}
```

```
// Returns bool, because "value > 10" is a comparison which yields a Boolean result
auto l = [](int value) {
    return value > 10;
}
```

## auto<sup>o</sup> Lambdas

```
// Returns bool, because "value > 10" is a comparison which yields a Boolean result
auto l = [](int value) {
    return value > 10;
}
```

## lambda<sup>o</sup> lambda<sup>o</sup>

```
int a = 0;

[a]() {
    return a; // Ok, 'a' is captured by value
};
```

C ++ 14

```
int a = 0;

[a]() {
    return a;    // Ok, 'a' is captured by value
};
```

## C++ 14. lambda.

### C++ 14

```
int a = 0;

[a]() {
    return a;    // Ok, 'a' is captured by value
};
```

## lambda lambda. lambda operator() const.

const

```
int a = 0;

[a]() {
    return a;    // Ok, 'a' is captured by value
};
```

## const lambda<sub>mutable</sub>

```
int a = 0;

[a]() {
    return a;    // Ok, 'a' is captured by value
};
```

## lambda<sub>a</sub> . aλaa.

```
int a = 0;

[a]() {
    return a;    // Ok, 'a' is captured by value
};
```

### C++ 14

## Lambdas. lambdas

```
auto p = std::make_unique<T>(...);

auto lamb = [p = std::move(p)]() //Overrides capture-by-value of `p`.
{
    p->SomeFunc();
};
```

**plambda**`p` `lambmake_unique``lamb`

```
auto p = std::make_unique<T>(...);

auto lamb = [p = std::move(p)]() //Overrides capture-by-value of `p`.
{
    p->SomeFunc();
};
```

`lamb_move`

---

`std::function<>``std::function` **lambda**`shared_ptr`

```
auto p = std::make_unique<T>(...);

auto lamb = [p = std::move(p)]() //Overrides capture-by-value of `p`.
{
    p->SomeFunc();
};
```

**move-only lambda**`lambda``std::function`

---

`auto`

```
auto p = std::make_unique<T>(...);

auto lamb = [p = std::move(p)]() //Overrides capture-by-value of `p`.
{
    p->SomeFunc();
};
```

**Generalize**

```
auto p = std::make_unique<T>(...);

auto lamb = [p = std::move(p)]() //Overrides capture-by-value of `p`.
{
    p->SomeFunc();
};
```

**lambda**`lambda``lambda`

`&` `lambda``lambda``lambda`

```
// Declare variable 'a'
int a = 0;

// Declare a lambda which captures 'a' by reference
auto set = [&a]() {
    a = 1;
};
```

```
set();  
assert(a == 1);
```

mutable aconst。

◦ lambda◦ lambda◦

lambda◦ lambda

```
int a = 1;  
int b = 2;  
  
// Default capture by value  
[=] () { return a + b; }; // OK; a and b are captured by value  
  
// Default capture by reference  
[&] () { return a + b; }; // OK; a and b are captured by reference
```

◦

```
int a = 1;  
int b = 2;  
  
// Default capture by value  
[=] () { return a + b; }; // OK; a and b are captured by value  
  
// Default capture by reference  
[&] () { return a + b; }; // OK; a and b are captured by reference
```

## lambdas

### C++ 14

Lambda◦ lambda

```
auto twice = [] (auto x){ return x+x; };  
  
int i = twice(2); // i == 4  
std::string s = twice("hello"); // s == "hellohello"
```

operator() C++◦ lambda

```
auto twice = [] (auto x){ return x+x; };  
  
int i = twice(2); // i == 4  
std::string s = twice("hello"); // s == "hellohello"
```

## lambda

```
auto twice = [] (auto x){ return x+x; };
```

```
int i = twice(2); // i == 4
std::string s = twice("hello"); // s == "hellohello"
```

xyint。

### **lambda**auto&◦ auto&& *rvalue*

```
auto twice = [] (auto x){ return x+x; };

int i = twice(2); // i == 4
std::string s = twice("hello"); // s == "hellohello"
```

## Lambda

```
auto twice = [] (auto x){ return x+x; };

int i = twice(2); // i == 4
std::string s = twice("hello"); // s == "hellohello"
```

```
auto twice = [] (auto x){ return x+x; };

int i = twice(2); // i == 4
std::string s = twice("hello"); // s == "hellohello"
```

auto&&”◦

### **lambda**◦

```
auto twice = [] (auto x){ return x+x; };

int i = twice(2); // i == 4
std::string s = twice("hello"); // s == "hellohello"
```

;

```
auto twice = [] (auto x){ return x+x; };

int i = twice(2); // i == 4
std::string s = twice("hello"); // s == "hellohello"
```

std::ostream& std::ostream&◦ ; auto for(:) auto◦

### **lambda lambda**

```
auto sorter = [] (int lhs, int rhs) -> bool {return lhs < rhs;};
using func_ptr = bool(*)(int, int);
func_ptr sorter_func = sorter; // implicit conversion
```

```
auto sorter = [] (int lhs, int rhs) -> bool {return lhs < rhs;};
```

```
using func_ptr = bool(*)(int, int);
func_ptr sorter_func = sorter; // implicit conversion
```

lambda<sub>operator()</sub>◦ lambda◦ lambda◦

lambdaAPIC ++◦

C ++ 14

lambda◦ ◦

```
auto sorter = [](int lhs, int rhs) -> bool {return lhs < rhs;};
using func_ptr = bool(*)(int, int);
func_ptr sorter_func = sorter; // implicit conversion
```

lambda

lambda

```
class Foo
{
private:
    int i;

public:
    Foo(int val) : i(val) {}

    // definition of a member function
    void Test()
    {
        auto lamb = [] (Foo &foo, int val)
        {
            // modification of a private member variable
            foo.i = val;
        };

        // lamb is allowed to access a private member, because it is a friend of Foo
        lamb(*this, 30);
    }
};
```

lambda◦

Lambdas<sub>this this◦ this</sub>

```
class Foo
{
private:
    int i;

public:
    Foo(int val) : i(val) {}
```

```

// definition of a member function
void Test()
{
    auto lamb = [] (Foo &foo, int val)
    {
        // modification of a private member variable
        foo.i = val;
    };

    // lamb is allowed to access a private member, because it is a friend of Foo
    lamb(*this, 30);
}
};


```

this. this->.

this. this. **lambda** this. **lambda** outlambda.

**lambda**thismutable. const. const.

[=][&] this. . this.

## C++ 17

**Lambda**as**lambda**this. \*this

```

class Foo
{
private:
    int i;

public:
    Foo(int val) : i(val) {}

    // definition of a member function
    void Test()
    {
        auto lamb = [] (Foo &foo, int val)
        {
            // modification of a private member variable
            foo.i = val;
        };

        // lamb is allowed to access a private member, because it is a friend of Foo
        lamb(*this, 30);
    }
};

```

## **lambda**C++ 03

C++**Lambda**. **lambda**C++ 03

```

// Some dummy types:
struct T1 {int dummy;};
struct T2 {int dummy;};

```

```

struct R {int dummy;};

// Code using a lambda function (requires C++11)
R use_lambda(T1 val, T2 ref) {
    // Use auto because the type of the lambda is unknown.
    auto lambda = [val, &ref](int arg1, int arg2) -> R {
        /* lambda-body */
        return R();
    };
    return lambda(12, 27);
}

// The functor class (valid C++03)
// Similar to what the compiler generates for the lambda function.
class Functor {
    // Capture list.
    T1 val;
    T2& ref;

public:
    // Constructor
    inline Functor(T1 val, T2& ref) : val(val), ref(ref) {}

    // Functor body
    R operator()(int arg1, int arg2) const {
        /* lambda-body */
        return R();
    }
};

// Equivalent to use_lambda, but uses a functor (valid C++03).
R use_functor(T1 val, T2 ref) {
    Functor functor(val, ref);
    return functor(12, 27);
}

// Make this a self-contained example.
int main() {
    T1 t1;
    T2 t2;
    use_functor(t1,t2);
    use_lambda(t1,t2);
    return 0;
}

```

## lambda<sub>mutable</sub>functorcall-operatorconst

```

// Some dummy types:
struct T1 {int dummy;};
struct T2 {int dummy;};
struct R {int dummy;};

// Code using a lambda function (requires C++11)
R use_lambda(T1 val, T2 ref) {
    // Use auto because the type of the lambda is unknown.
    auto lambda = [val, &ref](int arg1, int arg2) -> R {
        /* lambda-body */
        return R();
    };
    return lambda(12, 27);
}

```

```

}

// The functor class (valid C++03)
// Similar to what the compiler generates for the lambda function.
class Functor {
    // Capture list.
    T1 val;
    T2& ref;

public:
    // Constructor
    inline Functor(T1 val, T2& ref) : val(val), ref(ref) {}

    // Functor body
    R operator()(int arg1, int arg2) const {
        /* lambda-body */
        return R();
    }
};

// Equivalent to use_lambda, but uses a functor (valid C++03).
R use_functor(T1 val, T2 ref) {
    Functor functor(val, ref);
    return functor(12, 27);
}

// Make this a self-contained example.
int main() {
    T1 t1;
    T2 t2;
    use_functor(t1,t2);
    use_lambda(t1,t2);
    return 0;
}

```

## lambdas

**Euclid<sub>gcd()</sub> lambda**

```

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}

```

lambda<sup>o</sup> lambda<sub>lambda</sub> this is this lambda<sup>o</sup>

std::function

**lambda<sub>std::function</sub>**

```

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}

```

◦ gcd<sub>gcd</sub> lambda<sub>lambda</sub> lambda<sup>o</sup>

```
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}
```

/◦ lambda◦

Y

```
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}
```

gcd

```
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}
```

y\_combinatorlambda◦ lambda◦

lambda◦ “recurse”◦ recurse◦

y\_combinator◦ “recurse”◦ y\_combinator◦ y\_combinatorlambda◦

```
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}
```

lambda◦

**lambdas**

C++ 14

◦

```
template<std::size_t...Is>
void print_indexes( std::index_sequence<Is...> ) {
    using discard=int[];
    (void)discard{0,((void)(
        std::cout << Is << '\n' // here Is is a compile-time constant.
    ),0)...};
}
template<std::size_t I>
void print_indexes_upto() {
    return print_indexes( std::make_index_sequence<I>{} );
}
```

print\_indexes\_upto◦ print\_indexes\_upto◦ ◦ ◦

lambdas◦

## lambda

```
template<std::size_t...Is>
void print_indexes( std::index_sequence<Is...> ) {
    using discard=int[];
    (void)discard{0,((void)(
        std::cout << Is << '\n' // here Is is a compile-time constant.
    ),0)...};
}
template<std::size_t I>
void print_indexes_upto() {
    return print_indexes( std::make_index_sequence<I>{} );
}
```

## C++ 17

### index\_over()

```
template<std::size_t...Is>
void print_indexes( std::index_sequence<Is...> ) {
    using discard=int[];
    (void)discard{0,((void)(
        std::cout << Is << '\n' // here Is is a compile-time constant.
    ),0)...};
}
template<std::size_t I>
void print_indexes_upto() {
    return print_indexes( std::make_index_sequence<I>{} );
}
```

### “inline”

```
template<std::size_t...Is>
void print_indexes( std::index_sequence<Is...> ) {
    using discard=int[];
    (void)discard{0,((void)(
        std::cout << Is << '\n' // here Is is a compile-time constant.
    ),0)...};
}
template<std::size_t I>
void print_indexes_upto() {
    return print_indexes( std::make_index_sequence<I>{} );
}
```

```
auto iindex_overstd::integral_constant<std::size_t, ???>constexpr std::size_t this std::get<i>。
```

```
template<std::size_t...Is>
void print_indexes( std::index_sequence<Is...> ) {
    using discard=int[];
    (void)discard{0,((void)(
        std::cout << Is << '\n' // here Is is a compile-time constant.
    ),0)...};
}
template<std::size_t I>
void print_indexes_upto() {
    return print_indexes( std::make_index_sequence<I>{} );
}
```

◦

◦

Lambda <https://riptutorial.com/zh-TW/cplusplus/topic/572/lambda>

# 22: Pimpl

PIMPL pointerIMPL ementation.cpp

◦ #include◦

pimpl◦

## Examples

### Pimpl

C++11

```
// widget.h

#include <memory> // std::unique_ptr
#include <experimental/propagate_const>

class Widget
{
public:
    Widget();
    ~Widget();
    void DoSomething();

private:
    // the pImpl idiom is named after the typical variable name used
    // ie, pImpl:
    struct Impl;           // forward declaration
    std::experimental::propagate_const<std::unique_ptr<Impl>> pImpl; // ptr to actual
implementation
};
```

```
// widget.h

#include <memory> // std::unique_ptr
#include <experimental/propagate_const>

class Widget
{
public:
    Widget();
    ~Widget();
    void DoSomething();

private:
    // the pImpl idiom is named after the typical variable name used
    // ie, pImpl:
    struct Impl;           // forward declaration
    std::experimental::propagate_const<std::unique_ptr<Impl>> pImpl; // ptr to actual
implementation
};
```

```
pImplWidget/。Widget。  
pImpl“”。Widget“”pImpl。  
unique_ptr<Impl> Widget()。=default=default<br/>Impl。  
unique_ptr<Impl>~Widget()。=default=default<br/>Impl。
```

Pimpl <https://riptutorial.com/zh-TW/cplusplus/topic/2143/pimpl>

# 23: RAII

RAII Resource Acquisition Initialization. SBRMRRIDRAII。C++ -。

◦ ◦

RAII。exit。◦ pid◦

unixexec-familyfork-exec。RAIexec◦ exec◦ fork◦

## Examples

```
std::mutex mtx;

void bad_lock_example() {
    mtx.lock();
    try
    {
        foo();
        bar();
        if (baz())
        {
            mtx.unlock(); // Have to unlock on each exit point.
            return;
        }
        quux();
        mtx.unlock(); // Normal unlock happens here.
    }
    catch(...)
    {
        mtx.unlock(); // Must also force unlock in the presence of
        throw; // exceptions and allow the exception to continue.
    }
}
```

◦ unlock()unlock()◦ ◦

## RAII

```
std::mutex mtx;

void bad_lock_example() {
    mtx.lock();
    try
    {
        foo();
        bar();
        if (baz())
        {
            mtx.unlock(); // Have to unlock on each exit point.
            return;
        }
        quux();
        mtx.unlock(); // Normal unlock happens here.
    }
    catch(...)
    {
        mtx.unlock(); // Must also force unlock in the presence of
```

```

        throw;           // exceptions and allow the exception to continue.
    }
}

```

lock\_guard lock\_guardlock() unlock() . lock\_guardmutex. - ;.

## / ScopeExit

```

template<typename Function>
class Finally final
{
public:
    explicit Finally(Function f) : f(std::move(f)) {}
    ~Finally() { f(); } // (1) See below

    Finally(const Finally&) = delete;
    Finally(Finally&&) = default;
    Finally& operator =(const Finally&) = delete;
    Finally& operator =(Finally&&) = delete;
private:
    Function f;
};

// Execute the function f when the returned object goes out of scope.
template<typename Function>
auto onExit(Function &&f) { return Finally<std::decay_t<Function>>{std::forward<Function>(f)}; }

```

```

template<typename Function>
class Finally final
{
public:
    explicit Finally(Function f) : f(std::move(f)) {}
    ~Finally() { f(); } // (1) See below

    Finally(const Finally&) = delete;
    Finally(Finally&&) = default;
    Finally& operator =(const Finally&) = delete;
    Finally& operator =(Finally&&) = delete;
private:
    Function f;
};

// Execute the function f when the returned object goes out of scope.
template<typename Function>
auto onExit(Function &&f) { return Finally<std::decay_t<Function>>{std::forward<Function>(f)}; }

```

1

- ~Finally() noexcept { f(); } std::terminate
- ~Finally() noexcept(noexcept(f())) { f(); } **terminate**.
- ~Finally() noexcept { try { f(); } catch (...) { /\* ignore exception (might log it) \*/ } std::terminate .

## ScopeSuccessc ++ 17

C ++ 17

```

int std::uncaught_exceptions() noexcept std::uncaught_exception() noexcept

#include <exception>
#include <iostream>

template <typename F>
class ScopeSuccess
{
private:
    F f;
    int uncaughtExceptionCount = std::uncaught_exceptions();
public:
    explicit ScopeSuccess(const F& f) : f(f) {}
    ScopeSuccess(const ScopeSuccess&) = delete;
    ScopeSuccess& operator=(const ScopeSuccess&) = delete;

    // f() might throw, as it can be caught normally.
    ~ScopeSuccess() noexcept(noexcept(f())) {
        if (uncaughtExceptionCount == std::uncaught_exceptions()) {
            f();
        }
    }
};

struct Foo {
    ~Foo() {
        try {
            ScopeSuccess logSuccess{}() {std::cout << "Success 1\n";};
            // Scope succeeds,
            // even if Foo is destroyed during stack unwinding
            // (so when 0 < std::uncaught_exceptions())
            // (or previously std::uncaught_exception() == true)
        } catch (...) {
        }
        try {
            ScopeSuccess logSuccess{}() {std::cout << "Success 2\n";};

            throw std::runtime_error("Failed"); // returned value
                                              // of std::uncaught_exceptions increases
        } catch (...) { // returned value of std::uncaught_exceptions decreases
        }
    }
};

int main()
{
    try {
        Foo foo;

        throw std::runtime_error("Failed"); // std::uncaught_exceptions() == 1
    } catch (...) { // std::uncaught_exceptions() == 0
    }
}

```

```

#include <exception>
#include <iostream>

template <typename F>
class ScopeSuccess

```

```

{
private:
    F f;
    int uncaughtExceptionCount = std::uncaught_exceptions();
public:
    explicit ScopeSuccess(const F& f) : f(f) {}
    ScopeSuccess(const ScopeSuccess&) = delete;
    ScopeSuccess& operator =(const ScopeSuccess&) = delete;

    // f() might throw, as it can be caught normally.
    ~ScopeSuccess() noexcept(noexcept(f())) {
        if (uncaughtExceptionCount == std::uncaught_exceptions()) {
            f();
        }
    }
};

struct Foo {
    ~Foo() {
        try {
            ScopeSuccess logSuccess{}(); std::cout << "Success 1\n";
            // Scope succeeds,
            // even if Foo is destroyed during stack unwinding
            // (so when 0 < std::uncaught_exceptions())
            // (or previously std::uncaught_exception() == true)
        } catch (...) {
        }
        try {
            ScopeSuccess logSuccess{}(); std::cout << "Success 2\n";
            throw std::runtime_error("Failed"); // returned value
                                            // of std::uncaught_exceptions increases
        } catch (...) { // returned value of std::uncaught_exceptions decreases
        }
    }
};

int main()
{
    try {
        Foo foo;

        throw std::runtime_error("Failed"); // std::uncaught_exceptions() == 1
    } catch (...) { // std::uncaught_exceptions() == 0
    }
}

```

## ScopeFailc ++ 17

### C ++ 17

```
int std::uncaught_exceptions() noexcept std::uncaught_exception() noexcept
```

```
#include <exception>
#include <iostream>

template <typename F>
class ScopeFail
```

```

{
private:
    F f;
    int uncaughtExceptionCount = std::uncaught_exceptions();
public:
    explicit ScopeFail(const F& f) : f(f) {}
    ScopeFail(const ScopeFail&) = delete;
    ScopeFail& operator =(const ScopeFail&) = delete;

    // f() should not throw, else std::terminate is called.
    ~ScopeFail() {
        if (uncaughtExceptionCount != std::uncaught_exceptions()) {
            f();
        }
    }
};

struct Foo {
    ~Foo() {
        try {
            ScopeFail logFailure{}(); std::cout << "Fail 1\n";
            // Scope succeeds,
            // even if Foo is destroyed during stack unwinding
            // (so when 0 < std::uncaught_exceptions())
            // (or previously std::uncaught_exception() == true)
        } catch (...) {
        }
        try {
            ScopeFail logFailure{}(); std::cout << "Failure 2\n";
            throw std::runtime_error("Failed"); // returned value
                                              // of std::uncaught_exceptions increases
        } catch (...) { // returned value of std::uncaught_exceptions decreases
        }
    }
};

int main()
{
    try {
        Foo foo;

        throw std::runtime_error("Failed"); // std::uncaught_exceptions() == 1
    } catch (...) { // std::uncaught_exceptions() == 0
    }
}

```

```

#include <exception>
#include <iostream>

template <typename F>
class ScopeFail
{
private:
    F f;
    int uncaughtExceptionCount = std::uncaught_exceptions();
public:
    explicit ScopeFail(const F& f) : f(f) {}
    ScopeFail(const ScopeFail&) = delete;

```

```

ScopeFail& operator =(const ScopeFail&) = delete;

// f() should not throw, else std::terminate is called.
~ScopeFail() {
    if (uncaughtExceptionCount != std::uncaught_exceptions()) {
        f();
    }
}

struct Foo {
    ~Foo() {
        try {
            ScopeFail logFailure{}() {std::cout << "Fail 1\n";};
            // Scope succeeds,
            // even if Foo is destroyed during stack unwinding
            // (so when 0 < std::uncaught_exceptions())
            // (or previously std::uncaught_exception() == true)
        } catch (...) {
        }
        try {
            ScopeFail logFailure{}() {std::cout << "Failure 2\n";};

            throw std::runtime_error("Failed"); // returned value
                                            // of std::uncaught_exceptions increases
        } catch (...) { // returned value of std::uncaught_exceptions decreases
        }
    }
};

int main()
{
    try {
        Foo foo;

        throw std::runtime_error("Failed"); // std::uncaught_exceptions() == 1
    } catch (...) { // std::uncaught_exceptions() == 0
    }
}

```

RAII <https://riptutorial.com/zh-TW/cplusplus/topic/1320/raii->

# 24: RTTI

## Examples

```
.name() std::type_info::name();

#include <iostream>
#include <typeinfo>

int main()
{
    int speed = 110;

    std::cout << typeid(speed).name() << '\n';
}
```

```
#include <iostream>
#include <typeinfo>

int main()
{
    int speed = 110;

    std::cout << typeid(speed).name() << '\n';
}
```

## dynamic\_cast

```
dynamic_cast<>() ;

BCclass A

class A { public: virtual ~A(){} };

class B: public A
{ public: void work4B(){} };

class C: public A
{ public: void work4C(){} };

void non_polymorphic_work(A* ap)
{
    if (B* bp = dynamic_cast<B*>(ap))
        bp->work4B();
    if (C* cp = dynamic_cast<C*>(ap))
        cp->work4C();
}
```

## typeid

```
typeid. const std::type_info. CV.
```

```
struct Base {  
    virtual ~Base() = default;  
};  
struct Derived : Base {};  
Base* b = new Derived;  
assert(typeid(*b) == typeid(Derived{})); // OK
```

typeid◦ **cv-qualification**◦ typeid(Derived) typeid(Derived{})

```
struct Base {  
    virtual ~Base() = default;  
};  
struct Derived : Base {};  
Base* b = new Derived;  
assert(typeid(*b) == typeid(Derived{})); // OK
```

typeid◦ **info**◦

```
struct Base {  
    virtual ~Base() = default;  
};  
struct Derived : Base {};  
Base* b = new Derived;  
assert(typeid(*b) == typeid(Derived{})); // OK
```

**C ++**

**dynamic\_cast**/◦

**static\_cast**◦

**reinterpret\_cast**◦ ◦

**const\_cast**◦ **const / volatile**◦ **constAPI**◦

**RTTI** <https://riptutorial.com/zh-TW/cplusplus/topic/3129/rtti->

# 25: SFINAE

## Examples

### enable\_if

std::enable\_if SFINAE。

```
template <bool Cond, typename Result=void>
struct enable_if { };

template <typename Result>
struct enable_if<true, Result> {
    using type = Result;
};
```

enable\_if<true, R>::typeRenable\_if<false, T>::typeenable\_if<false, T>::typeenable\_iftype。

std::enable\_if

```
template <bool Cond, typename Result=void>
struct enable_if { };

template <typename Result>
struct enable_if<true, Result> {
    using type = Result;
};
```

negate(1)。

```
template <bool Cond, typename Result=void>
struct enable_if { };

template <typename Result>
struct enable_if<true, Result> {
    using type = Result;
};
```

negate<int>!std::is\_arithmetic<int>::valuefalse。 SFINAE。 negate(1) - 。

std::enable\_if SFINAE SFINAE。 std::size size(arg)

```
template <bool Cond, typename Result=void>
struct enable_if { };

template <typename Result>
struct enable_if<true, Result> {
    using type = Result;
};
```

is\_sizeable SFINAE。

- incr(i, 3) i += 3 if INT\_MAX int.

```
template <bool Cond, typename Result=void>
struct enable_if { };

template <typename Result>
struct enable_if<true, Result> {
    using type = Result;
};
```

std::enable\_if API。 is\_sizeable<Cont>::value cont.size() size is\_sizeable。 std::is\_signed incr1。

## void\_t

### C++11

void\_t void\_t void\_t

### std::void\_t C++17

```
template <class...> using void_t = void;
```

```
template <class...> using void_t = void;
```

void\_t foo()

```
template <class...> using void_t = void;
```

has\_foo<T>::value has\_foo<T, void> has\_foo<T, void>。

- T foo() void。 has\_foo<T>::value true
- T。 has\_foo<T>::value false。

```
template <class...> using void_t = void;
```

std::declval decltype。

## void。

```
template <class...> using void_t = void;
```

std::void\_t can\_apply can\_apply

```
template <class...> using void_t = void;
```

```
template <class...> using void_t = void;
```

◦

std::C++17 can\_apply ◦

void\_t Walter Brown◦ CppCon 2016◦

## decltype

C++11

decltype

```
namespace details {
    using std::to_string;

    // this one is constrained on being able to call to_string(T)
    template <class T>
    auto convert_to_string(T const& val, int )
        -> decltype(to_string(val))
    {
        return to_string(val);
    }

    // this one is unconstrained, but less preferred due to the ellipsis argument
    template <class T>
    std::string convert_to_string(T const& val, ... )
    {
        std::ostringstream oss;
        oss << val;
        return oss.str();
    }
}

template <class T>
std::string convert_to_string(T const& val)
{
    return details::convert_to_string(val, 0);
}
```

to\_string() convert\_to\_string() details::convert\_to\_string()◦ 0int0.....

to\_string() convert\_to\_string() decltype(to\_string(val))◦ ◦ operator<<(std::ostream&, T)◦ oss << val◦

## SFINAE

SFINAE S ubstitution F ailure I s N ot A n E rror◦

-◦

```
template <class T>
auto begin(T& c) -> decltype(c.begin()) { return c.begin(); }

template <class T, size_t N>
```

```
T* begin(T (&arr) [N]) { return arr; }

int vals[10];
begin(vals); // OK. The first function template substitution fails because
             // vals.begin() is ill-formed. This is not an error! That function
             // is just removed from consideration as a viable overload candidate,
             // leaving us with the array overload.
```

◦

```
template <class T>
auto begin(T& c) -> decltype(c.begin()) { return c.begin(); }

template <class T, size_t N>
T* begin(T (&arr) [N]) { return arr; }

int vals[10];
begin(vals); // OK. The first function template substitution fails because
             // vals.begin() is ill-formed. This is not an error! That function
             // is just removed from consideration as a viable overload candidate,
             // leaving us with the array overload.
```

## enable\_if\_all / enable\_if\_any

### C++ 11

---

```
template<typename ...Args> void func(Args &&...args) { //... };
```

### C++ 17 enable\_if SFINAE Args Args ◦ C++ 17 std::conjunction std::disjunction ◦

```
template<typename ...Args> void func(Args &&...args) { //... };
```

### C++ 17 ◦ ◦

```
std::conjunction std::disjunction ◦ std::enable_if enable_if_all enable_if_any ◦ ◦
```

---

enable\_if\_all enable\_if\_any

---

seq\_and seq\_or std::conjunction std::disjunction

```
template<typename ...Args> void func(Args &&...args) { //... };
```

```
template<typename ...Args> void func(Args &&...args) { //... };
```

```
template<typename ...Args> void func(Args &&...args) { //... };
```

---

---

```
template<typename ...Args> void func(Args &&...args) { //... };
```

## is\_detected

type\_trait~~SFINAE~~detected\_or detected\_t is\_detected。

typename Default template <typename...> Optypename ... Args

- is\_detected std::true\_typestd::false\_typeOp<Args...>
- detected\_t Op<Args...>nonesuchOp<Args...>
- detected\_or value\_t is\_detected typeOp<Args...>DefaultOp<Args...>

std::void\_t SFINAE

## C++ 17

```
namespace detail {
    template <class Default, class AlwaysVoid,
              template<class...> class Op, class... Args>
    struct detector
    {
        using value_t = std::false_type;
        using type = Default;
    };

    template <class Default, template<class...> class Op, class... Args>
    struct detector<Default, std::void_t<Op<Args...>>, Op, Args...>
    {
        using value_t = std::true_type;
        using type = Op<Args...>;
    };
}

} // namespace detail

// special type to indicate detection failure
struct nosuch {
    nosuch() = delete;
    ~nosuch() = delete;
    nosuch(nosuch const&) = delete;
    void operator=(nosuch const&) = delete;
};

template <template<class...> class Op, class... Args>
using is_detected =
    typename detail::detector<nosuch, void, Op, Args...>::value_t;

template <template<class...> class Op, class... Args>
using detected_t = typename detail::detector<nosuch, void, Op, Args...>::type;

template <class Default, template<class...> class Op, class... Args>
using detected_or = detail::detector<Default, void, Op, Args...>;
```

```
namespace detail {
    template <class Default, class AlwaysVoid,
              template<class...> class Op, class... Args>
    struct detector
```

```

{
    using value_t = std::false_type;
    using type = Default;
};

template <class Default, template<class...> class Op, class... Args>
struct detector<Default, std::void_t<Op<Args...>>, Op, Args...>
{
    using value_t = std::true_type;
    using type = Op<Args...>;
};

} // namespace detail

// special type to indicate detection failure
struct nosuch {
    nosuch() = delete;
    ~nosuch() = delete;
    nosuch(nosuch const&) = delete;
    void operator=(nosuch const&) = delete;
};

template <template<class...> class Op, class... Args>
using is_detected =
    typename detail::detector<nosuch, void, Op, Args...>::value_t;

template <template<class...> class Op, class... Args>
using detected_t = typename detail::detector<nosuch, void, Op, Args...>::type;

template <class Default, template<class...> class Op, class... Args>
using detected_or = detail::detector<Default, void, Op, Args...>;

```

enable\_if<>.

◦

decltype◦

" - random\_access\_tag◦ ◦

```

#include <algorithm>
#include <iterator>

namespace detail
{
    // this gives us infinite types, that inherit from each other
    template<std::size_t N>
    struct pick : pick<N-1> {};
    template<>
    struct pick<0> {};

    // the overload we want to be preferred have a higher N in pick<N>
    // this is the first helper template function
    template<typename T>
    auto stable_sort(T& t, pick<2>)
        -> decltype( t.stable_sort(), void() )
    {
        // if the container have a member stable_sort, use that
        t.stable_sort();
    }
}

```

```

}

// this helper will be second best match
template<typename T>
auto stable_sort(T& t, pick<1>)
    -> decltype( t.sort(), void() )
{
    // if the container have a member sort, but no member stable_sort
    // it's customary that the sort member is stable
    t.sort();
}

// this helper will be picked last
template<typename T>
auto stable_sort(T& t, pick<0>)
    -> decltype( std::stable_sort(std::begin(t), std::end(t)), void() )
{
    // the container have neither a member sort, nor member stable_sort
    std::stable_sort(std::begin(t), std::end(t));
}

// this is the function the user calls. it will dispatch the call
// to the correct implementation with the help of 'tags'.
template<typename T>
void stable_sort(T& t)
{
    // use an N that is higher than any used above.
    // this will pick the highest overload that is well formed.
    detail::stable_sort(t, detail::pick<10>());
}

```

◦

## tag-dispatch◦

SFINAE <https://riptutorial.com/zh-TW/cplusplus/topic/1169/sfinae-->

# 26: static\_assert

- `static_assert bool_constexpr message`
- `static_assert bool_constexpr / *C++17*/`



◦

## Examples

### static\_assert

- `static_assert()` ◦

```
template<typename T>
T mul10(const T t)
{
    static_assert( std::is_integral<T>::value, "mul10() only works for integral types" );
    return (t << 3) + (t << 1);
}
```

`static_assert() bool constexpr` ◦ C++17; ◦

### C++17

```
template<typename T>
T mul10(const T t)
{
    static_assert( std::is_integral<T>::value, "mul10() only works for integral types" );
    return (t << 3) + (t << 1);
}
```

- `constexpr`
- - 
  - `constexpr`
  -
- C++
- `sizeof(T)` `sizeof(T)` `32int`
- 

`static_assert() SFINAE /std::enable_if<>` ◦ /◦ SFINAE◦

`static_assert` <https://riptutorial.com/zh-TW/cplusplus/topic/3822/static-assert>

# 27: std :: function

## Examples

```
#include <iostream>
#include <functional>
std::function<void(int , const std::string&)> myFuncObj;
void theFunc(int i, const std::string& s)
{
    std::cout << s << ":" << i << std::endl;
}
int main(int argc, char *argv[])
{
    myFuncObj = theFunc;
    myFuncObj(10, "hello world");
}
```

## std :: function std :: bind

- std::bind std::function

```
class A
{
public:
    std::function<void(int, const std::string&)> m_CbFunc = nullptr;
    void foo()
    {
        if (m_CbFunc)
        {
            m_CbFunc(100, "event fired");
        }
    }
};

class B
{
public:
    B()
    {
        auto aFunc = std::bind(&B::eventHandler, this, std::placeholders::_1,
std::placeholders::_2);
        anObjA.m_CbFunc = aFunc;
    }
    void eventHandler(int i, const std::string& s)
    {
        std::cout << s << ":" << i << std::endl;
    }

    void DoSomethingOnA()
    {
        anObjA.foo();
    }
}

A anObjA;
```

```

};

int main(int argc, char *argv[])
{
    B anObjB;
    anObjB.DoSomethingOnA();
}

```

## std :: function with lambdastd :: bind

```

#include <iostream>
#include <functional>

using std::placeholders::_1; // to be used in std::bind example

int stdf_foobar (int x, std::function<int(int)> moo)
{
    return x + moo(x); // std::function moo called
}

int foo (int x) { return 2+x; }

int foo_2 (int x, int y) { return 9*x + y; }

int main()
{
    int a = 2;

    /* Function pointers */
    std::cout << stdf_foobar(a, &foo) << std::endl; // 6 ( 2 + (2+2) )
    // can also be: stdf_foobar(2, foo)

    /* Lambda expressions */
    /* An unnamed closure from a lambda expression can be
     * stored in a std::function object:
     */
    int capture_value = 3;
    std::cout << stdf_foobar(a,
                           [capture_value](int param) -> int { return 7 + capture_value *
param; })
                           << std::endl;
    // result: 15 == value + (7 * capture_value * value) == 2 + (7 + 3 * 2)

    /* std::bind expressions */
    /* The result of a std::bind expression can be passed.
     * For example by binding parameters to a function pointer call:
     */
    int b = stdf_foobar(a, std::bind(foo_2, _1, 3));
    std::cout << b << std::endl;
    // b == 23 == 2 + ( 9*2 + 3 )
    int c = stdf_foobar(a, std::bind(foo_2, 5, _1));
    std::cout << c << std::endl;
    // c == 49 == 2 + ( 9*5 + 2 )

    return 0;
}

```

**function**

```
std::function< std::function[value semantics] [1]callable> callable;
```

```
function<"function> noexcept C++
```

```
//Header file
using MyPredicate = std::function<bool(const MyValue &, const MyValue &) >;

void SortMyContainer(MyContainer &C, const MyPredicate &pred);

//Source file
void SortMyContainer(MyContainer &C, const MyPredicate &pred)
{
    std::sort(C.begin(), C.end(), pred);
}
```

```
SortMyContainer SortMyContainer pred
```

```
function</> functioncallable< SortMyContainer> function<>
```

```
◦ ◦
```

```
function< allocator [..... ] [2]>
```

## C++ 17

```
allocator function function ◦
```

```
function< function function function > function
```

## std :: function

```
/*
 * This example show some ways of using std::function to call
 * a) C-like function
 * b) class-member function
 * c) operator()
 * d) lambda function
 *
 * Function call can be made:
 * a) with right arguments
 * b) argumens with different order, types and count
 */
#include <iostream>
#include <functional>
#include <iostream>
#include <vector>

using std::cout;
using std::endl;
using namespace std::placeholders;

// simple function to be called
double foo_fn(int x, float y, double z)
```

```

{
    double res = x + y + z;
    std::cout << "foo_fn called with arguments: "
        << x << ", " << y << ", " << z
        << " result is : " << res
        << std::endl;
    return res;
}

// structure with member function to call
struct foo_struct
{
    // member function to call
    double foo_fn(int x, float y, double z)
    {
        double res = x + y + z;
        std::cout << "foo_struct::foo_fn called with arguments: "
            << x << ", " << y << ", " << z
            << " result is : " << res
            << std::endl;
        return res;
    }
    // this member function has different signature - but it can be used too
    // please note that argument order is changed too
    double foo_fn_4(int x, double z, float y, long xx)
    {
        double res = x + y + z + xx;
        std::cout << "foo_struct::foo_fn_4 called with arguments: "
            << x << ", " << z << ", " << y << ", " << xx
            << " result is : " << res
            << std::endl;
        return res;
    }
    // overloaded operator() makes whole object to be callable
    double operator()(int x, float y, double z)
    {
        double res = x + y + z;
        std::cout << "foo_struct::operator() called with arguments: "
            << x << ", " << y << ", " << z
            << " result is : " << res
            << std::endl;
        return res;
    }
};

int main(void)
{
    // typedefs
    using function_type = std::function<double(int, float, double)>;
    // foo_struct instance
    foo_struct fs;

    // here we will store all binded functions
    std::vector<function_type> bindings;

    // var #1 - you can use simple function
    function_type var1 = foo_fn;
    bindings.push_back(var1);
}

```

```

// var #2 - you can use member function
function_type var2 = std::bind(&foo_struct::foo_fn, fs, _1, _2, _3);
bindings.push_back(var2);

// var #3 - you can use member function with different signature
// foo_fn_4 has different count of arguments and types
function_type var3 = std::bind(&foo_struct::foo_fn_4, fs, _1, _3, _2, _01);
bindings.push_back(var3);

// var #4 - you can use object with overloaded operator()
function_type var4 = fs;
bindings.push_back(var4);

// var #5 - you can use lambda function
function_type var5 = [](int x, float y, double z)
{
    double res = x + y + z;
    std::cout << "lambda called with arguments: "
        << x << ", " << y << ", " << z
        << " result is : " << res
        << std::endl;
    return res;
};
bindings.push_back(var5);

std::cout << "Test stored functions with arguments: x = 1, y = 2, z = 3"
    << std::endl;

for (auto f : bindings)
    f(1, 2, 3);

}

```

```

/*
 * This example show some ways of using std::function to call
 * a) C-like function
 * b) class-member function
 * c) operator()
 * d) lambda function
 *
 * Function call can be made:
 * a) with right arguments
 * b) argumens with different order, types and count
 */
#include <iostream>
#include <functional>
#include <iostream>
#include <vector>

using std::cout;
using std::endl;
using namespace std::placeholders;

// simple function to be called
double foo_fn(int x, float y, double z)
{
    double res = x + y + z;
    std::cout << "foo_fn called with arguments: "

```

```

        << x << ", " << y << ", " << z
        << " result is : " << res
        << std::endl;
    return res;
}

// structure with member function to call
struct foo_struct
{
    // member function to call
    double foo_fn(int x, float y, double z)
    {
        double res = x + y + z;
        std::cout << "foo_struct::foo_fn called with arguments: "
            << x << ", " << y << ", " << z
            << " result is : " << res
            << std::endl;
        return res;
    }
    // this member function has different signature - but it can be used too
    // please note that argument order is changed too
    double foo_fn_4(int x, double z, float y, long xx)
    {
        double res = x + y + z + xx;
        std::cout << "foo_struct::foo_fn_4 called with arguments: "
            << x << z << ", " << y << ", " << xx
            << " result is : " << res
            << std::endl;
        return res;
    }
    // overloaded operator() makes whole object to be callable
    double operator()(int x, float y, double z)
    {
        double res = x + y + z;
        std::cout << "foo_struct::operator() called with arguments: "
            << x << ", " << y << ", " << z
            << " result is : " << res
            << std::endl;
        return res;
    }
};

int main(void)
{
    // typedefs
    using function_type = std::function<double(int, float, double)>

    // foo_struct instance
    foo_struct fs;

    // here we will store all binded functions
    std::vector<function_type> bindings;

    // var #1 - you can use simple function
    function_type var1 = foo_fn;
    bindings.push_back(var1);

    // var #2 - you can use member function
    function_type var2 = std::bind(&foo_struct::foo_fn, fs, _1, _2, _3);
    bindings.push_back(var2);
}

```

```

// var #3 - you can use member function with different signature
// foo_fn_4 has different count of arguments and types
function_type var3 = std::bind(&foo_struct::foo_fn_4, fs, _1, _3, _2, 01);
bindings.push_back(var3);

// var #4 - you can use object with overloaded operator()
function_type var4 = fs;
bindings.push_back(var4);

// var #5 - you can use lambda function
function_type var5 = [](int x, float y, double z)
{
    double res = x + y + z;
    std::cout << "lambda called with arguments: "
        << x << ", " << y << ", " << z
        << " result is : " << res
        << std::endl;
    return res;
};
bindings.push_back(var5);

std::cout << "Test stored functions with arguments: x = 1, y = 2, z = 3"
    << std::endl;

for (auto f : bindings)
    f(1, 2, 3);

}

```

## std :: tuple

◦

### std :: tuple

```

#include <iostream>
#include <functional>
#include <tuple>
#include <iostream>

// simple function to be called
double foo_fn(int x, float y, double z)
{
    double res = x + y + z;
    std::cout << "foo_fn called. x = " << x << " y = " << y << " z = " << z
        << " res=" << res;
    return res;
}

// helpers for tuple unrolling
template<int ...> struct seq {};
template<int N, int ...S> struct gens : gens<N-1, N-1, S...> {};
template<int ...S> struct gens<0, S...>{ typedef seq<S...> type; };

// invocation helper
template<typename FN, typename P, int ...S>
double call_fn_internal(const FN& fn, const P& params, const seq<S...>)
{

```

```

    return fn(std::get<S>(params) ...);
}
// call function with arguments stored in std::tuple
template<typename Ret, typename ...Args>
Ret call_fn(const std::function<Ret(Args...)>& fn,
            const std::tuple<Args...>& params)
{
    return call_fn_internal(fn, params, typename gens<sizeof...(Args)>::type());
}

int main(void)
{
    // arguments
    std::tuple<int, float, double> t = std::make_tuple(1, 5, 10);
    // function to call
    std::function<double(int, float, double)> fn = foo_fn;

    // invoke a function with stored arguments
    call_fn(fn, t);
}

```

```

#include <iostream>
#include <functional>
#include <tuple>
#include <iostream>

// simple function to be called
double foo_fn(int x, float y, double z)
{
    double res = x + y + z;
    std::cout << "foo_fn called. x = " << x << " y = " << y << " z = " << z
           << " res=" << res;
    return res;
}

// helpers for tuple unrolling
template<int ...> struct seq {};
template<int N, int ...S> struct gens : gens<N-1, N-1, S...> {};
template<int ...S> struct gens<0, S...>{ typedef seq<S...> type; };

// invocation helper
template<typename FN, typename P, int ...S>
double call_fn_internal(const FN& fn, const P& params, const seq<S...>)
{
    return fn(std::get<S>(params) ...);
}
// call function with arguments stored in std::tuple
template<typename Ret, typename ...Args>
Ret call_fn(const std::function<Ret(Args...)>& fn,
            const std::tuple<Args...>& params)
{
    return call_fn_internal(fn, params, typename gens<sizeof...(Args)>::type());
}

int main(void)
{
    // arguments
    std::tuple<int, float, double> t = std::make_tuple(1, 5, 10);

```

```
// function to call
std::function<double(int, float, double)> fn = foo_fn;

// invoke a function with stored arguments
call_fn(fn, t);
}
```

std :: function <https://riptutorial.com/zh-TW/cplusplus/topic/2294/std---function->

# 28: std :: setstd :: multiset

set. multisetmultiset.

C ++. C ++ 98;.

## Examples

◦

• . .  
• . . ; .  
• . .

```
#include <iostream>
#include <set>

int main ()
{
    std::set<int> sut;
    std::set<int>::iterator it;
    std::pair<std::set<int>::iterator,bool> ret;

    // Basic insert
    sut.insert(7);
    sut.insert(5);
    sut.insert(12);

    ret = sut.insert(23);
    if (ret.second==true)
        std::cout << "# 23 has been inserted!" << std::endl;

    ret = sut.insert(23); // since it's a set and 23 is already present in it, this insert
    should fail
    if (ret.second==false)
        std::cout << "# 23 already present in set!" << std::endl;

    // Insert with hint for optimization
    it = sut.end();
    // This case is optimized for C++11 and above
    // For earlier version, point to the element preceding your insertion
    sut.insert(it, 30);

    // inserting a range of values
    std::set<int> sut2;
    sut2.insert(20);
    sut2.insert(30);
    sut2.insert(45);
    std::set<int>::iterator itStart = sut2.begin();
    std::set<int>::iterator itEnd = sut2.end();

    sut.insert (itStart, itEnd); // second iterator is excluded from insertion

    std::cout << std::endl << "Set under test contains:" << std::endl;
```

```

for (it = sut.begin(); it != sut.end(); ++it)
{
    std::cout << *it << std::endl;
}

return 0;
}

#include <iostream>
#include <set>

int main ()
{
    std::set<int> sut;
    std::set<int>::iterator it;
    std::pair<std::set<int>::iterator,bool> ret;

    // Basic insert
    sut.insert(7);
    sut.insert(5);
    sut.insert(12);

    ret = sut.insert(23);
    if (ret.second==true)
        std::cout << "# 23 has been inserted!" << std::endl;

    ret = sut.insert(23); // since it's a set and 23 is already present in it, this insert
should fail
    if (ret.second==false)
        std::cout << "# 23 already present in set!" << std::endl;

    // Insert with hint for optimization
    it = sut.end();
    // This case is optimized for C++11 and above
    // For earlier version, point to the element preceding your insertion
    sut.insert(it, 30);

    // inserting a range of values
    std::set<int> sut2;
    sut2.insert(20);
    sut2.insert(30);
    sut2.insert(45);
    std::set<int>::iterator itStart = sut2.begin();
    std::set<int>::iterator itEnd = sut2.end();

    sut.insert (itStart, itEnd); // second iterator is excluded from insertion

    std::cout << std::endl << "Set under test contains:" << std::endl;
    for (it = sut.begin(); it != sut.end(); ++it)
    {
        std::cout << *it << std::endl;
    }

    return 0;
}

```

## ◦ initializer\_list

```
auto il = { 7, 5, 12 };
```

```

std::multiset<int> msut;
msut.insert(il);

setmultiset.

◦ ◦ intcompare

#include <iostream>
#include <set>
#include <stdlib.h>

struct custom_compare final
{
    bool operator() (const std::string& left, const std::string& right) const
    {
        int nLeft = atoi(left.c_str());
        int nRight = atoi(right.c_str());
        return nLeft < nRight;
    }
};

int main ()
{
    std::set<std::string> sut({"1", "2", "5", "23", "6", "290"});

    std::cout << "### Default sort on std::set<std::string> :" << std::endl;
    for (auto &&data: sut)
        std::cout << data << std::endl;

    std::set<std::string, custom_compare> sut_custom({"1", "2", "5", "23", "6", "290"},
                                                       custom_compare{}); //< Compare object
optional as its default constructible.

    std::cout << std::endl << "### Custom sort on set :" << std::endl;
    for (auto &&data : sut_custom)
        std::cout << data << std::endl;

    auto compare_via_lambda = [] (auto &&lhs, auto &&rhs){ return lhs > rhs; };
    using set_via_lambda = std::set<std::string, decltype(compare_via_lambda)>;
    set_via_lambda sut_reverse_via_lambda {"1", "2", "5", "23", "6", "290"},

    compare_via_lambda;

    std::cout << std::endl << "### Lambda sort on set :" << std::endl;
    for (auto &&data : sut_reverse_via_lambda)
        std::cout << data << std::endl;

    return 0;
}

```

```

#include <iostream>
#include <set>
#include <stdlib.h>

struct custom_compare final
{
    bool operator() (const std::string& left, const std::string& right) const
    {
        int nLeft = atoi(left.c_str());

```

```

        int nRight = atoi(right.c_str());
        return nLeft < nRight;
    }
};

int main ()
{
    std::set<std::string> sut({"1", "2", "5", "23", "6", "290"});

    std::cout << "### Default sort on std::set<std::string> :" << std::endl;
    for (auto &&data: sut)
        std::cout << data << std::endl;

    std::set<std::string, custom_compare> sut_custom({"1", "2", "5", "23", "6", "290"}, 
                                                    custom_compare{}); //< Compare object
optional as its default constructible.

    std::cout << std::endl << "### Custom sort on set :" << std::endl;
    for (auto &&data : sut_custom)
        std::cout << data << std::endl;

    auto compare_via_lambda = [] (auto &&lhs, auto &&rhs){ return lhs > rhs; };
    using set_via_lambda = std::set<std::string, decltype(compare_via_lambda)>;
    set_via_lambda sut_reverse_via_lambda {"1", "2", "5", "23", "6", "290"}, 
                                         compare_via_lambda);

    std::cout << std::endl << "### Lambda sort on set :" << std::endl;
    for (auto &&data : sut_reverse_via_lambda)
        std::cout << data << std::endl;

    return 0;
}

```

3 std::set

- std::less<T>◦ operator<◦ ◦

**API23◦ operator<◦**

- 

- ◦ std::pairfirst◦

- ◦ **constconst.....◦**

- ◦ **compare◦**

## Lambda

**Lambda◦◦**

**lambda** lambda`decltype(lambda)◦◦`

std::set◦

```
std::set compare.
```

- 
- 2.

## setmultiset

```
std::set std::multiset
```

```
find()。 end()。
```

```
std::set<int> sut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3); // contains 3, 10, 15, 22

auto itS = sut.find(10); // the value is found, so *itS == 10
itS = sut.find(555); // the value is not found, so itS == sut.end()

std::multiset<int> msut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(15);
sut.insert(3); // contains 3, 10, 15, 15, 22

auto itMS = msut.find(10);
```

```
count() set / multiset 01.
```

```
std::set<int> sut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3); // contains 3, 10, 15, 22

auto itS = sut.find(10); // the value is found, so *itS == 10
itS = sut.find(555); // the value is not found, so itS == sut.end()

std::multiset<int> msut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(15);
sut.insert(3); // contains 3, 10, 15, 15, 22

auto itMS = msut.find(10);
```

```
std::multiset。 equal_range()。 std::pair。 multiset。
```

```
std::set<int> sut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3); // contains 3, 10, 15, 22
```

```
auto itS = sut.find(10); // the value is found, so *itS == 10
itS = sut.find(555); // the value is not found, so itS == sut.end()

std::multiset<int> msut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(15);
sut.insert(3); // contains 3, 10, 15, 15, 22

auto itMS = msut.find(10);
```

## set / multiset clear

```
std::set<int> sut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3);
sut.clear(); //size of sut is 0
```

## erase ◦

```
std::set<int> sut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3);
sut.clear(); //size of sut is 0
```

```
std::set<int> sut;
sut.insert(10);
sut.insert(15);
sut.insert(22);
sut.insert(3);
sut.clear(); //size of sut is 0
```

multiset ◦ multiset ◦

std :: set std :: multiset <https://riptutorial.com/zh-TW/cplusplus/topic/9005/std---setstd----multiset>

# 29: Typedef

using `typedef` C ++ 11.

- `typedef type-specifier-seq init-declarator-list ;`
- `attribute-specifier-seq typedef decl-specifier-seq init-declarator-list ; //C ++ 11`
- `attribute-specifier-seq opt = type-id ; //C ++ 11`

## Examples

### `typedef`

`typedef` `typedef` `o` `typedef` `o`

```
int T;           // T has type int
typedef int T; // T is an alias for int

int A[100];      // A has type "array of 100 ints"
typedef int A[100]; // A is an alias for the type "array of 100 ints"
```

◦

```
int T;           // T has type int
typedef int T; // T is an alias for int

int A[100];      // A has type "array of 100 ints"
typedef int A[100]; // A is an alias for the type "array of 100 ints"
```

`typedef` `o` `o`

```
int T;           // T has type int
typedef int T; // T is an alias for int

int A[100];      // A has type "array of 100 ints"
typedef int A[100]; // A is an alias for the type "array of 100 ints"
```

### `typedef`

`typedef` `o`

```
void (*f)(int);      // f has type "pointer to function of int returning void"
typedef void (*f)(int); // f is an alias for "pointer to function of int returning void"
```

◦

```
void (*f)(int);      // f has type "pointer to function of int returning void"
typedef void (*f)(int); // f is an alias for "pointer to function of int returning void"
```

```
void (*f)(int);           // f has type "pointer to function of int returning void"  
typedef void (*f)(int); // f is an alias for "pointer to function of int returning void"
```

typedef

```
void (*f)(int);           // f has type "pointer to function of int returning void"  
typedef void (*f)(int); // f is an alias for "pointer to function of int returning void"
```

## typedef

typedef。 typedef。

```
int *x, (*p)();           // x has type int*, and p has type int(*)()  
typedef int *x, (*p)(); // x is an alias for int*, while p is an alias for int(*)()
```

“””

## C ++ 11

using。。

```
using I = int;  
using A = int[100];          // array of 100 ints  
using FP = void(*)(int);      // pointer to function of int returning void  
using MP = void (Foo::*)(int); // pointer to member function of Foo of int returning void
```

usingusing typedef。。

typedef using。 using“template typedef”。

Typedef <https://riptutorial.com/zh-TW/cplusplus/topic/9328/typedef>

# 30: ODR

## Examples

◦ ◦

foo.h

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
void foo() { std::cout << "foo"; }
void bar();
#endif
```

foo.cpp

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
void foo() { std::cout << "foo"; }
void bar();
#endif
```

main.cpp

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
void foo() { std::cout << "foo"; }
void bar();
#endif
```

foofoo.h foo.cpp main.cpp ◦ foo◦ foo.h foo.cpp main.cpp foo.h ◦ foo◦

.cpp◦

inline◦ ◦ ◦

◦

foo.h

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
inline void foo() { std::cout << "foo"; }
void bar();
#endif
```

foo.cpp

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
inline void foo() { std::cout << "foo"; }
void bar();
#endif
```

main.cpp

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
inline void foo() { std::cout << "foo"; }
void bar();
#endif
```

foobar◦ foo.cppmain.cppfoo◦

◦

```
#ifndef FOO_H
#define FOO_H
#include <iostream>
inline void foo() { std::cout << "foo"; }
void bar();
#endif
```

Foo::baz◦

## ODR

ODR◦ func

- header.h

```
void overloaded(int);
inline void func() { overloaded('*'); }
```

- Foo.cpp

```
void overloaded(int);
inline void func() { overloaded('*'); }
```

- bar.cpp

```
void overloaded(int);
inline void func() { overloaded('*'); }
```

ODR<sub>overloaded</sub>◦

ODR <https://riptutorial.com/zh-TW/cplusplus/topic/4907/-odr->

## Examples

C++ 11

C++ 11◦ C++ 03 Rule of Three C++ 11 Five of Rule◦

◦ ◦ ◦

```
class Person
{
    char* name;
    int age;

public:
    // Destructor
    ~Person() { delete [] name; }

    // Implement Copy Semantics
    Person(Person const& other)
        : name(new char[strlen(other.name) + 1])
        , age(other.age)
    {
        std::strcpy(name, other.name);
    }

    Person &operator=(Person const& other)
    {
        // Use copy and swap idiom to implement assignment.
        Person copy(other);
        swap(*this, copy);
        return *this;
    }

    // Implement Move Semantics
    // Note: It is usually best to mark move operators as noexcept
    //       This allows certain optimizations in the standard library
    //       when the class is used in a container.

    Person(Person&& that) noexcept
        : name(nullptr) // Set the state so we know it is undefined
        , age(0)
    {
        swap(*this, that);
    }

    Person& operator=(Person&& that) noexcept
    {
        swap(*this, that);
        return *this;
    }

    friend void swap(Person& lhs, Person& rhs) noexcept
    {
        std::swap(lhs.name, rhs.name);
    }
}
```

```

        std::swap(lhs.age, rhs.age);
    }
};

°

class Person
{
    char* name;
    int age;

public:
    // Destructor
    ~Person() { delete [] name; }

    // Implement Copy Semantics
    Person(Person const& other)
        : name(new char[std::strlen(other.name) + 1])
        , age(other.age)
    {
        std::strcpy(name, other.name);
    }

    Person &operator=(Person const& other)
    {
        // Use copy and swap idiom to implement assignment.
        Person copy(other);
        swap(*this, copy);
        return *this;
    }

    // Implement Move Semantics
    // Note: It is usually best to mark move operators as noexcept
    //       This allows certain optimizations in the standard library
    //       when the class is used in a container.

    Person(Person&& that) noexcept
        : name(nullptr) // Set the state so we know it is undefined
        , age(0)
    {
        swap(*this, that);
    }

    Person& operator=(Person&& that) noexcept
    {
        swap(*this, that);
        return *this;
    }

    friend void swap(Person& lhs, Person& rhs) noexcept
    {
        std::swap(lhs.name, rhs.name);
        std::swap(lhs.age, rhs.age);
    }
};

```

“””。° °

C ++ 11

RAII◦ default◦

## Rule of Three Personcstrings

```
class cstring {  
private:  
    char* p;  
  
public:  
    ~cstring() { delete [] p; }  
    cstring(cstring const& );  
    cstring(cstring&& );  
    cstring& operator=(cstring const& );  
    cstring& operator=(cstring&& );  
  
    /* other members as appropriate */  
};
```

Person

```
class cstring {  
private:  
    char* p;  
  
public:  
    ~cstring() { delete [] p; }  
    cstring(cstring const& );  
    cstring(cstring&& );  
    cstring& operator=(cstring const& );  
    cstring& operator=(cstring&& );  
  
    /* other members as appropriate */  
};
```

Person◦ Person◦ ◦

```
class cstring {  
private:  
    char* p;  
  
public:  
    ~cstring() { delete [] p; }  
    cstring(cstring const& );  
    cstring(cstring&& );  
    cstring& operator=(cstring const& );  
    cstring& operator=(cstring&& );  
  
    /* other members as appropriate */  
};
```

cstring delete d/Person◦

R. Martinho Fernandes

C ++ 03

## Rule of Three。

- RAIIL◦

```
class Person
{
    char* name;
    int age;

public:
    Person(char const* new_name, int new_age)
        : name(new char[strlen(new_name) + 1])
        , age(new_age)
    {
        std::strcpy(name, new_name);
    }

    ~Person() {
        delete [] name;
    }
};
```

name◦

```
class Person
{
    char* name;
    int age;

public:
    Person(char const* new_name, int new_age)
        : name(new char[strlen(new_name) + 1])
        , age(new_age)
    {
        std::strcpy(name, new_name);
    }

    ~Person() {
        delete [] name;
    }
};
```

p1◦ p2p1◦ C ++◦ p1.namep2.name◦

main◦ p2;◦ p1◦ ◦ delete◦

◦ Person◦ ◦ ◦

```
class Person
{
    char* name;
    int age;

public:
    Person(char const* new_name, int new_age)
        : name(new char[strlen(new_name) + 1])
```

```
, age(new_age)
{
    std::strcpy(name, new_name);
}

~Person() {
    delete [] name;
}
};
```

- ◦ \*thiscopycopycopy ◦ copy\*this◦

- 

```
SomeType t = ...;
t = t;
```

- Person◦

- 

```
SomeType t = ...;
t = t;
```

- ◦ ◦

- copy and swap idiom◦ ◦ ◦

## C++ 11

- std::swap /◦ ◦

- 

<https://riptutorial.com/zh-TW/cplusplus/topic/1206-->

# 32: C ++C ++ 11C ++ 14C ++ 17C ++

## Examples

### C ++c

```
for(size_t i = 0; i < c.size(); ++i) c[i] = 0;
```

```
for(size_t i = 0; i < c.size(); ++i) c[i] = 0;
```

```
for(size_t i = 0; i < c.size(); ++i) c[i] = 0;
```

### C ++ 11forauto

```
for(size_t i = 0; i < c.size(); ++i) c[i] = 0;
```

c x o o

### C ++ 11

```
for(size_t i = 0; i < c.size(); ++i) c[i] = 0;
```

auto begin = c.begin(), end = c.end(); **force**beginendend. for/. C ++ 17

```
for(size_t i = 0; i < c.size(); ++i) c[i] = 0;
```

beginend. /.

**C ++C ++ 11C ++ 14C ++ 17C ++** <https://riptutorial.com/zh-TW/cplusplus/topic/7134/c-plusplus-c-plusplus-11c-plusplus-14c-plusplus-17c-plusplus>

## 33:

# ***std :: shared\_mutex*** ***std :: shared\_timed\_mutex***。

◦  
RWLock。  
std :: shared\_mutexshared\_timed\_mutex。  
std :: shared\_timed\_mutex'standard shared\_mutex + time method'。

## **std :: shared\_mutexMSVC14.1。**

```
class shared_mutex
{
public:
typedef _Smtx_t * native_handle_type;

shared_mutex() __NOEXCEPT
: _Myhandle(0)
{ // default construct
}

~shared_mutex() __NOEXCEPT
{ // destroy the object
}

void lock() __NOEXCEPT
{ // lock exclusive
    _Smtx_lock_exclusive(&_Myhandle);
}

bool try_lock() __NOEXCEPT
{ // try to lock exclusive
    return (_Smtx_try_lock_exclusive(&_Myhandle) != 0);
}

void unlock() __NOEXCEPT
{ // unlock exclusive
    _Smtx_unlock_exclusive(&_Myhandle);
}

void lock_shared() __NOEXCEPT
{ // lock non-exclusive
    _Smtx_lock_shared(&_Myhandle);
}

bool try_lock_shared() __NOEXCEPT
{ // try to lock non-exclusive
}
```

```

return (_Smtx_try_lock_shared(&_Myhandle) != 0);
}

void unlock_shared() _NOEXCEPT
{   // unlock non-exclusive
    _Smtx_unlock_shared(&_Myhandle);
}

native_handle_type native_handle() _NOEXCEPT
{   // get native handle
    return (&_Myhandle);
}

shared_mutex(const shared_mutex&) = delete;
shared_mutex& operator=(const shared_mutex&) = delete;
private:
    _Smtx_t _Myhandle;
};

void __cdecl _Smtx_lock_exclusive(_Smtx_t * smtx)
{   /* lock shared mutex exclusively */
AcquireSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smthx));
}

void __cdecl _Smtx_lock_shared(_Smtx_t * smtx)
{   /* lock shared mutex non-exclusively */
AcquireSRWLockShared(reinterpret_cast<PSRWLOCK>(smthx));
}

int __cdecl _Smtx_try_lock_exclusive(_Smtx_t * smtx)
{   /* try to lock shared mutex exclusively */
return (TryAcquireSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smthx)));
}

int __cdecl _Smtx_try_lock_shared(_Smtx_t * smtx)
{   /* try to lock shared mutex non-exclusively */
return (TryAcquireSRWLockShared(reinterpret_cast<PSRWLOCK>(smthx)));
}

void __cdecl _Smtx_unlock_exclusive(_Smtx_t * smtx)
{   /* unlock exclusive shared mutex */
ReleaseSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smthx));
}

void __cdecl _Smtx_unlock_shared(_Smtx_t * smtx)
{   /* unlock non-exclusive shared mutex */
ReleaseSRWLockShared(reinterpret_cast<PSRWLOCK>(smthx));
}

```

std :: shared\_mutex Windows Slim Reader / Write Locks [https://msdn.microsoft.com/ko-kr/library/windows/desktop/aa904937\(v=vs.85\).aspx](https://msdn.microsoft.com/ko-kr/library/windows/desktop/aa904937(v=vs.85).aspx)

std :: shared\_timed\_mutex。

## std :: shared\_timed\_mutex MSVC14.1。

```

class shared_mutex
{

```

```

public:
typedef _Smtx_t * native_handle_type;

shared_mutex() __NOEXCEPT
: _Myhandle(0)
{ // default construct
}

~shared_mutex() __NOEXCEPT
{ // destroy the object
}

void lock() __NOEXCEPT
{ // lock exclusive
_Smtx_lock_exclusive(&_Myhandle);
}

bool try_lock() __NOEXCEPT
{ // try to lock exclusive
return (_Smtx_try_lock_exclusive(&_Myhandle) != 0);
}

void unlock() __NOEXCEPT
{ // unlock exclusive
_Smtx_unlock_exclusive(&_Myhandle);
}

void lock_shared() __NOEXCEPT
{ // lock non-exclusive
_Smtx_lock_shared(&_Myhandle);
}

bool try_lock_shared() __NOEXCEPT
{ // try to lock non-exclusive
return (_Smtx_try_lock_shared(&_Myhandle) != 0);
}

void unlock_shared() __NOEXCEPT
{ // unlock non-exclusive
_Smtx_unlock_shared(&_Myhandle);
}

native_handle_type native_handle() __NOEXCEPT
{ // get native handle
return (&_Myhandle);
}

shared_mutex(const shared_mutex&) = delete;
shared_mutex& operator=(const shared_mutex&) = delete;
private:
_Smtx_t _Myhandle;
};

void __cdecl _Smtx_lock_exclusive(_Smtx_t * smtx)
{ /* lock shared mutex exclusively */
AcquireSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smth));
}

void __cdecl _Smtx_lock_shared(_Smtx_t * smtx)
{ /* lock shared mutex non-exclusively */
AcquireSRWLockShared(reinterpret_cast<PSRWLOCK>(smth));
}

```

```

}

int __cdecl _Smtx_try_lock_exclusive(_Smtx_t * smtx)
{   /* try to lock shared mutex exclusively */
return (TryAcquireSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smtx)));
}

int __cdecl _Smtx_try_lock_shared(_Smtx_t * smtx)
{   /* try to lock shared mutex non-exclusively */
return (TryAcquireSRWLockShared(reinterpret_cast<PSRWLOCK>(smtx)));
}

void __cdecl _Smtx_unlock_exclusive(_Smtx_t * smtx)
{   /* unlock exclusive shared mutex */
ReleaseSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smtx));
}

void __cdecl _Smtx_unlock_shared(_Smtx_t * smtx)
{   /* unlock non-exclusive shared mutex */
ReleaseSRWLockShared(reinterpret_cast<PSRWLOCK>(smtx));
}

```

**std :: shared\_timed\_mutex/std :: condition\_value。**

- 
- 

STLSharedMutex READ :	486647
STLSharedMutex WRITE :	205986
TOTAL READ&WRITE :	692633
STLSharedTimedMutex READ :	140291
STLSharedTimedMutex WRITE :	178849
TOTAL READ&WRITE :	319140

1000/。

**std :: shared\_mutex/std :: shared\_timed\_mutex2。**

- /◦
- 
- 

```

class shared_mutex
{
public:
typedef _Smtx_t * native_handle_type;

shared_mutex() __NOEXCEPT
: _Myhandle(0)
{   // default construct
}

```

```

~shared_mutex() _NOEXCEPT
{
    // destroy the object
}

void lock() _NOEXCEPT
{
    // lock exclusive
    _Smtx_lock_exclusive(&_Myhandle);
}

bool try_lock() _NOEXCEPT
{
    // try to lock exclusive
    return (_Smtx_try_lock_exclusive(&_Myhandle) != 0);
}

void unlock() _NOEXCEPT
{
    // unlock exclusive
    _Smtx_unlock_exclusive(&_Myhandle);
}

void lock_shared() _NOEXCEPT
{
    // lock non-exclusive
    _Smtx_lock_shared(&_Myhandle);
}

bool try_lock_shared() _NOEXCEPT
{
    // try to lock non-exclusive
    return (_Smtx_try_lock_shared(&_Myhandle) != 0);
}

void unlock_shared() _NOEXCEPT
{
    // unlock non-exclusive
    _Smtx_unlock_shared(&_Myhandle);
}

native_handle_type native_handle() _NOEXCEPT
{
    // get native handle
    return (&_Myhandle);
}

shared_mutex(const shared_mutex&) = delete;
shared_mutex& operator=(const shared_mutex&) = delete;
private:
    _Smtx_t _Myhandle;
};

void __cdecl _Smtx_lock_exclusive(_Smtx_t * smtx)
{
    /* lock shared mutex exclusively */
    AcquireSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smth));
}

void __cdecl _Smtx_lock_shared(_Smtx_t * smtx)
{
    /* lock shared mutex non-exclusively */
    AcquireSRWLockShared(reinterpret_cast<PSRWLOCK>(smth));
}

int __cdecl _Smtx_try_lock_exclusive(_Smtx_t * smtx)
{
    /* try to lock shared mutex exclusively */
    return (TryAcquireSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smth)));
}

int __cdecl _Smtx_try_lock_shared(_Smtx_t * smtx)

```

```

    /* try to lock shared mutex non-exclusively */
return (TryAcquireSRWLockShared(reinterpret_cast<PSRWLOCK>(smtx)));
}

void __cdecl _Smtx_unlock_exclusive(_Smtx_t * smtx)
{
    /* unlock exclusive shared mutex */
ReleaseSRWLockExclusive(reinterpret_cast<PSRWLOCK>(smtx));
}

void __cdecl _Smtx_unlock_shared(_Smtx_t * smtx)
{
    /* unlock non-exclusive shared mutex */
ReleaseSRWLockShared(reinterpret_cast<PSRWLOCK>(smtx));
}

```

## Examples

### **std :: unique\_lockstd :: shared\_lockstd :: lock\_guard**

RAlltrytry。

```

std::unique_lock<std::shared_mutex> lock{ mtx };

std::shared_lock<std::shared_mutex> l{ mtx };

```

```

#include <unordered_map>
#include <mutex>
#include <shared_mutex>
#include <thread>
#include <string>
#include <iostream>

class PhoneBook {
public:
    std::string getPhoneNo( const std::string & name )
    {
        std::shared_lock<std::shared_timed_mutex> l{_protect};
        auto it = _phonebook.find( name );
        if ( it != _phonebook.end() )
            return (*it).second;
        return "";
    }
    void addPhoneNo ( const std::string & name, const std::string & phone )
    {
        std::unique_lock<std::shared_timed_mutex> l{_protect};
        _phonebook[name] = phone;
    }
    std::shared_timed_mutex _protect;
    std::unordered_map<std::string, std::string> _phonebook;
};

```

### **std :: try\_to\_lockstd :: adopt\_lockstd :: defer\_lock**

```
std::unique_lock std::try_to_lock std::defer_lock std::adopt_lock
```

## 1. std::try\_to\_lock

```
{  
    std::atomic_int temp {0};  
    std::mutex _mutex;  
  
    std::thread t( [&] () {  
  
        while( temp != -1){  
            std::this_thread::sleep_for(std::chrono::seconds(5));  
            std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);  
  
            if(lock.owns_lock()){  
                //do something  
                temp=0;  
            }  
        }  
    });  
  
    while ( true )  
    {  
        std::this_thread::sleep_for(std::chrono::seconds(1));  
        std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);  
        if(lock.owns_lock()){  
            if (temp < INT_MAX){  
                ++temp;  
            }  
            std::cout << temp << std::endl;  
        }  
    }  
}
```

## 2. std::defer\_lock

```
{  
    std::atomic_int temp {0};  
    std::mutex _mutex;  
  
    std::thread t( [&] () {  
  
        while( temp != -1){  
            std::this_thread::sleep_for(std::chrono::seconds(5));  
            std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);  
  
            if(lock.owns_lock()){  
                //do something  
                temp=0;  
            }  
        }  
    });  
  
    while ( true )  
    {  
        std::this_thread::sleep_for(std::chrono::seconds(1));  
        std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);  
        if(lock.owns_lock()){  
            if (temp < INT_MAX){  
                ++temp;  
            }  
            std::cout << temp << std::endl;  
        }  
    }  
}
```

```

        ++temp;
    }
    std::cout << temp << std::endl;
}
}

{
std::atomic_int temp {0};
std::mutex _mutex;

std::thread t( [&] () {

    while( temp!= -1){
        std::this_thread::sleep_for(std::chrono::seconds(5));
        std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);

        if(lock.owns_lock()){
            //do something
            temp=0;
        }
    }
});

while ( true )
{
    std::this_thread::sleep_for(std::chrono::seconds(1));
    std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);
    if(lock.owns_lock()){
        if (temp < INT_MAX){
            ++temp;
        }
        std::cout << temp << std::endl;
    }
}
}

```

### 3. std::adopt\_lock

```

{
std::atomic_int temp {0};
std::mutex _mutex;

std::thread t( [&] () {

    while( temp!= -1){
        std::this_thread::sleep_for(std::chrono::seconds(5));
        std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);

        if(lock.owns_lock()){
            //do something
            temp=0;
        }
    }
});

while ( true )
{
    std::this_thread::sleep_for(std::chrono::seconds(1));
}

```

```

        std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);
        if(lock.owns_lock()) {
            if (temp < INT_MAX) {
                ++temp;
            }
            std::cout << temp << std::endl;
        }
    }
}

```

**std :: adopt\_lock** .

**std ::**

**std :: mutex** .

```

std::atomic_int temp{0};
std::mutex _mutex;

std::thread t( [&] () {

    while( temp!= -1){
        std::this_thread::sleep_for(std::chrono::seconds(5));
        std::unique_lock<std::mutex> lock( _mutex);

        temp=0;
    }
});


while ( true )
{
    std::this_thread::sleep_for(std::chrono::milliseconds(1));
    std::unique_lock<std::mutex> lock( _mutex, std::try_to_lock);
    if ( temp < INT_MAX )
        temp++;
    cout << temp << endl;
}

```

**std :: scoped\_lockC ++ 17**

**std::scoped\_lock** **RAll****std::lock** . **std::scoped\_lock** .

```

{
    std::scoped_lock lock{_mutex1,_mutex2};
    //do something
}

```

**C ++ 1x**

- **std :: mutex** - .
- **std :: timed\_mutex** - **try\_to\_lock**
- **std :: recursive\_mutex** - .

- std :: shared\_mutexstd :: shared\_timed\_mutex - ◦

**std ::**

```
std::lock o std::lock o
```

```
    std::lock(_mutex1, _mutex2);
```

<https://riptutorial.com/zh-TW/cplusplus/topic/9895/>

34:

7

```
struct foo {  
    unsigned x;  
    unsigned y;  
}  
  
static struct foo my_var;
```

```
struct foo {  
    unsigned x;  
    unsigned y;  
}  
static struct foo my_var;
```

`sizeof (unsigned) == 4` xy4.

```
struct foo {  
    unsigned x;  
    unsigned y;  
}  
  
static struct foo my_var;
```

xy.

```
struct foo {  
    unsigned x;  
    unsigned y;  
}  
static struct foo my_var;
```

xv 4° 18° v

```
struct foo {  
    unsigned x;  
    unsigned y;  
}  
static struct foo my_var;
```

## Examples

```
struct FileAttributes
```

```
{  
    unsigned int ReadOnly: 1;  
    unsigned int Hidden: 1;  
};
```

1。 : 1: 1: 1。 8int64int。 unsigned。

“1”。

```
struct FileAttributes  
{  
    unsigned int ReadOnly: 1;  
    unsigned int Hidden: 1;  
};
```

22sizeof4。

◦ ◦

```
struct FileAttributes  
{  
    unsigned int ReadOnly: 1;  
    unsigned int Hidden: 1;  
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2710/>

# 35:

```
std::bitset<bitset> °
```

```
#include <bitset>
```

```
std::bitset csetbitset °
```

•

## Examples

### C

OR | °

```
// Bit x will be set  
number |= 1LL << x;
```

## std :: bitset

```
set(x) set(x,true) - x1 °
```

```
// Bit x will be set  
number |= 1LL << x;
```

### C

AND & °

```
// Bit x will be cleared  
number &= ~(1LL << x);
```

## std :: bitset

```
reset(x) set(x,false) - x°
```

```
// Bit x will be cleared  
number &= ~(1LL << x);
```

---

**C****XOR** ^ .

```
// Bit x will be the opposite value of what it is currently  
number ^= 1LL << x;
```

---

## std :: bitset

```
// Bit x will be the opposite value of what it is currently  
number ^= 1LL << x;
```

---

**C****xAND** &

```
(number >> x) & 1LL; // 1 if the 'x'th bit of 'number' is set, 0 otherwise
```

◦ number >> x ◦

```
(number >> x) & 1LL; // 1 if the 'x'th bit of 'number' is set, 0 otherwise
```

◦

---

## std :: bitset

```
(number >> x) & 1LL; // 1 if the 'x'th bit of 'number' is set, 0 otherwise
```

**nx**

---

**C**

```
// Bit n will be set if x is 1 and cleared if x is 0.  
number ^= (-x ^ number) & (1LL << n);
```

---

## std :: bitset

set(n, val) - nval ◦

```
// Bit n will be set if x is 1 and cleared if x is 0.  
number ^= (-x ^ number) & (1LL << n);
```

## C

```
x = -1; // -1 == 1111 1111 ... 1111b
```

◦

## std :: bitset

```
x = -1; // -1 == 1111 1111 ... 1111b
```

## C

```
template <typename T>  
T rightmostSetBitRemoved(T n)  
{  
    // static_assert(std::is_integral<T>::value && !std::is_signed<T>::value, "type should be  
    unsigned"); // For c++11 and later  
    return n & (n - 1);  
}
```

- $n_0 \& 0xFF..FF$
- $n_0bxxxxxx10..00n - 10bxxxxxx011..11 n \& (n - 1)0bxxxxxx000..00$

◦

```
unsigned value = 1234;  
unsigned bits = 0; // accumulates the total number of bits set in `n'  
  
for (bits = 0; value; value >>= 1)  
    bits += value & 1;
```

---

```
unsigned value = 1234;  
unsigned bits = 0; // accumulates the total number of bits set in `n'  
  
for (bits = 0; value; value >>= 1)  
    bits += value & 1;
```

value◦

Peter WegnerCACM 3/322 - 1960CBrian W. KernighanDennis M. Ritchie◦

## 12

```
unsigned value = 1234;
unsigned bits = 0; // accumulates the total number of bits set in `n`

for (bits = 0; value; value >>= 1)
    bits += value & 1;
```

◦

---

### CPUx86popcnt ° g ++

```
unsigned value = 1234;
unsigned bits = 0; // accumulates the total number of bits set in `n`

for (bits = 0; value; value >>= 1)
    bits += value & 1;
```

## 2

n & (n - 1) 2

```
bool power_of_2 = n && !(n & (n - 1));
```

n && 02◦

◦ a01(1)0000101(0)00001 ° ° 1◦

```
*****
convert small letter to captial letter.
=====
a: 01100001
mask: 11011111 <-- (0xDF) 11(0)11111
      :-----
a&mask: 01000001 <-- A letter
*****
```

## A

```
*****
convert small letter to captial letter.
=====
a: 01100001
mask: 11011111 <-- (0xDF) 11(0)11111
      :-----
a&mask: 01000001 <-- A letter
*****
```

```
*****
convert small letter to captial letter.
=====
```

```
a: 01100001
mask: 11011111 <-- (0xDF) 11(0)11111
      -----
a&mask: 01000001 <-- A letter
*****
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3016/>

# 36:

◦

```
int a = ~0;
int b = a << 1;
```

6432x86PIC◦

>> XOR◦ Endian-nessendian-ness◦

## Examples

### - AND

```
int a = 6;      // 0110b  (0x06)
int b = 10;     // 1010b  (0x0A)
int c = a & b; // 0010b  (0x02)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

a = 6, b = 10, c = 2

AND

```
int a = 6;      // 0110b  (0x06)
int b = 10;     // 1010b  (0x0A)
int c = a & b; // 0010b  (0x02)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

a 0110 b 1010 AND “0010

```
int a = 6;      // 0110b  (0x06)
int b = 10;     // 1010b  (0x0A)
int c = a & b; // 0010b  (0x02)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

&=AND

```
int a = 6;      // 0110b  (0x06)
int b = 10;     // 1010b  (0x0A)
int c = a & b; // 0010b  (0x02)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

| - OR

```

int a = 5;      // 0101b  (0x05)
int b = 12;     // 1100b  (0x0C)
int c = a | b; // 1101b  (0x0D)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;

```

a = 5, b = 12, c = 13

OR

```

int a = 5;      // 0101b  (0x05)
int b = 12;     // 1100b  (0x0C)
int c = a | b; // 1101b  (0x0D)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;

```

a 0101 b 1100 OR '1101

```

int a = 5;      // 0101b  (0x05)
int b = 12;     // 1100b  (0x0C)
int c = a | b; // 1101b  (0x0D)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;

```

=OR

```

int a = 5;      // 0101b  (0x05)
int b = 12;     // 1100b  (0x0C)
int c = a | b; // 1101b  (0x0D)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;

```

**A -**

```

int a = 5;      // 0101b  (0x05)
int b = 9;      // 1001b  (0x09)
int c = a ^ b; // 1100b  (0x0C)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;

```

a = 5, b = 9, c = 12

XOR

```

int a = 5;      // 0101b  (0x05)
int b = 9;      // 1001b  (0x09)
int c = a ^ b; // 1100b  (0x0C)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;

```

**XOR**<sub>true OR true = false</sub> true AND/OR true = true **XOR**。

a 0101 b 1001 XOR '1100

```
int a = 5;      // 0101b  (0x05)
int b = 9;      // 1001b  (0x09)
int c = a ^ b; // 1100b  (0x0C)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

## $\wedge$ =XOR

```
int a = 5;      // 0101b  (0x05)
int b = 9;      // 1001b  (0x09)
int c = a ^ b; // 1100b  (0x0C)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

## XOR。

- std::swap()

## XOR

```
int a = 5;      // 0101b  (0x05)
int b = 9;      // 1001b  (0x09)
int c = a ^ b; // 1100b  (0x0C)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

◦

```
int a = 5;      // 0101b  (0x05)
int b = 9;      // 1001b  (0x09)
int c = a ^ b; // 1100b  (0x0C)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

- $x \wedge a \wedge c = a \wedge c$

## 2015+

```
int a = 5;      // 0101b  (0x05)
int b = 9;      // 1001b  (0x09)
int c = a ^ b; // 1100b  (0x0C)

std::cout << "a = " << a << ", b = " << b << ", c = " << c << std::endl;
```

## - NOT

```
unsigned char a = 234; // 1110 1010b  (0xEA)
unsigned char b = ~a; // 0001 0101b  (0x15)

std::cout << "a = " << static_cast<int>(a) <<
", b = " << static_cast<int>(b) << std::endl;
```

```
a = 234, b = 21
```

NOT。 1 0 0 1。 NOT

```
unsigned char a = 234; // 1110 1010b (0xEA)
unsigned char b = ~a; // 0001 0101b (0x15)

std::cout << "a = " << static_cast<int>(a) <<
    ", b = " << static_cast<int>(b) << std::endl;
```

NOT

```
unsigned char a = 234; // 1110 1010b (0xEA)
unsigned char b = ~a; // 0001 0101b (0x15)

std::cout << "a = " << static_cast<int>(a) <<
    ", b = " << static_cast<int>(b) << std::endl;
```

NOT<sub>a</sub> ≈ 10。

NOT ~ NOT ! ;NOTNOT (!1) != (~1)

<< -

```
int a = 1; // 0001b
int b = a << 1; // 0010b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

a = 1, b = 2

a 1 05 0000 0101 45 << 4 80 0101 0000。 12

```
int a = 1; // 0001b
int b = a << 1; // 0010b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

```
int a = 1; // 0001b
int b = a << 1; // 0010b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

a = 2147483647, b = -2

shift。

```
int a = 1; // 0001b
int b = a << 1; // 0010b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

<<=

```
int a = 1;      // 0001b
int b = a << 1; // 0010b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

>> -

```
int a = 2;      // 0010b
int b = a >> 1; // 0001b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

a = 2, b = 1

a 1;

```
int a = 2;      // 0010b
int b = a >> 1; // 0001b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

```
int a = 2;      // 0010b
int b = a >> 1; // 0001b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

>>=

```
int a = 2;      // 0010b
int b = a >> 1; // 0001b

std::cout << "a = " << a << ", b = " << b << std::endl;
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2572/>

# 37: OpenMP

OpenMP C++ API [OpenMP](#)

◦

OpenMP API [omp\\_get\\_thread\\_num\(\)](#) `omp.h`

OpenMP [OpenMP](#) pragma

- **GCC**-fopenmp
- **Clang**-fopenmp
- **MSVC**/openmp

## Examples

OpenMP

◦

OpenMP API `omp.h`

```
std::cout << "begin ";
// This pragma statement hints the compiler that the
// contents within the { } are to be executed in as
// parallel sections using openMP, the compiler will
// generate this chunk of code for parallel execution
#pragma omp parallel sections
{
    // This pragma statement hints the compiler that
    // this is a section that can be executed in parallel
    // with other section, a single section will be executed
    // by a single thread.
    // Note that it is "section" as opposed to "sections" above
    #pragma omp section
    {
        std::cout << "hello " << std::endl;
        /** Do something **/
    }
    #pragma omp section
    {
        std::cout << "world " << std::endl;
        /** Do something **/
    }
}
// This line will not be executed until all the
// sections defined above terminates
std::cout << "end" << std::endl;
```

2◦

## OpenMP

```
std::cout << "begin ";
//      Start of parallel sections
#pragma omp parallel sections
{
    //      Execute these sections in parallel
    #pragma omp section
    {
        ... do something ...
        std::cout << "hello ";
    }
    #pragma omp section
    {
        ... do something ...
        std::cout << "world ";
    }
    #pragma omp section
    {
        ... do something ...
        std::cout << "forever ";
    }
}
//      end of parallel sections
std::cout << "end";
```

- 
- 
- 
- 

## OpenMPParallel For Loop

```
//      Splits element vector into element.size() / Thread Qty
//      and allocate that range for each thread.
#pragma omp parallel for
for (size_t i = 0; i < element.size(); ++i)
    element[i] = ...

//      Example Allocation (100 element per thread)
//      Thread 1 : 0 ~ 99
//      Thread 2 : 100 ~ 199
//      Thread 3 : 200 ~ 299
//      ...
//      Continue process
//      Only when all threads completed their allocated
//      loop job
...
```

\*for。

## OpenMP/

```
std::vector<int> OpenMP() {
    int i;
    std::vector<int> Master;
    #pragma omp parallel
    {
        // In this area, you can write any code you want for each
        // slave thread, in this case a vector to hold each of their results
        // We don't have to worry about how many threads were spawn or if we need
        // to repeat this declaration or not.
        std::vector<int> Slave;
        #pragma omp for nowait
        for (size_t i = 0; i < 1000000; ++i)
        {
            /* Do something */
            ...
            Slave.push_back(...);
        }
        // Slaves that finished their part of the job
        // will perform this thread by thread one at a time
        // critical section ensures that only 0 or 1 thread performs
        // the {} at any time
        #pragma omp critical
        {
            // Merge slave into master
            // use move iterators instead, avoid copy unless
            // you want to use it for something else after this section
            Master.insert(Master.end(),
                          std::make_move_iterator(Slave.begin()),
                          std::make_move_iterator(Slave.end()));
        }
    }
    // Have fun with Master vector
    ...
}
```

OpenMP <https://riptutorial.com/zh-TW/cplusplus/topic/8222/openmp>

## **38: std :: unordered\_map**

## std :: unordered\_map

- O1◦ ◦ operator[]◦

## Examples

○ ○

```
unordered_map<string, int> first; //declaration of the map
first["One"] = 1; // [] operator used to insert the value
first["Two"] = 2;
first["Three"] = 3;
first["Four"] = 4;
first["Five"] = 5;

pair <string,int> bar = make_pair("Nine", 9); //make a pair of same type
first.insert(bar); //can also use insert to feed the values
```

```
unordered_map<data_type, data_type> variable_name; //declaration

variable_name[key_value] = mapped_value; //inserting values

variable_name.find(key_value); //returns iterator to the key value

variable_name.begin(); // iterator to the first element

variable_name.end(); // iterator to the last + 1 element
```

std :: unordered\_map <https://riptutorial.com/zh-TW/cplusplus/topic/10540/std---unordered-map>

## Examples

system\_clock

### C++ 11

```
#include <iostream>
#include <chrono>
#include <thread>

int main() {
    auto start = std::chrono::system_clock::now(); // This and "end"'s type is
std::chrono::time_point
    { // The code to test
        std::this_thread::sleep_for(std::chrono::seconds(2));
    }
    auto end = std::chrono::system_clock::now();

    std::chrono::duration<double> elapsed = end - start;
    std::cout << "Elapsed time: " << elapsed.count() << "s";
}
```

sleep\_forstd::chrono::seconds

◦ ////◦

2000◦

```
#include <iostream>
#include <string>
#include <chrono>
#include <ctime>

/***
 * Creates a std::tm structure from raw date.
 *
 * \param year (must be 1900 or greater)
 * \param month months since January - [1, 12]
 * \param day day of the month - [1, 31]
 * \param minutes minutes after the hour - [0, 59]
 * \param seconds seconds after the minute - [0, 61] (until C++11) / [0, 60] (since C++11)
 *
 * Based on http://en.cppreference.com/w/cpp/chrono/c/tm
 */
std::tm CreateTmStruct(int year, int month, int day, int hour, int minutes, int seconds) {
    struct tm tm_ret = {0};

    tm_ret.tm_sec = seconds;
    tm_ret.tm_min = minutes;
    tm_ret.tm_hour = hour;
    tm_ret.tm_mday = day;
    tm_ret.tm_mon = month - 1;
```

```

tm_ret.tm_year = year - 1900;

return tm_ret;
}

int get_days_in_year(int year) {

using namespace std;
using namespace std::chrono;

// We want results to be in days
typedef duration<int, ratio_multiply<hours::period, ratio<24> >::type> days;

// Create start time span
std::tm tm_start = CreateTmStruct(year, 1, 1, 0, 0, 0);
auto tms = system_clock::from_time_t(std::mktime(&tm_start));

// Create end time span
std::tm tm_end = CreateTmStruct(year + 1, 1, 1, 0, 0, 0);
auto tme = system_clock::from_time_t(std::mktime(&tm_end));

// Calculate time duration between those two dates
auto diff_in_days = std::chrono::duration_cast<days>(tme - tms);

return diff_in_days.count();
}

int main()
{
    for ( int year = 2000; year <= 2016; ++year )
        std::cout << "There are " << get_days_in_year(year) << " days in " << year << "\n";
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/3936/-chrono->

# 40:

using。

- typename *opt nested-name-specifier unqualified-id* ;
- using :: *unqualified-id* ;

*using* using 。 using-directive using namespace。

*using* typedef。 。

## Examples

```
using cout std main std::cout cout。
```

```
#include <iostream>
int main() {
    using std::cout;
    cout << "Hello, world!\n";
}
```

*using* 。 using std::cout 。

◦ d1.fooDerived1::foo(const char\*)◦ Base::foo(int)◦ d2.foo(42) using Base::foo(int) Derived2foo◦ fooBase::foo 。

```
struct Base {
    void foo(int);
};

struct Derived1 : Base {
    void foo(const char*);
};

struct Derived2 : Base {
    using Base::foo;
    void foo(const char*);
};

int main() {
    Derived1 d1;
    d1.foo(42); // error
    Derived2 d2;
    d2.foo(42); // OK
}
```

## C++ 11

*using*◦ 。

```
struct Base {
    Base(int x, const char* s);
};
```

```
struct Derived1 : Base {
    Derived1(int x, const char* s) : Base(x, s) {}
};

struct Derived2 : Base {
    using Base::Base;
};

int main() {
    Derived1 d1(42, "Hello, world");
    Derived2 d2(42, "Hello, world");
}
```

Derived1Derived2Base。 Derived1Derived2**C ++ 11**。

<https://riptutorial.com/zh-TW/cplusplus/topic/9301/>

## Examples

try/catch◦ try catch◦

```
#include <iostream>
#include <string>
#include <stdexcept>

int main() {
    std::string str("foo");

    try {
        str.at(10); // access element, may throw std::out_of_range
    } catch (const std::out_of_range& e) {
        // what() is inherited from std::exception and contains an explanatory message
        std::cout << e.what();
    }
}
```

catch◦ catch

```
#include <iostream>
#include <string>
#include <stdexcept>

int main() {
    std::string str("foo");

    try {
        str.at(10); // access element, may throw std::out_of_range
    } catch (const std::out_of_range& e) {
        // what() is inherited from std::exception and contains an explanatory message
        std::cout << e.what();
    }
}
```

catch◦ catchstd::length\_errorstd::out\_of\_rangestd::exception

```
#include <iostream>
#include <string>
#include <stdexcept>

int main() {
    std::string str("foo");

    try {
        str.at(10); // access element, may throw std::out_of_range
    } catch (const std::out_of_range& e) {
        // what() is inherited from std::exception and contains an explanatory message
        std::cout << e.what();
    }
}
```

## catch catch

```
#include <iostream>
#include <string>
#include <stdexcept>

int main() {
    std::string str("foo");

    try {
        str.at(10); // access element, may throw std::out_of_range
    } catch (const std::out_of_range& e) {
        // what() is inherited from std::exception and contains an explanatory message
        std::cout << e.what();
    }
}
```

## catch-all

```
#include <iostream>
#include <string>
#include <stdexcept>

int main() {
    std::string str("foo");

    try {
        str.at(10); // access element, may throw std::out_of_range
    } catch (const std::out_of_range& e) {
        // what() is inherited from std::exception and contains an explanatory message
        std::cout << e.what();
    }
}
```

## Rethrow

◦

```
try {
    ... // some code here
} catch (const SomeException& e) {
    std::cout << "caught an exception";
    throw;
}
```

throw;

## C++ 11

std::exception\_ptr C++rethrow\_exception<exception>;

```
try {
    ... // some code here
} catch (const SomeException& e) {
    std::cout << "caught an exception";
```

```
        throw;  
    }
```

## Try Blocks

```
struct A : public B  
{  
    A() try : B(), foo(1), bar(2)  
    {  
        // constructor body  
    }  
    catch (...)  
    {  
        // exceptions from the initializer list and constructor are caught here  
        // if no exception is thrown here  
        // then the caught exception is re-thrown.  
    }  
  
private:  
    Foo foo;  
    Bar bar;  
};
```

```
void function_with_try_block()  
try  
{  
    // try block body  
}  
catch (...)  
{  
    // catch block body  
}
```

```
void function_with_try_block()  
try  
{  
    // try block body  
}  
catch (...)  
{  
    // catch block body  
}
```

catchcatch◦

```
maintrymaintry◦ std::terminate ◦
```

```
struct A  
{  
    ~A() noexcept(false) try  
    {  
        // destructor body  
    }  
    catch (...)  
    {
```

```
// exceptions of destructor body are caught here  
// if no exception is thrown here  
// then the caught exception is re-thrown.  
}  
};
```

std::terminate .

## const

const .

```
try {  
    // throw new std::runtime_error("Error!");    // Don't do this!  
    // This creates an exception object  
    // on the heap and would require you to catch the  
    // pointer and manage the memory yourself. This can  
    // cause memory leaks!  
  
    throw std::runtime_error("Error!");  
} catch (const std::runtime_error& e) {  
    std::cout << e.what() << std::endl;  
}
```

catch catch . throw e; ;throw e;

```
try {  
    // throw new std::runtime_error("Error!");    // Don't do this!  
    // This creates an exception object  
    // on the heap and would require you to catch the  
    // pointer and manage the memory yourself. This can  
    // cause memory leaks!  
  
    throw std::runtime_error("Error!");  
} catch (const std::runtime_error& e) {  
    std::cout << e.what() << std::endl;  
}
```

const .

catch . logic\_error runtime\_error bad\_alloc I/O .

## C++ 11

Web . callstack .

.

std::nested\_exception std::throw\_with\_nested

```
#include <stdexcept>  
#include <exception>  
#include <string>
```

```

#include <fstream>
#include <iostream>

struct MyException
{
    MyException(const std::string& message) : message(message) {}
    std::string message;
};

void print_current_exception(int level)
{
    try {
        throw;
    } catch (const std::exception& e) {
        std::cerr << std::string(level, ' ') << "exception: " << e.what() << '\n';
    } catch (const MyException& e) {
        std::cerr << std::string(level, ' ') << "MyException: " << e.message << '\n';
    } catch (...) {
        std::cerr << "Unknown exception\n";
    }
}

void print_current_exception_with_nested(int level = 0)
{
    try {
        throw;
    } catch (...) {
        print_current_exception(level);
    }
    try {
        throw;
    } catch (const std::nested_exception& nested) {
        try {
            nested.rethrow_nested();
        } catch (...) {
            print_current_exception_with_nested(level + 1); // recursion
        }
    } catch (...) {
        //Empty // End recursion
    }
}

// sample function that catches an exception and wraps it in a nested exception
void open_file(const std::string& s)
{
    try {
        std::ifstream file(s);
        file.exceptions(std::ios_base::failbit);
    } catch(...) {
        std::throw_with_nested(MyException{"Couldn't open " + s});
    }
}

// sample function that catches an exception and wraps it in a nested exception
void run()
{
    try {
        open_file("nonexistent.file");
    } catch(...) {
        std::throw_with_nested( std::runtime_error("run() failed") );
    }
}

```

```

}

// runs the sample function above and prints the caught exception
int main()
{
    try {
        run();
    } catch(...) {
        print_current_exception_with_nested();
    }
}

```

```

#include <stdexcept>
#include <exception>
#include <string>
#include <fstream>
#include <iostream>

struct MyException
{
    MyException(const std::string& message) : message(message) {}
    std::string message;
};

void print_current_exception(int level)
{
    try {
        throw;
    } catch (const std::exception& e) {
        std::cerr << std::string(level, ' ') << "exception: " << e.what() << '\n';
    } catch (const MyException& e) {
        std::cerr << std::string(level, ' ') << "MyException: " << e.message << '\n';
    } catch (...) {
        std::cerr << "Unknown exception\n";
    }
}

void print_current_exception_with_nested(int level = 0)
{
    try {
        throw;
    } catch (...) {
        print_current_exception(level);
    }
    try {
        throw;
    } catch (const std::nested_exception& nested) {
        try {
            nested.rethrow_nested();
        } catch (...) {
            print_current_exception_with_nested(level + 1); // recursion
        }
    } catch (...) {
        //Empty // End recursion
    }
}

// sample function that catches an exception and wraps it in a nested exception
void open_file(const std::string& s)
{

```

```

try {
    std::ifstream file(s);
    file.exceptions(std::ios_base::failbit);
} catch(...) {
    std::throw_with_nested(MyException("Couldn't open " + s));
}

// sample function that catches an exception and wraps it in a nested exception
void run()
{
    try {
        open_file("nonexistent.file");
    } catch(...) {
        std::throw_with_nested( std::runtime_error("run() failed") );
    }
}

// runs the sample function above and prints the caught exception
int main()
{
    try {
        run();
    } catch(...) {
        print_current_exception_with_nested();
    }
}

```

std::exception exception std::exception °

## std :: uncaught\_exceptions

### C++ 17

**C++ 17** int std::uncaught\_exceptions() bool std::uncaught\_exception() ° °

```

#include <exception>
#include <string>
#include <iostream>

// Apply change on destruction:
// Rollback in case of exception (failure)
// Else Commit (success)
class Transaction
{
public:
    Transaction(const std::string& s) : message(s) {}
    Transaction(const Transaction&) = delete;
    Transaction& operator =(const Transaction&) = delete;
    void Commit() { std::cout << message << ": Commit\n"; }
    void RollBack() noexcept(true) { std::cout << message << ": Rollback\n"; }

    // ...

    ~Transaction() {
        if (uncaughtExceptionCount == std::uncaught_exceptions()) {
            Commit(); // May throw.
        } else { // current stack unwinding
    }
}
```

```

        RollBack();
    }
}

private:
    std::string message;
    int uncaughtExceptionCount = std::uncaught_exceptions();
};

class Foo
{
public:
    ~Foo() {
        try {
            Transaction transaction("In ~Foo"); // Commit,
                                            // even if there is an uncaught exception
            //...
        } catch (const std::exception& e) {
            std::cerr << "exception/~Foo:" << e.what() << std::endl;
        }
    }
};

int main()
{
    try {
        Transaction transaction("In main"); // RollBack
        Foo foo; // ~Foo commit its transaction.
        //...
        throw std::runtime_error("Error");
    } catch (const std::exception& e) {
        std::cerr << "exception/main:" << e.what() << std::endl;
    }
}

```

```

#include <exception>
#include <string>
#include <iostream>

// Apply change on destruction:
// Rollback in case of exception (failure)
// Else Commit (success)
class Transaction
{
public:
    Transaction(const std::string& s) : message(s) {}
    Transaction(const Transaction&) = delete;
    Transaction& operator =(const Transaction&) = delete;
    void Commit() { std::cout << message << ": Commit\n"; }
    void RollBack() noexcept(true) { std::cout << message << ": Rollback\n"; }

    // ...

    ~Transaction() {
        if (uncaughtExceptionCount == std::uncaught_exceptions()) {
            Commit(); // May throw.
        } else { // current stack unwinding
            RollBack();
        }
    }
}

```

```

private:
    std::string message;
    int uncaughtExceptionCount = std::uncaught_exceptions();
};

class Foo
{
public:
    ~Foo() {
        try {
            Transaction transaction("In ~Foo"); // Commit,
                                            // even if there is an uncaught exception
            //...
        } catch (const std::exception& e) {
            std::cerr << "exception/~Foo:" << e.what() << std::endl;
        }
    }
};

int main()
{
    try {
        Transaction transaction("In main"); // RollBack
        Foo foo; // ~Foo commit its transaction.
        //...
        throw std::runtime_error("Error");
    } catch (const std::exception& e) {
        std::cerr << "exception/main:" << e.what() << std::endl;
    }
}

```

◦

std::exception◦ std::exception

```

#include <exception>

class Except: virtual public std::exception {

protected:

    int error_number;           ///< Error number
    int error_offset;          ///< Error offset
    std::string error_message;  ///< Error message

public:

    /** Constructor (C++ STL string, int, int).
     *  @param msg The error message
     *  @param err_num Error number
     *  @param err_off Error offset
     */
    explicit
    Except(const std::string& msg, int err_num, int err_off):
        error_number(err_num),
        error_offset(err_off),
        error_message(msg)
    {}

```

```

/** Destructor.
 * Virtual to allow for subclassing.
 */
virtual ~Except() throw () {}

/** Returns a pointer to the (constant) error description.
 * @return A pointer to a const char*. The underlying memory
 * is in possession of the Except object. Callers must
 * not attempt to free the memory.
 */
virtual const char* what() const throw () {
    return error_message.c_str();
}

/** Returns error number.
 * @return #error_number
 */
virtual int getErrorNumber() const throw() {
    return error_number;
}

/** Returns error offset.
 * @return #error_offset
 */
virtual int getErrorOffset() const throw() {
    return error_offset;
}

};


```

```

#include <exception>

class Except: virtual public std::exception {

protected:

    int error_number;           ///< Error number
    int error_offset;          ///< Error offset
    std::string error_message; ///< Error message

public:

    /** Constructor (C++ STL string, int, int).
     * @param msg The error message
     * @param err_num Error number
     * @param err_off Error offset
     */
    explicit
    Except(const std::string& msg, int err_num, int err_off):
        error_number(err_num),
        error_offset(err_off),
        error_message(msg)
    {}

    /** Destructor.
     * Virtual to allow for subclassing.
     */
    virtual ~Except() throw () {}

```

```

/** Returns a pointer to the (constant) error description.
 * @return A pointer to a const char*. The underlying memory
 * is in possession of the Except object. Callers must
 * not attempt to free the memory.
 */
virtual const char* what() const throw () {
    return error_message.c_str();
}

/** Returns error number.
 * @return #error_number
 */
virtual int getErrorNumber() const throw() {
    return error_number;
}

/** Returns error offset.
 * @return #error_offset
 */
virtual int getErrorOffset() const throw() {
    return error_offset;
}

};


```

◦

std::runtime\_error

```

#include <exception>

class Except: virtual public std::exception {

protected:

    int error_number;           ///< Error number
    int error_offset;          ///< Error offset
    std::string error_message;  ///< Error message

public:

    /** Constructor (C++ STL string, int, int).
     * @param msg The error message
     * @param err_num Error number
     * @param err_off Error offset
     */
    explicit
    Except(const std::string& msg, int err_num, int err_off):
        error_number(err_num),
        error_offset(err_off),
        error_message(msg)
    {}

    /** Destructor.
     * Virtual to allow for subclassing.
     */
    virtual ~Except() throw () {}

    /** Returns a pointer to the (constant) error description.
     * @return A pointer to a const char*. The underlying memory
     */


```

```
*  is in possession of the Except object. Callers must
*  not attempt to free the memory.
*/
virtual const char* what() const throw () {
    return error_message.c_str();
}

/** Returns error number.
 *  @return #error_number
 */
virtual int getErrorNumber() const throw() {
    return error_number;
}

/** Returns error offset.
 *  @return #error_offset
 */
virtual int getErrorOffset() const throw() {
    return error_offset;
}

};
```

what() std::runtime\_error what()。。

<https://riptutorial.com/zh-TW/cplusplus/topic/1354/>

C++。

C++ 0x

## Examples

C++ 11

```
#include <mutex>
#include <condition_variable>

class Semaphore {
public:
    Semaphore (int count_ = 0)
        : count(count_)
    {
    }

    inline void notify( int tid ) {
        std::unique_lock<std::mutex> lock(mtx);
        count++;
        cout << "thread " << tid << " notify" << endl;
        //notify the waiting thread
        cv.notify_one();
    }

    inline void wait( int tid ) {
        std::unique_lock<std::mutex> lock(mtx);
        while(count == 0) {
            cout << "thread " << tid << " wait" << endl;
            //wait on the mutex until notify is called
            cv.wait(lock);
            cout << "thread " << tid << " run" << endl;
        }
        count--;
    }
private:
    std::mutex mtx;
    std::condition_variable cv;
    int count;
};

° ° notify_one() °
```

s1count1° notify°

```
int main()
{
    Semaphore sem(1);

    thread s1([&]() {
        while(true) {
            this_thread::sleep_for(std::chrono::seconds(5));
        }
    });
}
```

```

        sem.wait( 1 );
    }
}

thread s2([&]() {
    while(true) {
        sem.wait( 2 );
    }
});

thread s3([&]() {
    while(true) {
        this_thread::sleep_for(std::chrono::milliseconds(600));
        sem.wait( 3 );
    }
});

thread s4([&]() {
    while(true) {
        this_thread::sleep_for(std::chrono::seconds(5));
        sem.notify( 4 );
    }
});

s1.join();
s2.join();
s3.join();
s4.join();

...
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/9785/>

**43:**

## Examples

63

```
struct Inner {int i;};

const int NUM_INNER = 5;
class Value
{
private:
    Inner *array_; //Normally has reference semantics.

public:
    Value() : array_(new Inner[NUM_INNER]) {}

    ~Value() {delete[] array_;}

    Value(const Value &val) : array_(new Inner[NUM_INNER])
    {
        for(int i = 0; i < NUM_INNER; ++i)
            array_[i] = val.array_[i];
    }

    Value &operator=(const Value &val)
    {
        for(int i = 0; i < NUM_INNER; ++i)
            array_[i] = val.array_[i];
        return *this;
    }
};
```

C++ 11

Values.”

```
struct Inner {int i;};

const int NUM_INNER = 5;
class Value
{
private:
    Inner *array_; //Normally has reference semantics.

public:
    Value() : array_(new Inner[NUM_INNER]) {}

    ~Value() {delete[] array_;}

    Value(const Value &val) : array_(new Inner[NUM_INNER])
    {
        for(int i = 0; i < NUM_INNER; ++i)
            array_[i] = val.array_[i];
    }

    void print() const
    {
        for(int i = 0; i < NUM_INNER; ++i)
            cout << array_[i].i << endl;
    }
};
```

```

        array_[i] = val.array_[i];
    }

Value &operator=(const Value &val)
{
    for(int i = 0; i < NUM_INNER; ++i)
        array_[i] = val.array_[i];
    return *this;
}
};

```

◦

```

struct Inner {int i;};

const int NUM_INNER = 5;
class Value
{
private:
    Inner *array_; //Normally has reference semantics.

public:
    Value() : array_(new Inner[NUM_INNER]) {}

    ~Value() {delete[] array_;}

    Value(const Value &val) : array_(new Inner[NUM_INNER])
    {
        for(int i = 0; i < NUM_INNER; ++i)
            array_[i] = val.array_[i];
    }

    Value &operator=(const Value &val)
    {
        for(int i = 0; i < NUM_INNER; ++i)
            array_[i] = val.array_[i];
        return *this;
    }
};

```

## unique\_ptrZero

```

struct Inner {int i;};

const int NUM_INNER = 5;
class Value
{
private:
    Inner *array_; //Normally has reference semantics.

public:
    Value() : array_(new Inner[NUM_INNER]) {}

    ~Value() {delete[] array_;}

    Value(const Value &val) : array_(new Inner[NUM_INNER])
    {
        for(int i = 0; i < NUM_INNER; ++i)
            array_[i] = val.array_[i];
    }
};

```

```
}

Value &operator=(const Value &val)
{
    for(int i = 0; i < NUM_INNER; ++i)
        array_[i] = val.array_[i];
    return *this;
}
};
```

◦ ◦

## C ++

```
int i = 5;
int j = i; //Copied
j += 20;
std::cout << i; //Prints 5; i is unaffected by changes to j.
```

```
int i = 5;
int j = i; //Copied
j += 20;
std::cout << i; //Prints 5; i is unaffected by changes to j.
```

◦

## C ++

```
int i = 5;
int j = i; //Copied
j += 20;
std::cout << i; //Prints 5; i is unaffected by changes to j.
```

## C ++◦

<https://riptutorial.com/zh-TW/cplusplus/topic/1955/>

44:

## Examples

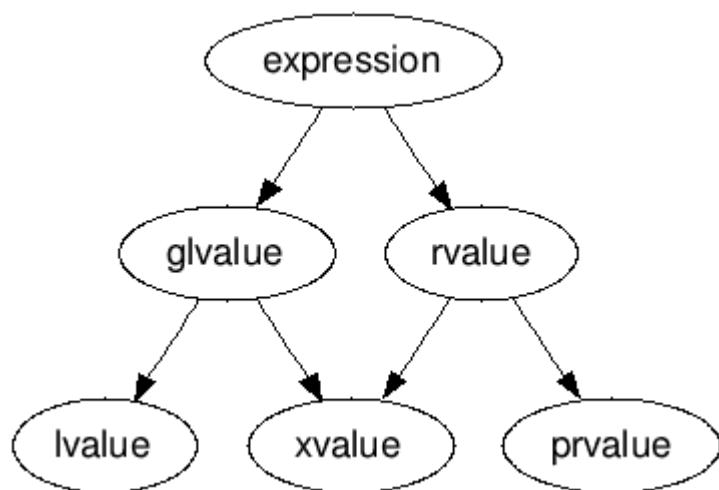
C ++。 C ++。

○ ○ ○ ○

०

C++3xvalueprvalue。 C++。

C++ glvaluervalue . .



## prvalue

**prvaluervalue.**

- `std::string("123")`。
  - 
  - - 1 true 0.5f'a'
  - `lambda`

& o

x

xvalueeXpiring。 xvalue“eXpiring”。

```
struct X { int n; };
extern X x;

4;           // prvalue: does not have an identity
x;           // lvalue
```

```

x.n;           // lvalue
std::move(x); // xvalue
std::forward<X&>(x); // lvalue
X{4};          // prvalue: does not have an identity
X{4}.n;        // xvalue: does have an identity and denotes resources
                // that can be reused

```

◦ ◦

```

struct X { ... };

X x;           // x is an lvalue
X* px = &x;   // px is an lvalue
*px = X{};    // *px is also an lvalue, X{} is a prvalue

X* foo_ptr(); // foo_ptr() is a prvalue
X& foo_ref(); // foo_ref() is an lvalue

```

4 'x' prvalues◦

## glvalue

glvalue“”。 xvaluesprvalues◦

### glvalue

```

struct X { int n; };
X foo();

X x;
x; // has a name, so it's a glvalue
std::move(x); // has a name (we're moving from "x"), so it's a glvalue
              // can be moved from, so it's an xvalue not an lvalue

foo(); // has no name, so is a prvalue, not a glvalue
X{};   // temporary has no name, so is a prvalue, not a glvalue
X{}.n; // HAS a name, so is a glvalue. can be moved from, so it's an xvalue

```

rvalue◦

rvalue<sub>T</sub> &&Texpr◦ rvalue;◦

rvalue xvalue prvalue◦

std::move◦ xvalue prvalue std::move identity xvalue◦

```

std::string str("init");           //1
std::string test1(str);           //2
std::string test2(std::move(str)); //3

str = std::string("new value");    //4
std::string &&str_ref = std::move(str); //5
std::string test3(str_ref);        //6

```

```
std::string std::string&& ""。 strrvalue。 const std::string& overload。  
3。 std::moveT&& T。 std::move(str) std::string&& 。 xvalue std::stringmove。 3 str。  
4 std::string。 std::string&& 。 std::string("new value")rvalueprvalue。 str。  
5 str_refrvalue str。 。  
str_ref std::stringrvalue str_refrvalue。 。 。 str_ref std::stringmove str_ref。 6 test3 str test3。  
std::move。
```

<https://riptutorial.com/zh-TW/cplusplus/topic/763/>

# 45:

- as-if◦

## Examples

/

◦ ◦

```
// source:  
  
int process(int value)  
{  
    return 2 * value;  
}  
  
int foo(int a)  
{  
    return process(a);  
}  
  
// program, after inlining:  
  
int foo(int a)  
{  
    return 2 * a; // the body of process() is copied into foo()  
}
```

◦

classclassstruct 1◦

class

```
#include <cassert>  
  
struct Base {}; // empty class  
  
struct Derived1 : Base {  
    int i;  
};  
  
int main() {  
    // the size of any object of empty class type is at least 1  
    assert(sizeof(Base) == 1);  
  
    // empty base optimization applies  
    assert(sizeof(Derived1) == sizeof(int));  
}
```

std::vector std::function std::shared\_ptr◦ vector begin endcapacity◦

cppreference

<https://riptutorial.com/zh-TW/cplusplus/topic/9767/>

# 46:

C++。

- 
- `constexpr`
- ◦

## Examples

- 

```
#include <iostream>

template<unsigned int n>
struct factorial
{
    enum
    {
        value = n * factorial<n - 1>::value
    };
};

template<>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{
    std::cout << factorial<7>::value << std::endl;      // prints "5040"
}
```

`factorial`◦ `::type::type::value`◦  
`factorial::value`◦

- `factorial<0>`◦ “”◦
- 

```
#include <iostream>

template<unsigned int n>
struct factorial
{
    enum
    {
        value = n * factorial<n - 1>::value
    };
};
```

```

    };
}

template<>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{
    std::cout << factorial<7>::value << std::endl;      // prints "5040"
}

```

◦ ◦

◦ g++256◦ g++ -ftemplate-depth-X◦

## C++ 11

### C++ 11 std::integral\_constant

```

#include <iostream>

template<unsigned int n>
struct factorial
{
    enum
    {
        value = n * factorial<n - 1>::value
    };
};

template<>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{
    std::cout << factorial<7>::value << std::endl;      // prints "5040"
}

```

constexpr◦

```

#include <iostream>

template<unsigned int n>
struct factorial
{
    enum
    {
        value = n * factorial<n - 1>::value
    };
};

```

```

template<>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{
    std::cout << factorial<7>::value << std::endl;      // prints "5040"
}

```

`factorial()` C++11 `constexpr`

C++14

C++14 `constexpr`

```

#include <iostream>

template<unsigned int n>
struct factorial
{
    enum
    {
        value = n * factorial<n - 1>::value
    };
};

template<>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{
    std::cout << factorial<7>::value << std::endl;      // prints "5040"
}

```

```

#include <iostream>

template<unsigned int n>
struct factorial
{
    enum
    {
        value = n * factorial<n - 1>::value
    };
};

template<>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{

```

```
    std::cout << factorial<7>::value << std::endl;      // prints "5040"
}
```

## C++ 17

### C++ 17fold

```
#include <iostream>

template<unsigned int n>
struct factorial
{
    enum
    {
        value = n * factorial<n - 1>::value
    };
};

template<>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{
    std::cout << factorial<7>::value << std::endl;      // prints "5040"
}
```

## ◦ C++ 17◦

## C++ 11

```
void print_all(std::ostream& os) {
    // base case
}

template <class T, class... Ts>
void print_all(std::ostream& os, T const& first, Ts const&... rest) {
    os << first;

    print_all(os, rest...);
}
```

◦

## C++ 11

```
void print_all(std::ostream& os) {
    // base case
}

template <class T, class... Ts>
void print_all(std::ostream& os, T const& first, Ts const&... rest) {
    os << first;

    print_all(os, rest...);
```

```
}
```

TC。

C++ 17

C++ 17。fold-expression

```
void print_all(std::ostream& os) {
    // base case
}

template <class T, class... Ts>
void print_all(std::ostream& os, T const& first, Ts const&... rest) {
    os << first;
    print_all(os, rest...);
}
```

if constexpr

```
void print_all(std::ostream& os) {
    // base case
}

template <class T, class... Ts>
void print_all(std::ostream& os, T const& first, Ts const&... rest) {
    os << first;
    print_all(os, rest...);
}
```

**std :: integer\_sequence**

C++ 14

```
template <class T, T... Ints>
class integer_sequence;

template <std::size_t... Ints>
using index_sequence = std::integer_sequence<std::size_t, Ints...>;
```

```
template <class T, T... Ints>
class integer_sequence;

template <std::size_t... Ints>
using index_sequence = std::integer_sequence<std::size_t, Ints...>;
```

C++ 14C++ 11。

---

std::tuple C++ 17 std::apply

```
template <class T, T... Ints>
class integer_sequence;

template <std::size_t... Ints>
using index_sequence = std::integer_sequence<std::size_t, Ints...>;
```

- `std::advance()`

```
namespace details {
    template <class RAIter, class Distance>
    void advance(RAIter& it, Distance n, std::random_access_iterator_tag) {
        it += n;
    }

    template <class BidirIter, class Distance>
    void advance(BidirIter& it, Distance n, std::bidirectional_iterator_tag) {
        if (n > 0) {
            while (n--) ++it;
        }
        else {
            while (n++) --it;
        }
    }

    template <class InputIter, class Distance>
    void advance(InputIter& it, Distance n, std::input_iterator_tag) {
        while (n--) {
            ++it;
        }
    }
}

template <class Iter, class Distance>
void advance(Iter& it, Distance n) {
    details::advance(it, n,
                    typename std::iterator_traits<Iter>::iterator_category{} );
}
```

```
details::advancestd::XY iterator tag. o details::advanceo
```

```
advanceiterator_traits<T>::iterator_category iterator_tagIter. iterator_category<Iter>::type  
details::advance. struct.
```

**SFINAE**<sub>enable\_if</sub>

**C++ 17** if *constexpr* advance.

- ### ◦ std::hash

```
#include <functional> // for std::hash
#include <type_traits> // for std::false_type and std::true_type
#include <utility> // for std::declval

template<class, class = void>
struct has_hash
    : std::false_type
```

```
{};

template<class T>
struct has_hash<T, decltype(std::hash<T>() (std::declval<T>()), void())>
    : std::true_type
{};


```

## C++ 17

### C++ 17 std::void\_t

```
#include <functional> // for std::hash
#include <type_traits> // for std::false_type and std::true_type
#include <utility> // for std::declval

template<class, class = void>
struct has_hash
    : std::false_type
{};

template<class T>
struct has_hash<T, decltype(std::hash<T>() (std::declval<T>()), void())>
    : std::true_type
{};


```

std::void\_t

```
#include <functional> // for std::hash
#include <type_traits> // for std::false_type and std::true_type
#include <utility> // for std::declval

template<class, class = void>
struct has_hash
    : std::false_type
{};

template<class T>
struct has_hash<T, decltype(std::hash<T>() (std::declval<T>()), void())>
    : std::true_type
{};


```

operator<

```
#include <functional> // for std::hash
#include <type_traits> // for std::false_type and std::true_type
#include <utility> // for std::declval

template<class, class = void>
struct has_hash
    : std::false_type
{};

template<class T>
struct has_hash<T, decltype(std::hash<T>() (std::declval<T>()), void())>
    : std::true_type
{};


```

```

Tstd::hashstd::unordered_map<T> std::map<T>

#include <functional> // for std::hash
#include <type_traits> // for std::false_type and std::true_type
#include <utility> // for std::declval

template<class, class = void>
struct has_hash
    : std::false_type
{};

template<class T>
struct has_hash<T, decltype(std::hash<T>() (std::declval<T>()), void())>
    : std::true_type
{};

```

## C++ 11

### C++ 11.

```

template <typename T>
constexpr T calculatePower(T value, unsigned power) {
    return power == 0 ? 1 : value * calculatePower(value, power-1);
}

```

constexpr

C++ 11 constexpr return.

- C++.

```

template <typename T>
constexpr T calculatePower(T value, unsigned power) {
    return power == 0 ? 1 : value * calculatePower(value, power-1);
}

```

## C++ 17

### fold

```

template <typename T>
constexpr T calculatePower(T value, unsigned power) {
    return power == 0 ? 1 : value * calculatePower(value, power-1);
}

```

## T

`std::enable_if` SFINAE

truefalsestd::true\_typestd::false\_type .

T is\_pointerstd::is\_pointer

```

template <typename T>
struct is_pointer_: std::false_type {};

template <typename T>
struct is_pointer_<T*>: std::true_type {};

template <typename T>
struct is_pointer: is_pointer_<typename std::remove_cv<T>::type> { }

```

1. is\_pointer\_ std::false\_type 。 std::false\_type“ false”。

2. T\*is\_pointer\_T。 std::true\_type 。

3. Tconstvolatile。

is\_pointer<T>

- Use ::value is\_pointer<int>::value - valuestd::true\_typestd::false\_typebool;
- is\_pointer<int>{} - std::is\_pointerstd::true\_typestd::false\_type constexprstd::true\_type  
std::false\_typestd::false\_typeconstexprbool 。

“”

```

template <typename T>
struct is_pointer_: std::false_type {};

template <typename T>
struct is_pointer_<T*>: std::true_type {};

template <typename T>
struct is_pointer: is_pointer_<typename std::remove_cv<T>::type> { }

```

## C ++ 17

### C ++ 17\_v

```

template <typename T>
struct is_pointer_: std::false_type {};

template <typename T>
struct is_pointer_<T*>: std::true_type {};

template <typename T>
struct is_pointer: is_pointer_<typename std::remove_cv<T>::type> { }

```

## IF-THEN-ELSE

### C ++ 11

<type\_traits>std::conditional

```

template<typename T>
struct ValueOrPointer
{

```

```
    typename std::conditional<(sizeof(T) > sizeof(void*)), T*, T>::type vop;  
};
```

TT:T<sup>o</sup> sizeof(ValueOrPointer) **<=** sizeof(void\*) 。

/

## C++ 11

min o min o

```
template <typename T1, typename T2>  
auto min(const T1 &a, const T2 &b)  
-> typename std::common_type<const T1&, const T2&>::type  
{  
    return a < b ? a : b;  
}  
  
template <typename T1, typename T2, typename ... Args>  
auto min(const T1 &a, const T2 &b, const Args& ... args)  
-> typename std::common_type<const T1&, const T2&, const Args& ...>::type  
{  
    return min(min(a, b), args...);  
}  
  
auto minimum = min(4, 5.8f, 3, 1.8, 3, 1.1, 9);
```

<https://riptutorial.com/zh-TW/cplusplus/topic/462/>

# 47:

- :: *opt new opt new-type-id new-initializer opt*
- :: *opt new opt type-id new-initializer opt*
- :: *opt cast-expression*
- :: *opt delete [] cast-expression*
- std :: unique\_ptr < type-id > var\_name new type-id opt ; // C ++ 11
- std :: shared\_ptr < type-id > var\_name new type-id opt ; // C ++ 11
- std :: shared\_ptr < type-id > var\_name = std :: make\_shared < type-id > opt ; // C ++ 11
- std :: unique\_ptr < type-id > var\_name = std :: make\_unique < type-id > opt ; // C ++ 14

::new delete new delete◦

new placement new ◦

new Foo auto C ++ 11 decltype(auto) C ++ 14◦

◦ ISO◦

delete[] *new-expression* *new-expression* [ ]◦

```
using IA = int[4];
int* pIA = new IA;
delete[] pIA; // right
// delete pIA; // wrong
```

## Examples

◦ ◦

```
int main() {
    int a = 0; //Stored on the stack
    return a;
}
```

“”◦

```
int main() {
    int a = 0; //Stored on the stack
    return a;
}
```

◦

```
int main() {
    int a = 0; //Stored on the stack
    return a;
}
```

\*\*\*

“”。

C++ 。

## Free Store。Free Store。

*newdelete*。

```
float *foo = nullptr;
{
    *foo = new float; // Allocates memory for a float
    float bar;           // Stack allocated
} // End lifetime of bar, while foo still alive

delete foo;           // Deletes the memory for the float at pF, invalidating the pointer
foo = nullptr;         // Setting the pointer to nullptr after delete is often considered good
practice
```

*newdelete*。。

```
float *foo = nullptr;
{
    *foo = new float; // Allocates memory for a float
    float bar;           // Stack allocated
} // End lifetime of bar, while foo still alive

delete foo;           // Deletes the memory for the float at pF, invalidating the pointer
foo = nullptr;         // Setting the pointer to nullptr after delete is often considered good
practice
```

*newdeletemallocfree*。 *newdeletemallocfree*。

```
float *foo = nullptr;
{
    *foo = new float; // Allocates memory for a float
    float bar;           // Stack allocated
} // End lifetime of bar, while foo still alive

delete foo;           // Deletes the memory for the float at pF, invalidating the pointer
foo = nullptr;         // Setting the pointer to nullptr after delete is often considered good
practice
```

C++ 11

C++ 11。

C++ 14

C++ 14 `std::make_unique` `STL` `std::make_unique` `std::make_shared` `naked newdelete`。

Free Store<sub>new</sub>。

Placement New `new'

```
int a4byteInteger;  
  
char *a4byteChar = new (&a4byteInteger) char[4];
```

a4byteChar a4byteInteger 'stack' 4.

◦ a4byteInteger 'delete a4byteChar'.

◦

```
int a4byteInteger;  
  
char *a4byteChar = new (&a4byteInteger) char[4];
```

a8byteChar a8byteDynamicInteger. delete a8byteDynamicInteger

## C ++

```
int a4byteInteger;  
  
char *a4byteChar = new (&a4byteInteger) char[4];
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2873/>

# 48:

inline◦ ◦ ◦

- `inline function_declaration`
- `inline function_definition`
- `class {function_definition};`

◦ ◦

inline◦ inline -◦ ◦ inline◦

C ++inline◦ ODR ;◦ inline◦

## C ++/'inline'

◦ ◦ ◦ ◦

## C ++/'inline'

inline◦

/

◦ ◦ ◦

- `'/inline'`
- 

## Examples

```
inline int add(int x, int y);
```

```
inline int add(int x, int y)
{
    return x + y;
}
```

```
// header (.hpp)
struct A
{
    void i_am_inlined()
    {
```

```
    }
};

struct B
{
    void i_am_NOT_inlined();
};

// source (.cpp)
void B::i_am_NOT_inlined()
{
}
```

```
inline int add(int x, int y)
{
    return x + y;
}

int main()
{
    int a = 1, b = 2;
    int c = add(a, b);
}
```

add

```
inline int add(int x, int y)
{
    return x + y;
}

int main()
{
    int a = 1, b = 2;
    int c = add(a, b);
}
```

◦ add ◦ - ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/7150/>

49:

○

## Examples

inline. . C ++ 17. cppFoo::num\_instancesC ++ 17Foo::num\_instancesFoo::num\_instancesint.

```
// warning: not thread-safe...
class Foo {
public:
    Foo() { ++num_instances; }
    ~Foo() { --num_instances; }
    inline static int num_instances = 0;
};
```

## constexpr

```
// warning: not thread-safe...
class Foo {
public:
    Foo() { ++num_instances; }
    ~Foo() { --num_instances; }
    inline static int num_instances = 0;
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/9265/>

## Examples

◦ ◦

```
enum myEnum
{
    enumName1,
    enumName2,
};
```

◦ myEnum ◦ ◦

◦ ::◦ enumName1 ◦

## C++ 11

◦ enumName1myEnum::enumName1 ◦

01◦ enumName10enumName21◦

;◦ + 1◦

```
enum myEnum
{
    enumName1,
    enumName2,
};
```

## switch

switch◦ switch◦

```
enum State {
    start,
    middle,
    end
};

...
switch(myState) {
    case start:
        ...
    case middle:
        ...
} // warning: enumeration value 'end' not handled in switch [-Wswitch]
```

◦

- enum

```
enum E {
    Begin,
    E1 = Begin,
    E2,
    // ..
    En,
    End
};

for (E e = E::Begin; e != E::End; ++e) {
    // Do job with e
}
```

## C++ 11

enum class operator ++

```
enum E {
    Begin,
    E1 = Begin,
    E2,
    // ..
    En,
    End
};

for (E e = E::Begin; e != E::End; ++e) {
    // Do job with e
}
```

- std::vector

```
enum E {
    Begin,
    E1 = Begin,
    E2,
    // ..
    En,
    End
};

for (E e = E::Begin; e != E::End; ++e) {
    // Do job with e
}
```

```
enum E {
    Begin,
    E1 = Begin,
    E2,
    // ..
    En,
    End
};

for (E e = E::Begin; e != E::End; ++e) {
```

```
// Do job with e  
}
```

## C++ 11

- std::initializer\_list

```
enum E {  
    Begin,  
    E1 = Begin,  
    E2,  
    // ..  
    En,  
    End  
};  
  
for (E e = E::Begin; e != E::End; ++e) {  
    // Do job with e  
}
```

```
enum E {  
    Begin,  
    E1 = Begin,  
    E2,  
    // ..  
    En,  
    End  
};  
  
for (E e = E::Begin; e != E::End; ++e) {  
    // Do job with e  
}
```

## Scoped enums

### C++ 11 ◦ enumname::membername ◦ enum class enum class ◦

```
enum class rainbow {  
    RED,  
    ORANGE,  
    YELLOW,  
    GREEN,  
    BLUE,  
    INDIGO,  
    VIOLET  
};
```

```
enum class rainbow {  
    RED,  
    ORANGE,  
    YELLOW,  
    GREEN,  
    BLUE,  
    INDIGO,  
    VIOLET
```

```
};
```

```
enum class ESint { int x = rainbow::RED };
```

## Scoped `int` `Tic-Tac-Toe`

```
enum class rainbow {
    RED,
    ORANGE,
    YELLOW,
    GREEN,
    BLUE,
    INDIGO,
    VIOLET
};
```

```
enum{
```

## C++ 11

```
...
enum class Status; // Forward declaration
Status doWork(); // Use the forward declaration
...
enum class Status { Invalid, Success, Fail };
Status doWork() // Full declaration required for implementation
{
    return Status::Success;
}
```

```
...
enum class Status; // Forward declaration
Status doWork(); // Use the forward declaration
...
enum class Status { Invalid, Success, Fail };
Status doWork() // Full declaration required for implementation
{
    return Status::Success;
}
```

## Blind fruit merchant

<https://riptutorial.com/zh-TW/cplusplus/topic/2796/>

# 51:

## Examples

### gccgprof

GNU gprof profiler [gprof](#)。

- 1.
- 2.
3. gprof

-pg◦

```
$ gcc -pg *.cpp -o app
```

```
$ gcc -pg *.cpp -o app
```

◦

app

```
$ gcc -pg *.cpp -o app
```

gmon.out◦

```
$ gcc -pg *.cpp -o app
```

◦

```
$ gcc -pg *.cpp -o app
```

◦

◦

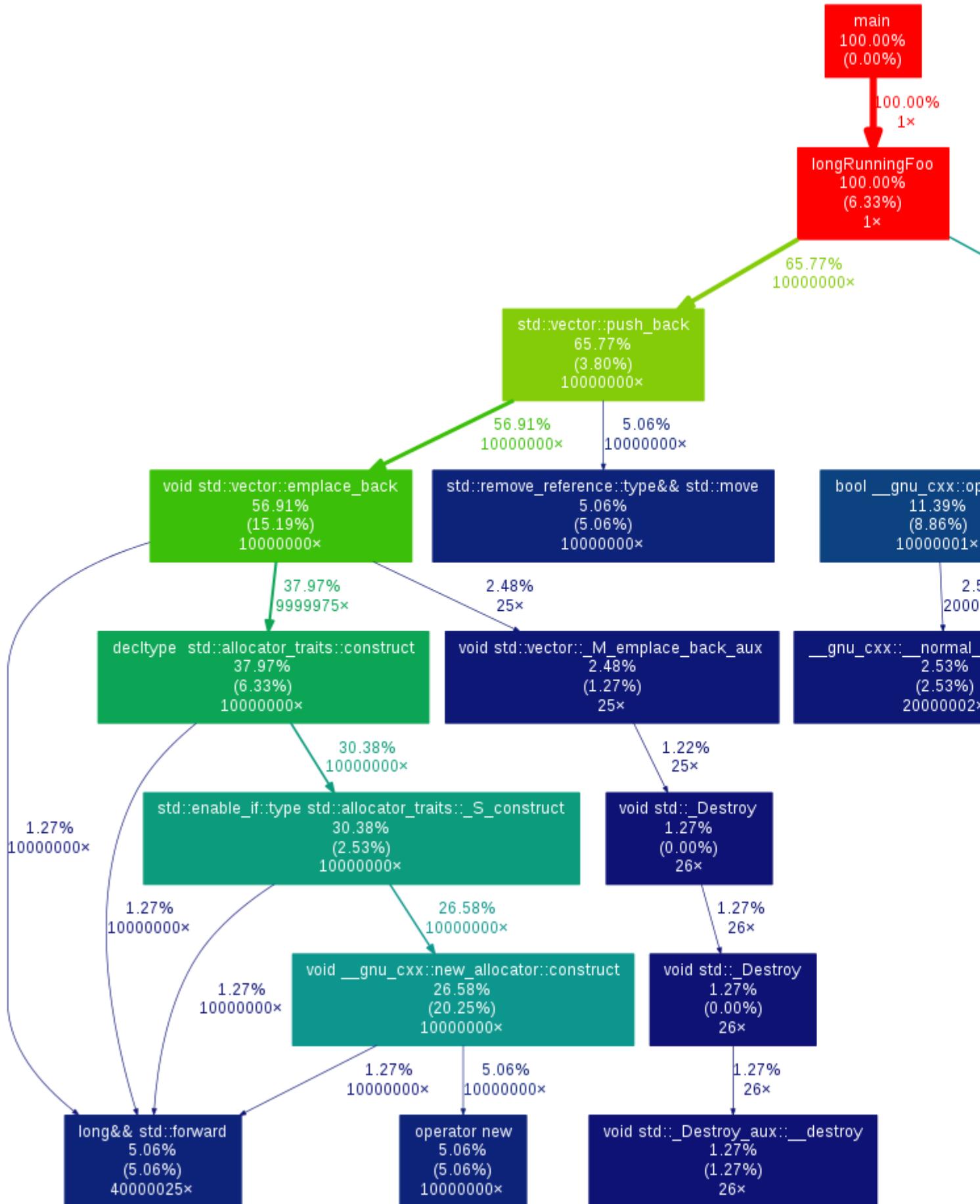
### gperf2dot

◦ ◦

[gperf2dot](#) Linux perfcallgrindoprofile◦ [gprof](#)

```
# compile with profiling flags
g++ *.cpp -pg
```

```
# run to generate profiling data
./main
# translate profiling data to text, create image
gprof ./main | gprof2dot -s | dot -Tpng -o output.png
```



gccGoogle Perf ToolsCPU

### 1. Google Perf

- 2.
3. libprofiler
4. pprof

```
# compile code
g++ -O3 -std=c++11 main.cpp -o main

# run with profiler
LD_PRELOAD=/usr/local/lib/libprofiler.so CPUPROFILE=main.prof CPUPROFILE_FREQUENCY=100000
./main
```

- CPUPROFILE
- CPUPROFILE\_FREQUENCY;

pprof°

```
# compile code
g++ -O3 -std=c++11 main.cpp -o main

# run with profiler
LD_PRELOAD=/usr/local/lib/libprofiler.so CPUPROFILE=main.prof CPUPROFILE_FREQUENCY=100000
./main
```

...pdf

```
# compile code
g++ -O3 -std=c++11 main.cpp -o main

# run with profiler
LD_PRELOAD=/usr/local/lib/libprofiler.so CPUPROFILE=main.prof CPUPROFILE_FREQUENCY=100000
./main
```

<https://riptutorial.com/zh-TW/cplusplus/topic/5347/>

# 52:

- ◦
- 

## Examples

- 
- 

◦ ◦ ODR ◦ ◦

```
template<typename T>
void f(T*) { }

template<typename T>
void f(T) { }
```

“T”“T \*”◦

```
template<typename T>
void f(T*) { }

template<typename T>
void f(T) { }
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4164/>

# 53:

◦

```
void foo(double, double);
void foo(long, long);

//Call foo with 2 ints
foo(1, 2); //Function call is ambiguous - int can be converted into a double/long at the same
time
```

## Examples

◦

std::string

```
void print(const std::string &str)
{
    std::cout << "This is a string: " << str << std::endl;
}
```

int.

```
void print(const std::string &str)
{
    std::cout << "This is a string: " << str << std::endl;
}
```

```
void print(const std::string &str)
{
    std::cout << "This is a string: " << str << std::endl;
}
```

print ◦ std::string int ◦

```
void print(const std::string &str)
{
    std::cout << "This is a string: " << str << std::endl;
}
```

```
void print(const std::string &str)
{
    std::cout << "This is a string: " << str << std::endl;
}
```

◦ ◦

```
void print(const std::string &str)
```

```
{  
    std::cout << "This is a string: " << str << std::endl;  
}
```

print

```
void print(const std::string &str)  
{  
    std::cout << "This is a string: " << str << std::endl;  
}
```

```
void print(const std::string &str)  
{  
    std::cout << "This is a string: " << str << std::endl;  
}
```

```
void print(const std::string &str)  
{  
    std::cout << "This is a string: " << str << std::endl;  
}
```

print(17)◦

◦

```
// WRONG CODE  
std::string getValue()  
{  
    return "hello";  
}  
  
int getValue()  
{  
    return 0;  
}  
  
int x = getValue();
```

getValue int ◦

## cv-qualifier

**CV**; const volatile const volatile◦ this**cv-qualifiers**◦ const volatile const volatile◦

**CV**◦ const const const const const◦ **CVCV**◦

print const

```
#include <iostream>  
  
class Integer  
{  
public:
```

```

    Integer(int i_): i{i_} {}

    void print()
    {
        std::cout << "int: " << i << std::endl;
    }

    void print() const
    {
        std::cout << "const int: " << i << std::endl;
    }

protected:
    int i;
};

int main()
{
    Integer i{5};
    const Integer &ic = i;

    i.print(); // prints "int: 5"
    ic.print(); // prints "const int: 5"
}

```

`const` `const` `const` `const` / `const` **cv-qualifiers**。

```

#include <iostream>

class Integer
{
public:
    Integer(int i_): i{i_} {}

    void print()
    {
        std::cout << "int: " << i << std::endl;
    }

    void print() const
    {
        std::cout << "const int: " << i << std::endl;
    }

protected:
    int i;
};

int main()
{
    Integer i{5};
    const Integer &ic = i;

    i.print(); // prints "int: 5"
    ic.print(); // prints "const int: 5"
}

```

`const` **mutator** `const`。

---

```
const. std::mutex mutable

#include <iostream>

class Integer
{
public:
    Integer(int i_): i{i_} {}

    void print()
    {
        std::cout << "int: " << i << std::endl;
    }

    void print() const
    {
        std::cout << "const int: " << i << std::endl;
    }

protected:
    int i;
};

int main()
{
    Integer i{5};
    const Integer &ic = i;

    i.print(); // prints "int: 5"
    ic.print(); // prints "const int: 5"
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/510/>

# 54:

- std ::<T>
- std :: atomic\_flag

std::atomic~~TriviallyCopyable~~ std::atomic\_flag。

## Examples

◦

```
#include <thread>
#include <iostream>

//function will add all values including and between 'a' and 'b' to 'result'
void add(int a, int b, int * result) {
    for (int i = a; i <= b; i++) {
        *result += i;
    }
}

int main() {
    //a primitive data type has no thread safety
    int shared = 0;

    //create a thread that may run parallel to the 'main' thread
    //the thread will run the function 'add' defined above with paramters a = 1, b = 100,
    result = &shared
    //analogous to 'add(1,100, &shared);'
    std::thread addingThread(add, 1, 100, &shared);

    //attempt to print the value of 'shared' to console
    //main will keep repeating this until the addingThread becomes joinable
    while (!addingThread.joinable()) {
        //this may cause undefined behavior or print a corrupted value
        //if the addingThread tries to write to 'shared' while the main thread is reading it
        std::cout << shared << std::endl;
    }

    //rejoin the thread at the end of execution for cleaning purposes
    addingThread.join();

    return 0;
}
```

◦

```
#include <thread>
#include <iostream>

//function will add all values including and between 'a' and 'b' to 'result'
```

```

void add(int a, int b, int * result) {
    for (int i = a; i <= b; i++) {
        *result += i;
    }
}

int main() {
    //a primitive data type has no thread safety
    int shared = 0;

    //create a thread that may run parallel to the 'main' thread
    //the thread will run the function 'add' defined above with paramters a = 1, b = 100,
    result = &shared
    //analogous to 'add(1,100, &shared);'
    std::thread addingThread(add, 1, 100, &shared);

    //attempt to print the value of 'shared' to console
    //main will keep repeating this until the addingThread becomes joinable
    while (!addingThread.joinable()) {
        //this may cause undefined behavior or print a corrupted value
        //if the addingThread tries to write to 'shared' while the main thread is reading it
        std::cout << shared << std::endl;
    }

    //rejoin the thread at the end of execution for cleaning purposes
    addingThread.join();

    return 0;
}

```

atomicstore() load() int。

<https://riptutorial.com/zh-TW/cplusplus/topic/3804/>

## Examples

“””<sub>T</sub>◦

- T◦
- T◦
- T◦ ◦
- TClassTemplate<TemplateArguments> ◦
- TUU U ◦ void (\*fptr) (A); f(fptr); void(A)◦
- T◦ void(A)A◦
- T◦ BA::\*p; void (A::\*pf) (B); f(p); f(pf); A B void(B)◦

◦ ◦ ◦

f◦

```
namespace A {
    struct Z { };
    namespace I { void g(Z); }
    using namespace I;

    struct X { struct Y { }; friend void f(Y) { } };
    void f(X p) { }
    void f(std::shared_ptr<X> p) { }
}

// example calls
f(A::X());
f(A::X::Y());
f(std::make_shared<A::X>());

g(A::Z()); // invalid: "using namespace I;" is ignored!
```

<https://riptutorial.com/zh-TW/cplusplus/topic/5163/>

## Examples

```
template<class ... Types> struct Tuple {};  
.  
.  
parameter_pack ...parameter_pack  
  
template<class T> // Base of recursion  
void variadic_printer(T last_argument) {  
    std::cout << last_argument;  
}  
  
template<class T, class ...Args>  
void variadic_printer(T first_argument, Args... other_arguments) {  
    std::cout << first_argument << "\n";  
    variadic_printer(other_arguments...); // Parameter pack expansion  
}  
  
variadic_printer(1, 2, 3, "hello");variadic_printer(1, 2, 3, "hello");  
  
template<class T> // Base of recursion  
void variadic_printer(T last_argument) {  
    std::cout << last_argument;  
}  
  
template<class T, class ...Args>  
void variadic_printer(T first_argument, Args... other_arguments) {  
    std::cout << first_argument << "\n";  
    variadic_printer(other_arguments...); // Parameter pack expansion  
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7668/>

## Examples

`const` ◦ & ◦

```
int i = 10;
int &refi = i;
```

`refi` ◦

```
int i = 10;
int &refi = i;
```

```
int i = 10;
int &refi = i;
```

◦

```
int i = 10;
int &refi = i;
```

`nullptr`

```
int i = 10;
int &refi = i;
```

## C ++

C ++ Reference Alias ◦ ◦

◦ ◦

```
int main() {
    int i = 10;
    int &j = i;

    cout << &i << endl;
    cout << &b << endl;
    return 0;
}
```

`cout` ◦

```
int main() {
    int i = 10;
    int &j = i;
```

```
    cout<<&i<<endl;
    cout<<&b<<endl;
    return 0;
}
```

cout◦

C++ References Alias◦

```
int main() {
    int i = 10;
    int &j = i;

    cout<<&i<<endl;
    cout<<&b<<endl;
    return 0;
}
```

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/1548/>

C++ C++ 17 STL invoke std::function operator lambdas STL

Stephan T. Lavavej [🔗](#)

## Examples

C C

```
typedef returnType(*name)(arguments); // All
using name = returnType(*)(arguments); // <= C++11
using name = std::add_pointer<returnType(arguments)>::type; // <= C++11
using name = std::add_pointer_t<returnType(arguments)>; // <= C++14
```

```
typedef returnType(*name)(arguments); // All
using name = returnType(*)(arguments); // <= C++11
using name = std::add_pointer<returnType(arguments)>::type; // <= C++11
using name = std::add_pointer_t<returnType(arguments)>; // <= C++14
```

- (\*lessThan)(v.front(), v.back()) // All
- std::invoke(lessThan, v.front(), v.back()) // <= C++17

## operatorFunctors

operator() C++ 11 Lambdas

```
struct Person {
    std::string name;
    unsigned int age;
};

// Functor which find a person by name
struct FindPersonByName {
    FindPersonByName(const std::string &name) : _name(name) {}

    // Overloaded method which will get called
    bool operator()(const Person &person) const {
        return person.name == _name;
    }
private:
    std::string _name;
};

std::vector<Person> v; // Assume this contains data
std::vector<Person>::iterator iFind =
    std::find_if(v.begin(), v.end(), FindPersonByName("Foobar"));
// ...
```

**typedef** std::find\_if

```
struct Person {
    std::string name;
    unsigned int age;
};

// Functor which find a person by name
struct FindPersonByName {
    FindPersonByName(const std::string &name) : _name(name) {}

    // Overloaded method which will get called
    bool operator()(const Person &person) const {
        return person.name == _name;
    }
private:
    std::string _name;
};

std::vector<Person> v; // Assume this contains data
std::vector<Person>::iterator iFind =
    std::find_if(v.begin(), v.end(), FindPersonByName("Foobar"));
// ...
```

C++ 17 invoke `std::invoke(predicate, *i)`。

<https://riptutorial.com/zh-TW/cplusplus/topic/6073/>

## Examples

`mutable const`

◦

`std::unique_lock const`

◦

```
class pi_calculator {
public:
    double get_pi() const {
        if (pi_calculated) {
            return pi;
        } else {
            double new_pi = 0;
            for (int i = 0; i < 1000000000; ++i) {
                // some calculation to refine new_pi
            }
            // note: if pi and pi_calculated were not mutable, we would get an error from a
compiler
            // because in a const method we can not change a non-mutable field
            pi = new_pi;
            pi_calculated = true;
            return pi;
        }
    }
private:
    mutable bool pi_calculated = false;
    mutable double pi = 0;
};
```

## Lambdas

`lambda operator() const` ◦ `lambda const` ◦ `lambda mutable operator() const`

```
int a = 0;

auto bad_counter = [a] {
    return a++; // error: operator() is const
                // cannot modify members
};

auto good_counter = [a]() mutable {
    return a++; // OK
}

good_counter(); // 0
good_counter(); // 1
good_counter(); // 2
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2705/>

# 60:

◦

- *opt { declaration-seq }*
- *opt { declaration-seq } / \*C ++ 11 \* /*
- *inline opt namespace attribute-specifier-seq identifier opt { declaration-seq } / \*C ++ 17 \* /*
- *enclosing-namespace-specifier :: identifier { declaration-seq } / \*C ++ 17 \* /*
- *namespace identifier = qualified-namespace-specifier ;*
- *using namespace nested-name-specifier opt namespace-name ;*
- *attribute-specifier-seq using namespace nested-name-specifier opt namespace-name ; / \*C ++ 11 \* /*

namespace

1. ◦ ◦

2. ◦

3. using *using* ◦ *using-directive*◦

using namespace std;◦ namespace std

```
//Really bad!
using namespace std;

//Calculates p^e and outputs it to std::cout
void pow(double p, double e) { /*...*/ }

//Calls pow
pow(5.0, 2.0); //Error! There is already a pow function in namespace std with the same
signature,
                //so the call is ambiguous
```

## Examples

C ++C ++◦ ◦ <namespace>::<entity> ◦

```
namespace Example
{
    const int test = 5;

    const int test2 = test + 12; //Works within `Example` namespace
}

const int test3 = test + 3; //Fails; `test` not found outside of namespace.

const int test3 = Example::test + 3; //Works; fully qualified name used.
```

◦ ◦ ◦ ◦ C ++◦

```
namespace Example
{
    const int test = 5;

    const int test2 = test + 12; //Works within `Example` namespace
}

const int test3 = test + 3; //Fails; `test` not found outside of namespace.

const int test3 = Example::test + 3; //Works; fully qualified name used.
```

:::

```
namespace Example
{
    const int test = 5;

    const int test2 = test + 12; //Works within `Example` namespace
}

const int test3 = test + 3; //Fails; `test` not found outside of namespace.

const int test3 = Example::test + 3; //Works; fully qualified name used.
```

```
//Creates namespace foo
namespace Foo
{
    //Declares function bar in namespace foo
    void bar() {}
}
```

bar ::

```
//Creates namespace foo
namespace Foo
{
    //Declares function bar in namespace foo
    void bar() {}
}
```

```
//Creates namespace foo
namespace Foo
{
    //Declares function bar in namespace foo
    void bar() {}
}
```

## C++ 17

```
//Creates namespace foo
namespace Foo
{
    //Declares function bar in namespace foo
    void bar() {}
}
```

```
namespace°
```

```
namespace Foo
{
    void bar() {}
}

//some other stuff

namespace Foo
{
    void bar2() {}
}
```

“” ° “””

```
FooFoo:: using namespace Foo; Foo°
```

```
namespace Foo
{
    void bar() {}
    void baz() {}
}

//Have to use Foo::bar()
Foo::bar();

//Import Foo
using namespace Foo;
bar(); //OK
baz(); //OK
```

```
namespace Foo
{
    void bar() {}
    void baz() {}
}

//Have to use Foo::bar()
Foo::bar();

//Import Foo
using namespace Foo;
bar(); //OK
baz(); //OK
```

using namespace° ° “using ”°

```
namespace Foo
{
    void bar() {}
    void baz() {}
}

//Have to use Foo::bar()
Foo::bar();
```

```
//Import Foo
using namespace Foo;
bar(); //OK
baz(); //OK
```

### *using-directive*◦

#### ◦ “Argument Dependent Lookup”ADL

```
namespace Test
{
    int call(int i);

    class SomeClass {...};

    int call_too(const SomeClass &data);
}

call(5); //Fails. Not a qualified function name.

Test::SomeClass data;

call_too(data); //Succeeds
```

callTest◦ call\_tooSomeClassTestADL◦

## ADL

### ADL◦

```
namespace Test
{
    int call(int i);

    class SomeClass {...};

    int call_too(const SomeClass &data);
}

call(5); //Fails. Not a qualified function name.

Test::SomeClass data;

call_too(data); //Succeeds
```

## C++ 11

inline namespace inline namespace

```
namespace Outer
{
    inline namespace Inner
    {
        void foo();
    }
}
```

```
    }
}
```

```
namespace Outer
{
    inline namespace Inner
    {
        void foo();
    }
}
```

Outer::Inner::Outer::

```
namespace Outer
{
    inline namespace Inner
    {
        void foo();
    }
}
```

using namespace Inner;

```
namespace Outer
{
    inline namespace Inner
    {
        void foo();
    }
}
```

- Outer::foo

```
namespace Outer
{
    inline namespace Inner
    {
        void foo();
    }
}
```

- using namespaceOuter::foo

```
namespace Outer
{
    inline namespace Inner
    {
        void foo();
    }
}
```

inline

```
namespace Outer
{
```

```
inline namespace Inner
{
    void foo();
}
```

```
namespace Outer
{
    inline namespace Inner
    {
        void foo();
    }
}
```

/

◦

```
namespace {
    int foo = 42;
}
```

foo◦

◦ const◦

```
namespace {
    int foo = 42;
}
```

## C++17

```
namespace a {
    namespace b {
        template<class T>
        struct qualifies : std::false_type {};
    }
}

namespace other {
    struct bob {};
}

namespace a::b {
    template<>
    struct qualifies<::other::bob> : std::true_type {};
}
```

namespace a::b C++17 ab namespace a::b ◦

◦

```
namespace boost
```

```

{
    namespace multiprecision
    {
        class Number ...
    }
}

namespace Name1 = boost::multiprecision;

// Both Type declarations are equivalent
boost::multiprecision::Number X // Writing the full namespace path, longer
Name1::Number Y // using the name alias, shorter

```

```

namespace boost
{
    namespace multiprecision
    {
        class Number ...
    }
}

using namespace boost;

// Both Namespace are equivalent
namespace Name1 = boost::multiprecision;
namespace Name2 = multiprecision;

```

```

namespace boost
{
    namespace multiprecision
    {
        class Number ...
    }
}

using namespace boost;

// Both Namespace are equivalent
namespace Name1 = boost::multiprecision;
namespace Name2 = multiprecision;

```

```

namespace =namespace ° ° AReallyLongName::AnotherReallyLongNamequxNN::°

namespace AReallyLongName {
    namespace AnotherReallyLongName {
        int foo();
        int bar();
        void baz(int x, int y);
    }
}
void qux() {
    namespace N = AReallyLongName::AnotherReallyLongName;
    N::baz(N::foo(), N::bar());
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/495/>

**Singleton** . .

- Singleton◦

- 

- ◦

- Singleton◦

[Robert Nystrom](#)

## Examples

Q & A <http://stackoverflow.com/a/1008289/3807729>

### c ++Singleton

- 

```
class S
{
public:
    static S& getInstance()
    {
        static S     instance; // Guaranteed to be destroyed.
                             // Instantiated on first use.
        return instance;
    }
private:
    S() {};           // Constructor? (the {} brackets) are needed here.

    // C++ 03
    // ======
    // Dont forget to declare these two. You want to make sure they
    // are unacceptable otherwise you may accidentally get copies of
    // your singleton appearing.
    S(S const&);          // Don't Implement
    void operator=(S const&); // Don't implement

    // C++ 11
    // =====
    // We can use the better technique of deleting the methods
    // we don't want.
public:
    S(S const&)            = delete;
    void operator=(S const&) = delete;

    // Note: Scott Meyers mentions in his Effective Modern
    //        C++ book, that deleted functions should generally
```

```
//      be public as it results in better error messages
//      due to the compilers behavior to check accessibility
//      before deleted status
};
```

:(  
**Singleton**

**C ++**

**C ++**

**SingletonGetInstance**

**C ++**

**C ++**

```
class API
{
public:
    static API& instance();

    virtual ~API() {}

    virtual const char* func1() = 0;
    virtual void func2() = 0;

protected:
    API() {}
    API(const API&) = delete;
    API& operator=(const API&) = delete;
};

class WindowsAPI : public API
{
public:
    virtual const char* func1() override { /* Windows code */ }
    virtual void func2() override { /* Windows code */ }
};

class LinuxAPI : public API
{
public:
    virtual const char* func1() override { /* Linux code */ }
    virtual void func2() override { /* Linux code */ }
};

API& API::instance() {
#if PLATFORM == WIN32
    static WindowsAPI instance;
#elif PLATFORM == LINUX
    static LinuxAPI instance;
```

```
#endif  
    return instance;  
}
```

API。 API。

# Singeton

C ++ 11

C++11。。

```
class Foo
{
public:
    static Foo& instance()
    {
        static Foo inst;
        return inst;
    }
private:
    Foo() {}
    Foo(const Foo&) = delete;
    Foo& operator =(const Foo&) = delete;
};
```

1

9

```
std::shared_ptr
```

```
class Singleton
{
public:
    Singleton(Singleton const&) = delete;
    Singleton& operator=(Singleton const&) = delete;

    static std::shared_ptr<Singleton> instance()
    {
        static std::shared_ptr<Singleton> s{new Singleton};
        return s;
    }

private:
    Singleton() {}
};


```

<https://riptutorial.com/zh-TW/cplusplus/topic/2713/>

## Examples

### INT

“”-32767+32767◦

```
int x = 2;
int y = 3;
int z = x + y;

unsigned short longlong long qv◦
truefalse ◦
```

```
bool is_even(int x) {
    return x%2 == 0;
}
const bool b = is_even(47); // false
```

“”◦ char-127+1270255◦

```
const char zero = '0';
const char one = zero + 1;
const char newline = '\n';
std::cout << one << newline; // prints 1 followed by a newline
```

### char16\_t

#### C++ 11

uint\_least16\_t UTF-16◦

```
const char16_t message[] = u"Hello\World\n";           // Chinese for "hello, world\n"
std::cout << sizeof(message)/sizeof(char16_t) << "\n"; // prints 7
```

### char32\_t

#### C++ 11

uint\_least32\_t UTF-32◦

```
const char32_t full_house[] = U"\u2000\u2000";           // non-BMP characters
std::cout << sizeof(full_house)/sizeof(char32_t) << "\n"; // prints 6
```

## ◦ C ++◦

```
float area(float radius) {  
    const float pi = 3.14159f;  
    return pi*radius*radius;  
}
```

◦ float ◦ longlong doubledouble◦

```
double area(double radius) {
    const double pi = 3.141592653589793;
    return pi*radius*radius;
}
```

`int -2147483647 + 2147483647 - 2 ^ 31-1 + 2 ^ 31-1。 long int。`

```
const long approx_seconds_per_year = 60L*60L*24L*365L;
```

`long double`。

```
const long approx_seconds_per_year = 60L*60L*24L*365L;
```

C ++ 11

`longlong long longlong -9223372036854775807+9223372036854775807 - ^ 2 - 63 - 1+2 ^ 63 - 1.`

```
const long approx_seconds_per_year = 60L*60L*24L*365L;
```

char-32767+32767。 short int。

```
// (during the last year)
short hours_worked(short days_worked) {
    return 8*days_worked;
}
```

:void voidvoid ° °

void int main() int main(void) { CC++ }

```
void* "void":: void*CAPIasort pthread create::
```

void:

```
static cast<void>(std::printf("Hello, %s!\n", name)); // discard return value
```

9

wchar\_t

- wchar\_t **UTF-16**

## ASCII 255char ◦

```
const wchar_t message_ahmaric[] = L"አላም አዴልና\n"; //Ahmaric for "hello, world\n"
const wchar_t message_chinese[] = L"你好世界\n"; // Chinese for "hello, world\n"
const wchar_t message_hebrew[] = L"שלום עולם\n"; //Hebrew for "hello, world\n"
const wchar_t message_russian[] = L"Привет мир\n"; //Russian for "hello, world\n"
const wchar_t message_tamil[] = L"ஓமால்டீ உலகம்\n"; //Tamil for "hello, world\n"
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7839/>

**63:**

## Examples

```
◦
```

```
class Shape {  
public:  
    virtual ~Shape() = default;  
    virtual double get_surface() const = 0;  
    virtual void describe_object() const { std::cout << "this is a shape"; }  
  
    double get_doubled_surface() const { return 2 * get_surface(); }  
};
```



```
class Shape {  
public:  
    virtual ~Shape() = default;  
    virtual double get_surface() const = 0;  
    virtual void describe_object() const { std::cout << "this is a shape" << std::endl; }  
  
    double get_doubled_surface() const { return 2 * get_surface(); }  
};
```

- ◦ ◦ virtual ◦ override ◦
  - ◦
  - ◦ ◦

```
class Shape {  
public:  
    virtual ~Shape() = default;  
    virtual double get_surface() const = 0;  
    virtual void describe_object() const { std::cout << "this is a shape" << std::endl; }  
  
    double get_doubled_surface() const { return 2 * get_surface(); }  
};
```

```
Shape *ps; // see example on defining a polymorphic class
```

```
ps = get_a_new_random_shape(); // if you don't have such a function yet, you
                             // could just write ps = new Square(0.0,0.0, 5);
```

ShapeSquareCircle °

```
Shape *ps;                      // see example on defining a polymorphic class
ps = get_a_new_random_shape(); // if you don't have such a function yet, you
                             // could just write ps = new Square(0.0,0.0, 5);
```

°

° °

° °

```
Shape *ps;                      // see example on defining a polymorphic class
ps = get_a_new_random_shape(); // if you don't have such a function yet, you
                             // could just write ps = new Square(0.0,0.0, 5);
```

get\_diameter() ° Shape

```
Shape *ps;                      // see example on defining a polymorphic class
ps = get_a_new_random_shape(); // if you don't have such a function yet, you
                             // could just write ps = new Square(0.0,0.0, 5);
```

psstatic\_cast

```
Shape *ps;                      // see example on defining a polymorphic class
ps = get_a_new_random_shape(); // if you don't have such a function yet, you
                             // could just write ps = new Square(0.0,0.0, 5);
```

° psCircle °

dynamic\_cast °

```
Shape *ps;                      // see example on defining a polymorphic class
ps = get_a_new_random_shape(); // if you don't have such a function yet, you
                             // could just write ps = new Square(0.0,0.0, 5);
```

dynamic\_cast ° °

/virtualprotected ° vtable ° /°

```
struct VirtualDestructor {
    virtual ~VirtualDestructor() = default;
};

struct VirtualDerived : VirtualDestructor {};

struct ProtectedDestructor {
    protected:
```

```
~ProtectedDestructor() = default;
};

struct ProtectedDerived : ProtectedDestructor {
    ~ProtectedDerived() = default;
};

// ...

VirtualDestructor* vd = new VirtualDerived;
delete vd; // Looks up VirtualDestructor::~VirtualDestructor() in vtable, sees it's
           // VirtualDerived::~VirtualDerived(), calls that.

ProtectedDestructor* pd = new ProtectedDerived;
delete pd; // Error: ProtectedDestructor::~ProtectedDestructor() is protected.
delete static_cast<ProtectedDerived*>(pd); // Good.
```

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/1717/>

# 64: CRTP

- CRTPC ++ ◦

## Examples

### CRTP

CRTP◦◦ static\_castthis◦◦

CRTP◦◦

C ++ 14

begin()end()◦◦ CRTPbegin()end()

```
#include <iterator>
template <typename Sub>
class Container {
private:
    // self() yields a reference to the derived type
    Sub& self() { return *static_cast<Sub*>(this); }
    Sub const& self() const { return *static_cast<Sub const*>(this); }

public:
    decltype(auto) front() {
        return *self().begin();
    }

    decltype(auto) back() {
        return *std::prev(self().end());
    }

    decltype(auto) size() const {
        return std::distance(self().begin(), self().end());
    }

    decltype(auto) operator[](std::size_t i) {
        return *std::next(self().begin(), i);
    }
};
```

begin()end()front() back() size()operator[]◦◦

```
#include <iterator>
template <typename Sub>
class Container {
private:
    // self() yields a reference to the derived type
    Sub& self() { return *static_cast<Sub*>(this); }
    Sub const& self() const { return *static_cast<Sub const*>(this); }

public:
```

```

 decltype(auto) front() {
    return *self().begin();
}

 decltype(auto) back() {
    return *std::prev(self().end());
}

 decltype(auto) size() const {
    return std::distance(self().begin(), self().end());
}

 decltype(auto) operator[](std::size_t i) {
    return *std::next(self().begin(), i);
}
};

```

## DynArray<sup>CRTP</sup>

```

#include <iterator>
template <typename Sub>
class Container {
private:
    // self() yields a reference to the derived type
    Sub& self() { return *static_cast<Sub*>(this); }
    Sub const& self() const { return *static_cast<Sub const*>(this); }

public:
    decltype(auto) front() {
        return *self().begin();
    }

    decltype(auto) back() {
        return *std::prev(self().end());
    }

    decltype(auto) size() const {
        return std::distance(self().begin(), self().end());
    }

    decltype(auto) operator[](std::size_t i) {
        return *std::next(self().begin(), i);
    }
};

```

```

#include <iterator>
template <typename Sub>
class Container {
private:
    // self() yields a reference to the derived type
    Sub& self() { return *static_cast<Sub*>(this); }
    Sub const& self() const { return *static_cast<Sub const*>(this); }

public:
    decltype(auto) front() {
        return *self().begin();
    }

    decltype(auto) back() {

```

```

        return *std::prev(self().end());
    }

    decltype(auto) size() const {
        return std::distance(self().begin(), self().end());
    }

    decltype(auto) operator[](std::size_t i) {
        return *std::next(self().begin(), i);
    }
};

```

`begin() Container<DynArray<int>>Container<DynArray<int>>;`

`DynArray`

```

#include <iterator>
template <typename Sub>
class Container {
private:
    // self() yields a reference to the derived type
    Sub& self() { return *static_cast<Sub*>(this); }
    Sub const& self() const { return *static_cast<Sub const*>(this); }

public:
    decltype(auto) front() {
        return *self().begin();
    }

    decltype(auto) back() {
        return *std::prev(self().end());
    }

    decltype(auto) size() const {
        return std::distance(self().begin(), self().end());
    }

    decltype(auto) operator[](std::size_t i) {
        return *std::next(self().begin(), i);
    }
};

```

## CRTP

### CRTP

```

struct IShape
{
    virtual ~IShape() = default;

    virtual void accept(IShapeVisitor&) const = 0;
};

struct Circle : IShape
{
    // ...
    // Each shape has to implement this method the same way
    void accept(IShapeVisitor& visitor) const override { visitor.visit(*this); }

```

```

    // ...
};

struct Square : IShape
{
    // ...
    // Each shape has to implement this method the same way
    void accept(IShapeVisitor& visitor) const override { visitor.visit(*this); }
    // ...
};


```

IShapeIShape.o .

```

struct IShape
{
    virtual ~IShape() = default;

    virtual void accept(IShapeVisitor&) const = 0;
};

struct Circle : IShape
{
    // ...
    // Each shape has to implement this method the same way
    void accept(IShapeVisitor& visitor) const override { visitor.visit(*this); }
    // ...
};

struct Square : IShape
{
    // ...
    // Each shape has to implement this method the same way
    void accept(IShapeVisitor& visitor) const override { visitor.visit(*this); }
    // ...
};


```

```

struct IShape
{
    virtual ~IShape() = default;

    virtual void accept(IShapeVisitor&) const = 0;
};

struct Circle : IShape
{
    // ...
    // Each shape has to implement this method the same way
    void accept(IShapeVisitor& visitor) const override { visitor.visit(*this); }
    // ...
};

struct Square : IShape
{
    // ...
    // Each shape has to implement this method the same way
    void accept(IShapeVisitor& visitor) const override { visitor.visit(*this); }
    // ...
};


```

◦

CRTP <https://riptutorial.com/zh-TW/cplusplus/topic/9269/-crtpprime>

# 65:

◦ 42x◦

C ++ 11◦

## Examples

bool◦

```
bool ok = true;
if (!f()) {
    ok = false;
    goto end;
}
```

bool◦

```
bool ok = true;
if (!f()) {
    ok = false;
    goto end;
}
```

## nullptr

C ++ 11

◦ ◦

```
Widget* p = new Widget();
delete p;
p = nullptr; // set the pointer to null after deletion
```

nullptr◦ nullptrstd::nullptr\_t◦

```
Widget* p = new Widget();
delete p;
p = nullptr; // set the pointer to null after deletion
```

this◦ this◦

```
struct S {
    int x;
    S& operator=(const S& other) {
        x = other.x;
        // return a reference to the object being assigned to
        return *this;
    }
}
```

```
};
```

this**CV**-X::fconst thisfconst X\* thisconst。 thisvolatile。

## C++ 11

this。

```
struct S {
    int x;
    S& operator=(const S& other) {
        x = other.x;
        // return a reference to the object being assigned to
        return *this;
    }
};
```

this。

•

1,2,3,4,5,6,7,8,90,1,2,3,4,5,6 7,8,9

int d = 42;

•

00,1,2,3,4,5,6,7

int o = 052

•

0x0X0,1,2,3,4,5,6,7,8,9aAbBcC dDeEfF

int x = 0x2a; int X = 0X2A;

- binary-literalC++ 14

0b0B0,1

int b = 0b101010; // C++14

- unsigned-suffixuU

unsigned int u\_1 = 42u;

- long-suffixlLlong-long-suffixllLC++ 11

```
unsigned long long l1 = 18446744073709550592ull; // C++11
unsigned long long l2 = 18'446'744'073'709'550'592llu; // C++14
unsigned long long l3 = 1844'6744'0737'0955'0592uLL; // C++14
unsigned long long l4 = 184467'440737'0'95505'92LLU; // C++14
```

0xDeAdBaBeU0XdeadBABELong-long-suffixlliLLILLI

◦ -1◦

C99CC ++long intunsigned long int◦

#if#elifstd :: intmax\_tstd :: uintmax\_t◦

<https://riptutorial.com/zh-TW/cplusplus/topic/7836/>

# 66:

◦ ◦

auto C ++ 11 register C ++ 17 static `thread_local` C ++ 11 `externmutable`。

*decl-specifier-seq*`thread_localstaticextern`。

◦ ◦ ◦

## Examples

◦ `const const` ◦

```
class C {
    int x;
    mutable int times_accessed;
public:
    C(): x(0), times_accessed(0) {
    }
    int get_x() const {
        ++times_accessed; // ok: const member function can modify mutable data member
        return x;
    }
    void set_x(int x) {
        ++times_accessed;
        this->x = x;
    }
};
```

C ++ 11

C ++ 11`mutable`◦ `lambda``lambda``const`◦ `lambda``copy`◦ `mutable lambdas`◦

```
class C {
    int x;
    mutable int times_accessed;
public:
    C(): x(0), times_accessed(0) {
    }
    int get_x() const {
        ++times_accessed; // ok: const member function can modify mutable data member
        return x;
    }
    void set_x(int x) {
        ++times_accessed;
        this->x = x;
    }
};
```

`lambda``mutable`◦

C ++ 17

◦ “”CPU◦ C ++ 11◦

```
register int i = 0;
while (i < 100) {
    f(i);
    int g = i*i;
    i += h(i, g);
}
```

register◦ CC ++register◦ register◦

C ++ 17

register◦ register◦

static◦

1.◦

```
// internal function; can't be linked to
static double semiperimeter(double a, double b, double c) {
    return (a + b + c)/2.0;
}
// exported to client
double area(double a, double b, double c) {
    const double s = semiperimeter(a, b, c);
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

2. thread\_local◦ ◦ ◦

```
// internal function; can't be linked to
static double semiperimeter(double a, double b, double c) {
    return (a + b + c)/2.0;
}
// exported to client
double area(double a, double b, double c) {
    const double s = semiperimeter(a, b, c);
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

3.◦

```
// internal function; can't be linked to
static double semiperimeter(double a, double b, double c) {
    return (a + b + c)/2.0;
}
// exported to client
double area(double a, double b, double c) {
    const double s = semiperimeter(a, b, c);
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

23 static◦

## C++ 03

◦ ◦

```
void f() {
    auto int x; // equivalent to: int x;
    auto y;     // illegal in C++; legal in C89
}
auto int z; // illegal: namespace-scope variable cannot be automatic
```

## C++ 11 auto ◦

### EXTERN

extern

1. ◦ ◦

```
// global scope
int x;           // definition; x will be default-initialized
extern int y;    // declaration; y is defined elsewhere, most likely another TU
extern int z = 42; // definition; "extern" has no effect here (compiler may warn)
```

2. constconstexpr◦

```
// global scope
int x;           // definition; x will be default-initialized
extern int y;    // declaration; y is defined elsewhere, most likely another TU
extern int z = 42; // definition; "extern" has no effect here (compiler may warn)
```

3. ◦ ◦

```
// global scope
int x;           // definition; x will be default-initialized
extern int y;    // declaration; y is defined elsewhere, most likely another TU
extern int z = 42; // definition; "extern" has no effect here (compiler may warn)
```

extern ◦ ◦

```
// global scope
int x;           // definition; x will be default-initialized
extern int y;    // declaration; y is defined elsewhere, most likely another TU
extern int z = 42; // definition; "extern" has no effect here (compiler may warn)
```

fexterngextern ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/9225/>

# 67:

ref. .

```
template<class T>
void f(T&& x) // x is a forwarding reference, because T is deduced from a call to f()
{
    g(std::forward<T>(x)); // g() will receive an lvalue or an rvalue, depending on x
}
```

T

```
template<class T>
void f(T&& x) // x is a forwarding reference, because T is deduced from a call to f()
{
    g(std::forward<T>(x)); // g() will receive an lvalue or an rvalue, depending on x
}
```

## C++ 17

C++ 17. “a”

```
template<class T>
void f(T&& x) // x is a forwarding reference, because T is deduced from a call to f()
{
    g(std::forward<T>(x)); // g() will receive an lvalue or an rvalue, depending on x
}
```

## Examples

◦ make\_unique Tunique\_ptr<T> ◦

◦

```
template<class T, class... A>
unique_ptr<T> make_unique(A&&... args)
{
    return unique_ptr<T>(new T(std::forward<A>(args)...));
}
```

...◦ std::forwardTo ref.

```
template<class T, class... A>
unique_ptr<T> make_unique(A&&... args)
{
    return unique_ptr<T>(new T(std::forward<A>(args)...));
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1750/>

## Examples

### TCP

Beej® TCP“Hello World” . . .

◦ -

```
#include <cstring>      // sizeof()
#include <iostream>
#include <string>

// headers for socket(), getaddrinfo() and friends
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>

#include <unistd.h>      // close()

int main(int argc, char *argv[])
{
    // Let's check if port number is supplied or not..
    if (argc != 2) {
        std::cerr << "Run program as 'program <port>'\n";
        return -1;
    }

    auto &portNum = argv[1];
    const unsigned int backLog = 8;    // number of connections allowed on the incoming queue

    addrinfo hints, *res, *p;      // we need 2 pointers, res to hold and p to iterate over
    memset(&hints, 0, sizeof(hints));

    // for more explanation, man socket
    hints.ai_family     = AF_UNSPEC;      // don't specify which IP version to use yet
    hints.ai_socktype   = SOCK_STREAM;    // SOCK_STREAM refers to TCP, SOCK_DGRAM will be?
    hints.ai_flags       = AI_PASSIVE;

    // man getaddrinfo
    int gAddRes = getaddrinfo(NULL, portNum, &hints, &res);
    if (gAddRes != 0) {
        std::cerr << gai_strerror(gAddRes) << "\n";
        return -2;
    }

    std::cout << "Detecting addresses" << std::endl;

    unsigned int numAddr = 0;
    char ipStr[INET6_ADDRSTRLEN];      // ipv6 length makes sure both ipv4/6 addresses can be
    stored in this variable
```

```

// Now since getaddrinfo() has given us a list of addresses
// we're going to iterate over them and ask user to choose one
// address for program to bind to
for (p = res; p != NULL; p = p->ai_next) {
    void *addr;
    std::string ipVer;

    // if address is ipv4 address
    if (p->ai_family == AF_INET) {
        ipVer          = "IPv4";
        sockaddr_in *ipv4 = reinterpret_cast<sockaddr_in *>(p->ai_addr);
        addr           = &(ipv4->sin_addr);
        ++numOfAddr;
    }

    // if address is ipv6 address
    else {
        ipVer          = "IPv6";
        sockaddr_in6 *ipv6 = reinterpret_cast<sockaddr_in6 *>(p->ai_addr);
        addr           = &(ipv6->sin6_addr);
        ++numOfAddr;
    }

    // convert IPv4 and IPv6 addresses from binary to text form
    inet_ntop(p->ai_family, addr, ipStr, sizeof(ipStr));
    std::cout << "(" << numOfAddr << ") " << ipVer << " : " << ipStr
        << std::endl;
}

// if no addresses found :(
if (!numOfAddr) {
    std::cerr << "Found no host address to use\n";
    return -3;
}

// ask user to choose an address
std::cout << "Enter the number of host address to bind with: ";
unsigned int choice = 0;
bool madeChoice     = false;
do {
    std::cin >> choice;
    if (choice > (numOfAddr + 1) || choice < 1) {
        madeChoice = false;
        std::cout << "Wrong choice, try again!" << std::endl;
    } else
        madeChoice = true;
} while (!madeChoice);

p = res;

// let's create a new socket, socketFD is returned as descriptor
// man socket for more information
// these calls usually return -1 as result of some error
int sockFD = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
if (sockFD == -1) {
    std::cerr << "Error while creating socket\n";
    freeaddrinfo(res);
}

```

```

        return -4;
    }

// Let's bind address to our socket we've just created
int bindR = bind(sockFD, p->ai_addr, p->ai_addrlen);
if (bindR == -1) {
    std::cerr << "Error while binding socket\n";

    // if some error occurs, make sure to close socket and free resources
    close(sockFD);
    freeaddrinfo(res);
    return -5;
}

// finally start listening for connections on our socket
int listenR = listen(sockFD, backLog);
if (listenR == -1) {
    std::cerr << "Error while Listening on socket\n";

    // if some error occurs, make sure to close socket and free resources
    close(sockFD);
    freeaddrinfo(res);
    return -6;
}

// structure large enough to hold client's address
sockaddr_storage client_addr;
socklen_t client_addr_size = sizeof(client_addr);

const std::string response = "Hello World";

// a fresh infinite loop to communicate with incoming connections
// this will take client connections one at a time
// in further examples, we're going to use fork() call for each client connection
while (1) {

    // accept call will give us a new socket descriptor
    int newFD
        = accept(sockFD, (sockaddr *) &client_addr, &client_addr_size);
    if (newFD == -1) {
        std::cerr << "Error while Accepting on socket\n";
        continue;
    }

    // send call sends the data you specify as second param and it's length as 3rd param,
    // also returns how many bytes were actually sent
    auto bytes_sent = send(newFD, response.data(), response.length(), 0);
    close(newFD);
}

close(sockFD);
freeaddrinfo(res);

return 0;
}

```

```

#include <cstring>      // sizeof()
#include <iostream>
#include <string>

// headers for socket(), getaddrinfo() and friends
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>

#include <unistd.h>      // close()

int main(int argc, char *argv[])
{
    // Let's check if port number is supplied or not..
    if (argc != 2) {
        std::cerr << "Run program as 'program <port>'\n";
        return -1;
    }

    auto &portNum = argv[1];
    const unsigned int backLog = 8; // number of connections allowed on the incoming queue

    addrinfo hints, *res, *p; // we need 2 pointers, res to hold and p to iterate over
    memset(&hints, 0, sizeof(hints));

    // for more explanation, man socket
    hints.ai_family = AF_UNSPEC; // don't specify which IP version to use yet
    hints.ai_socktype = SOCK_STREAM; // SOCK_STREAM refers to TCP, SOCK_DGRAM will be?
    hints.ai_flags = AI_PASSIVE;

    // man getaddrinfo
    int gAddRes = getaddrinfo(NULL, portNum, &hints, &res);
    if (gAddRes != 0) {
        std::cerr << gai_strerror(gAddRes) << "\n";
        return -2;
    }

    std::cout << "Detecting addresses" << std::endl;

    unsigned int numOfAddr = 0;
    char ipStr[INET6_ADDRSTRLEN]; // ipv6 length makes sure both ipv4/6 addresses can be
    stored in this variable

    // Now since getaddrinfo() has given us a list of addresses
    // we're going to iterate over them and ask user to choose one
    // address for program to bind to
    for (p = res; p != NULL; p = p->ai_next) {
        void *addr;
        std::string ipVer;

        // if address is ipv4 address
        if (p->ai_family == AF_INET) {
            ipVer = "IPv4";
            sockaddr_in *ipv4 = reinterpret_cast<sockaddr_in *>(p->ai_addr);
        }
    }
}

```

```

        addr          = &(ipv4->sin_addr);
        ++numOfAddr;
    }

    // if address is ipv6 address
    else {
        ipVer          = "IPv6";
        sockaddr_in6 *ipv6 = reinterpret_cast<sockaddr_in6 *>(p->ai_addr);
        addr          = &(ipv6->sin6_addr);
        ++numOfAddr;
    }

    // convert IPv4 and IPv6 addresses from binary to text form
    inet_ntop(p->ai_family, addr, ipStr, sizeof(ipStr));
    std::cout << "(" << numOfAddr << ")" << ipVer << " : " << ipStr
    << std::endl;
}

// if no addresses found :(
if (!numOfAddr) {
    std::cerr << "Found no host address to use\n";
    return -3;
}

// ask user to choose an address
std::cout << "Enter the number of host address to bind with: ";
unsigned int choice = 0;
bool madeChoice    = false;
do {
    std::cin >> choice;
    if (choice > (numOfAddr + 1) || choice < 1) {
        madeChoice = false;
        std::cout << "Wrong choice, try again!" << std::endl;
    } else
        madeChoice = true;
} while (!madeChoice);

p = res;

// let's create a new socket, socketFD is returned as descriptor
// man socket for more information
// these calls usually return -1 as result of some error
int sockFD = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
if (sockFD == -1) {
    std::cerr << "Error while creating socket\n";
    freeaddrinfo(res);
    return -4;
}

// Let's bind address to our socket we've just created
int bindR = bind(sockFD, p->ai_addr, p->ai_addrlen);
if (bindR == -1) {
    std::cerr << "Error while binding socket\n";

    // if some error occurs, make sure to close socket and free resources
    close(sockFD);
    freeaddrinfo(res);
    return -5;
}

```

```

}

// finally start listening for connections on our socket
int listenR = listen(sockFD, backLog);
if (listenR == -1) {
    std::cerr << "Error while Listening on socket\n";

    // if some error occurs, make sure to close socket and free resources
    close(sockFD);
    freeaddrinfo(res);
    return -6;
}

// structure large enough to hold client's address
sockaddr_storage client_addr;
socklen_t client_addr_size = sizeof(client_addr);

const std::string response = "Hello World";

// a fresh infinite loop to communicate with incoming connections
// this will take client connections one at a time
// in further examples, we're going to use fork() call for each client connection
while (1) {

    // accept call will give us a new socket descriptor
    int newFD
        = accept(sockFD, (sockaddr *) &client_addr, &client_addr_size);
    if (newFD == -1) {
        std::cerr << "Error while Accepting on socket\n";
        continue;
    }

    // send call sends the data you specify as second param and it's length as 3rd param,
    // also returns how many bytes were actually sent
    auto bytes_sent = send(newFD, response.data(), response.length(), 0);
    close(newFD);
}

close(sockFD);
freeaddrinfo(res);

return 0;
}

```

## TCP

Hello TCP Server. Hello TCP.

```

#include <cstring>
#include <iostream>
#include <string>

#include <arpa/inet.h>

```

```

#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    // Now we're taking an ipaddress and a port number as arguments to our program
    if (argc != 3) {
        std::cerr << "Run program as 'program <ipaddress> <port>'\n";
        return -1;
    }

    auto &ipAddress = argv[1];
    auto &portNum   = argv[2];

    addrinfo hints, *p;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family   = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags     = AI_PASSIVE;

    int gAddRes = getaddrinfo(ipAddress, portNum, &hints, &p);
    if (gAddRes != 0) {
        std::cerr << gai_strerror(gAddRes) << "\n";
        return -2;
    }

    if (p == NULL) {
        std::cerr << "No addresses found\n";
        return -3;
    }

    // socket() call creates a new socket and returns it's descriptor
    int sockFD = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
    if (sockFD == -1) {
        std::cerr << "Error while creating socket\n";
        return -4;
    }

    // Note: there is no bind() call as there was in Hello TCP Server
    // why? well you could call it though it's not necessary
    // because client doesn't necessarily has to have a fixed port number
    // so next call will bind it to a random available port number

    // connect() call tries to establish a TCP connection to the specified server
    int connectR = connect(sockFD, p->ai_addr, p->ai_addrlen);
    if (connectR == -1) {
        close(sockFD);
        std::cerr << "Error while connecting socket\n";
        return -5;
    }

    std::string reply(15, ' ');

    // recv() call tries to get the response from server
    // BUT there's a catch here, the response might take multiple calls
    // to recv() before it is completely received
    // will be demonstrated in another example to keep this minimal
}

```

```
auto bytes_recv = recv(sockFD, &reply.front(), reply.size(), 0);
if (bytes_recv == -1) {
    std::cerr << "Error while receiving bytes\n";
    return -6;
}

std::cout << "\nClient received: " << reply << std::endl;
close(sockFD);
freeaddrinfo(p);

return 0;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7177/>

## Examples

### Char

char◦ ◦

- char
- 
- 
- 
- char16\_tchar32\_t
- bool
- wchar\_t

sizeof(char) / sizeof(signed char) / sizeof(unsigned char) §3.9.1.1[**basic.fundamental** / 1]

§5.3.3.1[expr.sizeof]sizeof(bool) 3.9.1[**basic.fundamental**]◦

### **char**

C ++5.3.3.1sizeofunsigned char signed charchar1 charsignedunsigned ◦

### C ++ 14

char256UTF-8◦

3.9.1.2signed char short int int long intlong long int◦ ◦ 3.9.1.3 unsigned char unsigned short int unsigned int unsigned long intunsigned long long int◦ ◦ 3.9.1.1 charsigned charunsigned char◦

### C ++ 11

C ++ 11 long longunsigned long long C ++◦ C99C long long unsigned long long C◦

```
1 == sizeof(char) == sizeof(signed char) == sizeof(unsigned char)
<= sizeof(short) == sizeof(unsigned short)
<= sizeof(int) == sizeof(unsigned int)
<= sizeof(long) == sizeof(unsigned long)
```

### C ++ 11

```
1 == sizeof(char) == sizeof(signed char) == sizeof(unsigned char)
<= sizeof(short) == sizeof(unsigned short)
<= sizeof(int) == sizeof(unsigned int)
<= sizeof(long) == sizeof(unsigned long)
```

◦ 3.9.1.3C5.2.4.2.1◦ ;◦ ;◦

signed char	-127 to $2^7 - 1$	8
unsigned char	0 to $2^8 - 1$	8
signed short	-32,767 to $2^{15} - 1$	16
unsigned short	0 to $2^{16} - 1$	16
signed int	-32,767 to $2^{15} - 1$	16
unsigned int	0 to $2^{16} - 1$	16
signed long	-2,147,483,647 to $2^{31} - 1$	32
unsigned long	0 to $2^{32} - 1$	32

## C++ 11

signed long long	-9,223,372,036,854,775,807 to $9,223,372,036,854,775,807 - 2^{63} - 1$	64
unsigned long long	0 to $18,446,744,073,709,551,61502^{64} - 1$	64

◦ 64LP64LLP64LLP6464Windows32intslong ints LP6464Linux32int64long s◦ ◦

## C++ 11

<cstdint>int8\_t int16\_t int32\_t int64\_t intptr\_t uint8\_t uint16\_t uint32\_t uint64\_t uintptr\_t◦

## C++ 11

### char16\_tchar32\_t

#### char16\_tchar32\_t 5.3.3.13.9.1.5

- char16\_tUTF-16uint\_least16\_tuint\_least16\_t ;16◦
- char32\_tUTF-32uint\_least32\_tuint\_least32\_t ;32◦

### bool

bool◦

## wchar\_t

wchar\_t 3.9.1.5◦ ◦ 5.3.3.18,1632;Unicode wchar\_t32Windows wchar\_t16◦ C90ISO 98991990§4.1.5◦

wchar\_t 8,1632。

- Unix Unix wchar\_t 32 UTF-32。
- Windows wchar\_t 16 UTF-16。
- 8 wchar\_t 8。

C ++ 11

Unicode char UTF-8 char16\_t UTF-16 char32\_t UTF-32 wchar\_t。

---

◦

	int	long	
LP32/4/4	16	32	32
ILP32/4/4	32	32	32
LLP64/4/8	32	32	64
LP64/8/8	32	64	64

- 16 Windows LP32。
- 32\* nix Unix Linux Mac OSX Unix Windows ILP32。
- 64 Windows LLP64。
- 64\* nix LP64。

◦

C ++ char。 CHAR\_BIT limits 8.8 POSIX CHAR\_BIT 8 CHAR\_BIT 8,16,3264。

reinterpret\_cast “.....”

```
int x = 42;
int* p = &x;
long addr = reinterpret_cast<long>(p);
std::cout << addr << "\n"; // prints some numeric address,
                           // probably in the architecture's native address format
```

◦

uintptr\_t intptr\_t

```
int x = 42;
int* p = &x;
long addr = reinterpret_cast<long>(p);
std::cout << addr << "\n"; // prints some numeric address,
```

```
// probably in the architecture's native address format
```

## C++ 11

```
int x = 42;
int* p = &x;
long addr = reinterpret_cast<long>(p);
std::cout << addr << "\n"; // prints some numeric address,
                           // probably in the architecture's native address format
```

### uintptr\_t C996.3.2.3C++11C99

```
void void*
uintptr_t char *void * uintptr_t uintptr_t void *
```

- XSI / Open System Interfaces intptr\_t uintptr\_t.
- C; C uintptr\_t. POSIX2.12.3

```
void. void *. void *.
```

- C99§7.18.1

```
typedef u6.2.5;
```

```
uintptr_t.
```

◦ <limits>std::numeric\_limits<T>. C<climits> = C++ 11 <cinttypes>.

- std::numeric\_limits<signed char>::min() SCHAR\_MIN -127.
- std::numeric\_limits<signed char>::max() SCHAR\_MAX 127.
- std::numeric\_limits<unsigned char>::max() UCHAR\_MAX 255.
- std::numeric\_limits<short>::min() SHRT\_MIN -32767.
- std::numeric\_limits<short>::max() SHRT\_MAX 32767.
- std::numeric\_limits<unsigned short>::max() USHRT\_MAX 65535.
- std::numeric\_limits<int>::min() INT\_MIN -32767.
- std::numeric\_limits<int>::max() INT\_MAX 32767.
- std::numeric\_limits<unsigned int>::max() UINT\_MAX 65535.
- std::numeric\_limits<long>::min() LONG\_MIN -2147483647.
- std::numeric\_limits<long>::max() LONG\_MAX 2147483647.
- std::numeric\_limits<unsigned long>::max() ULONG\_MAX 4294967295.

## C++ 11

- std::numeric\_limits<long long>::min() LLONG\_MIN -9223372036854775807.
- std::numeric\_limits<long long>::max() LLONG\_MAX 9223372036854775807.
- std::numeric\_limits<unsigned long long>::max() ULLONG\_MAX 18446744073709551615.

- ```
T max()min()。 C<cfloat>。
```
- digits10。
    - std::numeric\_limits<float>::digits10FLT\_DIG **6**。
    - std::numeric\_limits<double>::digits10DBL\_DIG **10**。
    - std::numeric\_limits<long double>::digits10LDBL\_DIG **10**。
  - min\_exponent10**EE10**。
    - std::numeric\_limits<float>::min\_exponent10FLT\_MIN\_10\_EXP **-37**。
    - std::numeric\_limits<double>::min\_exponent10DBL\_MIN\_10\_EXP **-37**。
    - std::numeric\_limits<long double>::min\_exponent10LDBL\_MIN\_10\_EXP **-37**。
  - max\_exponent10**EE10**。
    - std::numeric\_limits<float>::max\_exponent10FLT\_MIN\_10\_EXP **37**。
    - std::numeric\_limits<double>::max\_exponent10DBL\_MIN\_10\_EXP **37**。
    - std::numeric\_limits<long double>::max\_exponent10LDBL\_MIN\_10\_EXP **37**。
  - is\_iec559**IEC 559 / IEEE 754**。

```
long doubledouble float ;long doubledoubledoublefloat。 。
```

```
T std::numeric_limits<T>::radixT。
```

```
std::numeric_limits<T>::is_iec559IEC 559 / IEEE 754。
```

◦

```
// Suppose that on this implementation, the range of signed char is -128 to +127 and
// the range of unsigned char is 0 to 255
int x = 12345;
signed char sc = x;    // sc has an implementation-defined value
unsigned char uc = x; // uc is initialized to 57 (i.e., 12345 modulo 256)
```

◦

```
enum E {
    RED,
    GREEN,
    BLUE,
};
using T = std::underlying_type<E>::type; // implementation-defined
```

```
intintunsigned int。 Tint unsigned intshort long long。
```

```
sizeof。
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1363/>

◦

## Examples

“class” class struct enum struct enum class ◦

- ;◦ pp + 1◦ 0◦ ◦

```
class Empty_1 {};                                // sizeof(Empty_1)      == 1
class Empty_2 {};                                // sizeof(Empty_2)      == 1
class Derived : Empty_1 {};                      // sizeof(Derived)      == 1
class DoubleDerived : Empty_1, Empty_2 {};        // sizeof(DoubleDerived) == 1
class Holder { Empty_1 e; };                     // sizeof(Holder)       == 1
class DoubleHolder { Empty_1 e1; Empty_2 e2; };   // sizeof(DoubleHolder) == 2
class DerivedHolder : Empty_1 { Empty_1 e; };     // sizeof(DerivedHolder) == 2
```

◦

```
class Empty_1 {};                                // sizeof(Empty_1)      == 1
class Empty_2 {};                                // sizeof(Empty_2)      == 1
class Derived : Empty_1 {};                      // sizeof(Derived)      == 1
class DoubleDerived : Empty_1, Empty_2 {};        // sizeof(DoubleDerived) == 1
class Holder { Empty_1 e; };                     // sizeof(Holder)       == 1
class DoubleHolder { Empty_1 e1; Empty_2 e2; };   // sizeof(DoubleHolder) == 2
class DerivedHolder : Empty_1 { Empty_1 e; };     // sizeof(DerivedHolder) == 2
```

Ssizeof(int)xsizeof(char\*)y◦ ◦

- t1, t2,...tN / sizeof(t1) + sizeof(t2) + ... + sizeof(tN) ◦ ◦

```
class Empty_1 {};                                // sizeof(Empty_1)      == 1
class Empty_2 {};                                // sizeof(Empty_2)      == 1
class Derived : Empty_1 {};                      // sizeof(Derived)      == 1
class DoubleDerived : Empty_1, Empty_2 {};        // sizeof(DoubleDerived) == 1
class Holder { Empty_1 e; };                     // sizeof(Holder)       == 1
class DoubleHolder { Empty_1 e1; Empty_2 e2; };   // sizeof(DoubleHolder) == 2
class DerivedHolder : Empty_1 { Empty_1 e; };     // sizeof(DerivedHolder) == 2
```

- nn◦ memNalignof(memN) nalignofalignof◦ alignofalignof◦ ◦ alignofalignof◦ “”◦ alignofalignof◦ alignof◦

```
class Empty_1 {};                                // sizeof(Empty_1)      == 1
class Empty_2 {};                                // sizeof(Empty_2)      == 1
class Derived : Empty_1 {};                      // sizeof(Derived)      == 1
class DoubleDerived : Empty_1, Empty_2 {};        // sizeof(DoubleDerived) == 1
class Holder { Empty_1 e; };                     // sizeof(Holder)       == 1
class DoubleHolder { Empty_1 e1; Empty_2 e2; };   // sizeof(DoubleHolder) == 2
class DerivedHolder : Empty_1 { Empty_1 e; };     // sizeof(DerivedHolder) == 2
```

## C++ 11

- alignas<sub>0</sub> Chars<sub>5</sub><sub>8.4</sub><sup>“”4</sup>

```
class Empty_1 {};                                // sizeof(Empty_1)      == 1
class Empty_2 {};                                // sizeof(Empty_2)      == 1
class Derived : Empty_1 {};                      // sizeof(Derived)      == 1
class DoubleDerived : Empty_1, Empty_2 {};        // sizeof(DoubleDerived) == 1
class Holder { Empty_1 e; };                     // sizeof(Holder)       == 1
class DoubleHolder { Empty_1 e1; Empty_2 e2; };   // sizeof(DoubleHolder) == 2
class DerivedHolder : Empty_1 { Empty_1 e; };     // sizeof(DerivedHolder) == 2
```

- . .
- .

unsigned char<sub>0</sub> unsigned char<sub>8</sub>char<sub>256256{0,1...255}</sub> 42<sub>00101010</sub> .

signed char signed char<sub>88</sub> .

◦

◦ unsigned int<sub>6402<sup>32</sup> - 1</sub> 32 .

◦ ◦ unsigned short<sub>00010100101010115291</sub> short<sub>5291</sub> .

◦

◦ floatdouble IEEE 7543264 float<sub>2381</sub> . . “” .

◦ TT .

◦

```
int a[5][3];
```

a<sub>5</sub>int<sub>5</sub> a[0]a[0][0] a[0][1] a[0][2] a[1]a[1]a[1][0] a[1][1]a[1][2] . .

<https://riptutorial.com/zh-TW/cplusplus/topic/9329/>

# 71:

C ++◦◦◦

21◦

- 161,2,4,81632.148216◦

- 
- voidchar◦
- CVCV◦

C ++ 03C ++ 11alignof alignas ◦

## Examples

C ++ 11

alignof◦ std::size\_t◦

```
#include <iostream>
int main() {
    std::cout << "The alignment requirement of int is: " << alignof(int) << '\n';
}
```

int4

◦◦◦

C ++ 11

alignas◦

- alignas(x) xx◦
- alignas(T) Talignof(T)◦

alignas◦

bufintunsigned char◦

```
alignas(int) unsigned char buf[sizeof(int)];
new (buf) int(42);
```

alignas

```
alignas(int) unsigned char buf[sizeof(int)];
new (buf) int(42);
```

alignas = 2; std::max\_align\_t std::max\_align\_t = . . .

C++ 17 operator new

<https://riptutorial.com/zh-TW/cplusplus/topic/9249/>

# 72:

- `[[details]]`
- `[[detailsarguments]]`
- `__attribute__  
GCC / Clang / IBM`
- `__declspec  
MSVC`

## Examples

`[[[]]]`

C++ 11

C++ 11 `[[noreturn]]`。 `return` `void`。 `std::terminate` `std::exit`。 `longjmp`。

`std::terminate [[noreturn]]`

```
[[noreturn]] void ownAssertFailureHandler(std::string message) {
    std::cerr << message << std::endl;
    if (THROW_EXCEPTION_ON_ASSERT)
        throw AssertException(std::move(message));
    std::terminate();
}
```

`return`。 `ownAssertFailureHandler`

```
[[noreturn]] void ownAssertFailureHandler(std::string message) {
    std::cerr << message << std::endl;
    if (THROW_EXCEPTION_ON_ASSERT)
        throw AssertException(std::move(message));
    std::terminate();
}
```

```
[[noreturn]] void ownAssertFailureHandler(std::string message) {
    std::cerr << message << std::endl;
    if (THROW_EXCEPTION_ON_ASSERT)
        throw AssertException(std::move(message));
    std::terminate();
}
```

`[[noreturn]] void`。

```
[[noreturn]] void ownAssertFailureHandler(std::string message) {
    std::cerr << message << std::endl;
    if (THROW_EXCEPTION_ON_ASSERT)
        throw AssertException(std::move(message));
    std::terminate();
}
```

```
}
```

- std ::
- std ::
- std :: quick\_exit
- std ::
- std ::
- std :: rethrow\_exception
- std :: throw\_with\_nested
- std :: nested\_exception :: rethrow\_nested

[[[]]]

C++ 17

```
caseswitch◦ 'break`◦ ◦
```

C++ 17◦ break[[fallthrough]] [[fallthrough]]◦

```
switch(input) {  
    case 2011:  
    case 2014:  
    case 2017:  
        std::cout << "Using modern C++" << std::endl;  
        [[fallthrough]]; // > No warning  
    case 1998:  
    case 2003:  
        standard = input;  
}
```

[[fallthrough]]◦

[[deprecated]][[deprecated“reason”]]

C++ 14

C++ 14◦ [[deprecated]]◦ [[deprecated("reason")]]◦

```
void function(std::unique_ptr<A> &&a);  
  
// Provides specific message which helps other programmers fixing there code  
[[deprecated("Use the variant with unique_ptr instead, this function will be removed in the  
next release")]]  
void function(std::auto_ptr<A> a);  
  
// No message, will result in generic warning if called.  
[[deprecated]]  
void function(A *a);
```

- 
- **typedef**
-

• • • •

c++ 14 7.6.5

**[[nodiscard]]**

C ++ 17

[ [nodiscard] ] 。 。

2

```
template<typename Function>
[[nodiscard]] Finally<std::decay_t<Function>> onExit(Function &&f);

void f(int &i) {
    assert(i == 0);                                // Just to make comments clear!
    ++i;   // i == 1
    auto exit1 = onExit([&i]{ --i; });           // Reduce by 1 on exiting f()
    ++i;   // i == 2
    onExit([&i]{ --i; });                         // BUG: Reducing by 1 directly
  // Compiler warning expected
    std::cout << i << std::endl;                 // Expected: 2, Real: 1
}
```

```
[[nodiscard]] .
```

`onExitFinally / onExit Finally / ScopeExit`。

[[maybe\_unused]]

[ [maybe unused] ] ° ° °

8

```
[[maybe_unused]] auto mapInsertResult = configuration.emplace("LicenseInfo",
stringifiedLicenseInfo);
assert(mapInsertResult.second); // We only get called during startup, so we can't be in the
map
```

◦◦◦ [ [maybe\_unused] ]◦◦◦

```
[[maybe_unused]] auto mapInsertResult = configuration.emplace("LicenseInfo",
stringifiedLicenseInfo);
assert(mapInsertResult.second); // We only get called during startup, so we can't be in the
map
```

[ [maybe\_unused] ] °

<https://riptutorial.com/zh-TW/cplusplus/topic/5251/>

# 73:

- C ++union。 C ++。

std::variant C ++ 17union。

- 。

## Examples

◦

```
union U {
    int a;
    short b;
    float c;
};

U u;

//Address of a and b will be equal
(void*)&u.a == (void*)&u.b;
(void*)&u.a == (void*)&u.c;

//Assigning to any union member changes the shared memory of all members
u.c = 4.f;
u.a = 5;
u.c != 4.f;
```

◦

```
struct AnyType {
    enum {
        IS_INT,
        IS_FLOAT
    } type;

    union Data {
        int as_int;
        float as_float;
    } value;

    AnyType(int i) : type(IS_INT) { value.as_int = i; }
    AnyType(float f) : type(IS_FLOAT) { value.as_float = f; }

    int get_int() const {
        if(type == IS_INT)
            return value.as_int;
        else
            return (int)value.as_float;
    }

    float get_float() const {
        if(type == IS_FLOAT)
            return value.as_float;
    }
}
```

```
        else
            return (float)value.as_int;
    }
};
```

```
union U {
    int a;
    short b;
    float c;
};

U u;

u.a = 10;
if (u.b == 10) {
    // this is undefined behavior since 'a' was the last member to be
    // written to. A lot of compilers will allow this and might issue a
    // warning, but the result will be "as expected"; this is a compiler
    // extension and cannot be guaranteed across compilers (i.e. this is
    // not compliant/portable code).
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2678/>

# 74:

◦

C++ 11◦

- `regex_match //`
- `regex_search //`
- `regex_replace //`
- `regex_token_iterator //`◦
- `regex_iterator //`◦

```
bool regex_match(BidirectionalIterator first,
BidirectionalIterator last, smatch& sm, const regex&
re, regex_constraints::match_flag_type flags)
```

```
bool regex_match(const string& str, smatch& sm, const
regex re&, regex_constraints::match_flag_type flags)
```

**BidirectionalIterators**  
`match`  
`cmatch`  
`match_results`  
`BidirectionalIterator`  
`smatch`  
**committed**  
`ref`  
`first`  
`last`

**string**  
`const char*`  
**L-Value**  
`string R`  
`string smatch`  
`cmatch`  
`str`  
`match_results`  
`smatch`  
`restr`

## Examples

### `regex_match` `regex_search`

```
const auto input = "Some people, when confronted with a problem, think \"I know, I'll use
regular expressions.\"";
smatch sm;

cout << input << endl;

// If input ends in a quotation that contains a word that begins with "reg" and another word
begining with "ex" then capture the preceeding portion of input
if (regex_match(input, sm, regex("(.*).*\\"\\breg.*\\bex.*\"\\s*$"))) {
    const auto capture = sm[1].str();

    cout << '\t' << capture << endl; // Outputs: "\tSome people, when confronted with a
problem, think\n"

    // Search our capture for "a problem" or "# problems"
    if(regex_search(capture, sm, regex("(a|d+)\\s+problems?")))
        const auto count = sm[1] == "a"s ? 1 : stoi(sm[1]);

        cout << '\t' << count << (count > 1 ? " problems\n" : " problem\n");
        // Outputs: "\t1
problem\n"
        cout << "Now they have " << count + 1 << " problems.\n";
        // Ouputs: "Now they have 2
problems\n"
    }
}
```

## regex\_replace

### One True Brace

```
const auto input = "if (KnR)\n\tfoo();\nif (spaces) {\n    foo();\n}\nif\n(allman)\n{\n\tfoo();\n}\nif (horstmann)\n{\n\tfoo();\n}\nif (pico)\n{\n\tfoo();\n}\nif\n(whitesmiths)\n{\n\t(\n\tfoo();\n\t)\n}s;\n\ncout << input << regex_replace(input, regex("( .?)\\s*\\{?\\s*(.+?;)\\s*\\}\\?\\s*"), "$1\n\\n$t$2\\n}\\n") << endl;
```

## regex\_token\_iterator

```
std::regex_token_iterator<string> m_vecFields{ sregex_token_iterator(cbegin(input), cend(input), re, 1), sregex_token_iterator() };
```

regex`L`。 `R`。

```
const auto input = "please split,this,csv, ,line,\\\",\\n"s;
const regex re{ "(?:[^\\\\\\\",]|\\\\\\\")+(?:,|\\$)" };
copy(cbegin(m_vecFields), cend(m_vecFields), ostream_iterator<string>(cout, "\\n"));
```

## regex\_iterator

```
regex_iterator<string> regex_iterator(regex_iterator regex_iterator::match_result); C++
```

```
enum TOKENS {
    NUMBER,
    ADDITION,
    SUBTRACTION,
    MULTIPLICATION,
    DIVISION,
    EQUALITY,
    OPEN_PARENTHESIS,
    CLOSE_PARENTHESIS
};
```

```
const auto input = "42/2 + -8\t=\n(2 + 2) * 2 * 2 -3"s regex_iterator
```

```
enum TOKENS {
    NUMBER,
    ADDITION,
    SUBTRACTION,
    MULTIPLICATION,
    DIVISION,
    EQUALITY,
    OPEN_PARENTHESIS,
    CLOSE_PARENTHESIS
};
```

## regexLR Visual Studio regex\_iterator Bug

```
std::vector<std::string> split(const std::string &str, std::string regex)
{
    std::regex r{ regex };
    std::sregex_token_iterator start{ str.begin(), str.end(), r, -1 }, end;
    return std::vector<std::string>(start, end);
}
```

```
std::vector<std::string> split(const std::string &str, std::string regex)
{
    std::regex r{ regex };
    std::sregex_token_iterator start{ str.begin(), str.end(), r, -1 }, end;
    return std::vector<std::string>(start, end);
}
```

const string input = regex\_match(input, regex("\\d\*")) regex\_match(input, regex("\\d+"))  
"123"7

```
regex_match(input, regex("\\d{7,}"))
```

input "123456789012". **nm**input711

```
regex_match(input, regex("\\d{7,}"))
```

[7,11]"123456789"

```
regex_match(input, regex("\\d{7,}"))
```

10\d{7,10}.\d{0,1}.

```
regex_match(input, regex("\\d{7,}"))
```

\d{7} 7. 734.

\d{3,4}? \d{7} \d{3,4}? 343. input "1234567".

---

o o o regex\_match(input, regex("\\d{3,4}+\d{7}")) "1234567890" input \d{3,4}+3 \d{3,4}+4  
o

```
regex_match(input, regex("\\d{7,}"))
```

input

123 456 7890  
123-456-7890  
(123)456-7890  
123456 - 7890

input

12345 - 67890

. \*34. '0'.\*\d{3,4}.\*.

## C ++4

- ^
- \$
- \b\W
- \B\w

```
auto input = "+1--12*123/+1234"s;
smatch sm;

if(regex_search(input, sm, regex{ "(?:^|\\b\\W)([+-]?\d+)" })) {
    do {
        cout << sm[1] << endl;
        input = sm.suffix().str();
    } while(regex_search(input, sm, regex{ "(?:^\\W|\\b\\W)([+-]?\d+)" }));
}
```

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/1681/>

# 75: /GCC

## Examples

\*\*\*

◦

```
#include <iostream>

int main(int argc, char *argv[])
{
{
    int i = 2;
}

std::cout << i << std::endl; // i is not in the scope of the main function

return 0;
}
```

```
#include <iostream>

int main(int argc, char *argv[])
{
{
    int i = 2;
}

std::cout << i << std::endl; // i is not in the scope of the main function

return 0;
}
```

```
#include <iostream> std::cout
```

```
#include <iostream>

int main(int argc, char *argv[])
{
{
    int i = 2;
}

std::cout << i << std::endl; // i is not in the scope of the main function

return 0;
}
```

```
#include <iostream>

int main(int argc, char *argv[])
{
```

```
{  
    int i = 2;  
}  
  
std::cout << i << std::endl; // i is not in the scope of the main function  
  
return 0;  
}
```

```
#include <iostream>  
  
int main(int argc, char *argv[])  
{  
    {  
        int i = 2;  
    }  
  
    std::cout << i << std::endl; // i is not in the scope of the main function  
  
    return 0;  
}
```

◦ ◦

~\*\*\*~

◦ ◦

## QMAKE

```
LIBS += nameOfLib
```

## cmake

```
LIBS += nameOfLib
```

## g ++

```
LIBS += nameOfLib
```

## .cppfunctionsModule.cpp

```
LIBS += nameOfLib
```

\*\*\*

```
#include "someFile.hpp" .
```

## QMAKE

```
INCLUDEPATH += dir/Of/File
```

## cmake

```
INCLUDEPATH += dir/Of/File
```

## g ++

```
INCLUDEPATH += dir/Of/File
```

/GCC <https://riptutorial.com/zh-TW/cplusplus/topic/4256/--gcc->

'const'。 const。 ""。 。

◦ const◦

const◦ 。

## Examples

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

class A {
public:
    map<string, string> * mapOfStrings;
public:
    A() {
        mapOfStrings = new map<string, string>();
    }

    void insertEntry(string const & key, string const & value) const {
        (*mapOfStrings)[key] = value;           // This works? Yes it does.
        delete mapOfStrings;                  // This also works
        mapOfStrings = new map<string, string>(); // This * does * not work
    }

    void refresh() {
        delete mapOfStrings;
        mapOfStrings = new map<string, string>(); // Works as refresh is non const function
    }

    void getEntry(string const & key) const {
        cout << mapOfStrings->at(key);
    }
};

int main(int argc, char* argv[]) {

    A var;
    var.insertEntry("abc", "abcValue");
    var.getEntry("abc");
    getchar();
    return 0;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7120/>

- C ++STL。 STL<sub>structs/classes/arrays</sub>。

## Examples

;const◦

```
void calculate(int a, int b, int& c, int& d, int& e, int& f) {
    c = a + b;
    d = a - b;
    e = a * b;
    f = a / b;
}
```

```
void calculate(int a, int b, int& c, int& d, int& e, int& f) {
    c = a + b;
    d = a - b;
    e = a * b;
    f = a / b;
}
```

'OUT'# #define ◦ ;

```
void calculate(int a, int b, int& c, int& d, int& e, int& f) {
    c = a + b;
    d = a - b;
    e = a * b;
    f = a / b;
}
```

## std :: tuple

### C ++ 11

std::tuple

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

### C ++ 17

### C ++ 17

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

tuple std::get

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

### std::tie

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

### std::ignore

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

## C++ 17

### std::tie

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

### std::tie std::make\_tuple ◦

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

```
std::tuple<int, int, int, int> foo(int a, int b) { // or auto (C++14)
    return std::make_tuple(a + b, a - b, a * b, a / b);
}
```

### std::forward\_as\_tuple std::tie ;◦

## std :: array

## C++ 11

### std::array◦

```
std::array<int, 4> bar(int a, int b) {
    return { a + b, a - b, a * b, a / b };
}
```

### int bar[4] C◦ C++ std◦ atsize ◦

## std :: pair

```
std::pair
```

```
#include <utility>
std::pair<int, int> foo(int a, int b) {
    return std::make_pair(a+b, a-b);
}
```

C++ 11 `std::make_pair`

C++ 11

```
#include <utility>
std::pair<int, int> foo(int a, int b) {
    return std::make_pair(a+b, a-b);
}
```

firstsecond`std::pair`

```
#include <utility>
std::pair<int, int> foo(int a, int b) {
    return std::make_pair(a+b, a-b);
}
```

10

**struct**

```
struct
```

C++ 11

```
struct foo_return_type {
    int add;
    int sub;
    int mul;
    int div;
};

foo_return_type foo(int a, int b) {
    return {a + b, a - b, a * b, a / b};
}

auto calc = foo(5, 12);
```

C++ 11

```
struct foo_return_type {
    int add;
    int sub;
    int mul;
    int div;
};

foo_return_type foo(int a, int b) {
    return {a + b, a - b, a * b, a / b};
```

```
}
```

```
auto calc = foo(5, 12);
```

```
struct calcfoo()
```

```
struct foo_return_type {
    int add;
    int sub;
    int mul;
    int div;
};

foo_return_type foo(int a, int b) {
    return {a + b, a - b, a * b, a / b};
}

auto calc = foo(5, 12);
```

17 -7 60 0

struct ◦ ◦

C ++ 17

struct◦ in-parametersout

```
struct foo_return_type {
    int add;
    int sub;
    int mul;
    int div;
};

foo_return_type foo(int a, int b) {
    return {a + b, a - b, a * b, a / b};
}

auto calc = foo(5, 12);
```

◦ struct◦ ◦

C ++ 17

C ++ 17 `std::tie()`

```
std::map<std::string, int> m;

// insert an element into the map and check if insertion succeeded
auto [iterator, success] = m.insert({"Hello", 42});

if (success) {
    // your code goes here
}
```

```
// iterate over all elements without having to use the cryptic 'first' and 'second' names
for (auto const& [key, value] : m) {
    std::cout << "The value for " << key << " is " << value << '\n';
}
```

std::pair std::tuple

```
std::map<std::string, int> m;

// insert an element into the map and check if insertion succeeded
auto [iterator, success] = m.insert({"Hello", 42});

if (success) {
    // your code goes here
}

// iterate over all elements without having to use the cryptic 'first' and 'second' names
for (auto const& [key, value] : m) {
    std::cout << "The value for " << key << " is " << value << '\n';
}
```

“”◦ tuple\_size tuple\_elementget

```
std::map<std::string, int> m;

// insert an element into the map and check if insertion succeeded
auto [iterator, success] = m.insert({"Hello", 42});

if (success) {
    // your code goes here
}

// iterate over all elements without having to use the cryptic 'first' and 'second' names
for (auto const& [key, value] : m) {
    std::cout << "The value for " << key << " is " << value << '\n';
}
```

```
std::map<std::string, int> m;

// insert an element into the map and check if insertion succeeded
auto [iterator, success] = m.insert({"Hello", 42});

if (success) {
    // your code goes here
}

// iterate over all elements without having to use the cryptic 'first' and 'second' names
for (auto const& [key, value] : m) {
    std::cout << "The value for " << key << " is " << value << '\n';
}
```

## C ++ 11

```
template <class F>
void foo(int a, int b, F consumer) {
    consumer(a + b, a - b, a * b, a / b);
```

```

}

// use is simple... ignoring some results is possible as well
foo(5, 12, [](int sum, int , int , int ){
    std::cout << "sum is " << sum << '\n';
});
```

“”。

## C++ 17

```

template <class F>
void foo(int a, int b, F consumer) {
    consumer(a + b, a - b, a * b, a / b);
}

// use is simple... ignoring some results is possible as well
foo(5, 12, [](int sum, int , int , int ){
    std::cout << "sum is " << sum << '\n';
});
```

## C++ 14C++ 11。

### std :: vector

std::vector<int> std::vector

```

#include <vector>
#include <iostream>

// the following function returns all integers between and including 'a' and 'b' in a vector
// (the function can return up to std::vector::max_size elements with the vector, given that
// the system's main memory can hold that many items)
std::vector<int> fillVectorFrom(int a, int b) {
    std::vector<int> temp;
    for (int i = a; i <= b; i++) {
        temp.push_back(i);
    }
    return temp;
}

int main() {
    // assigns the filled vector created inside the function to the new vector 'v'
    std::vector<int> v = fillVectorFrom(1, 10);

    // prints "1 2 3 4 5 6 7 8 9 10 "
    for (int i = 0; i < v.size(); i++) {
        std::cout << v[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

◦◦

```
template<typename Incrementable, typename OutputIterator>
void generate_sequence(Incrementable from, Incrementable to, OutputIterator output) {
    for (Incrementable k = from; k != to; ++k)
        *output++ = k;
}
```

```
template<typename Incrementable, typename OutputIterator>
void generate_sequence(Incrementable from, Incrementable to, OutputIterator output) {
    for (Incrementable k = from; k != to; ++k)
        *output++ = k;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/487/>

# 78:

- C ++3 for while do ... while ◦

- ;
- ;
- for *for-init-statement*; *condition* ; *expression statement* ;
- for *range-declaration* *for-range-initializer* ;
- 
- 

algorithm◦

- 
- 

## Examples

### C ++ 11

for

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

v val◦

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

## C++ 17

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

## C++ 20 Ranges TS

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

auto val<sup>o</sup> const auto &val<sup>o</sup> auto ;<sup>o</sup>

◦

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

const const

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

const<sup>o</sup> ◦

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

“”for

- ```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

- begin()end()。

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

- begin(type)end(type)end(type) type。

```
vector<float> v = {0.4f, 12.5f, 16.234f};

for(auto val: v)
{
    std::cout << val << " ";
}

std::cout << std::endl;
```

forloop bodycondition**true**。 initialization statement。 iteration execution。

for

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

- initialization statement for。 int i = 0, a = 2, b = 3。 。 。
- condition false。
- iteration execution iteration executionforbreak goto return。 iteration executiona++,

```
b+=10, c=b+a。
```

forwhile

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

foro

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

- - . .
- forforo
- initialization statementconditiono

010

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

- int counter = 0counter**0**.foro
- counter <= 10counter**10**.true o false o
- ++countercounter**1**o

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

```
while

for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

break gotoreturno

<algorithm>**STL** vector

```
for /*initialization statement*/; /*condition*/; /*iteration execution*/
{
    // body of the loop
}
```

whilefalse 。 。

## 09

```
int i = 0;
while (i < 10)
{
    std::cout << i << " ";
    ++i; // Increment counter
}
std::cout << std::endl; // End of line; "0 1 2 3 4 5 6 7 8 9" is printed to the console
```

## C++ 17

### C++ 17

```
int i = 0;
while (i < 10)
{
    std::cout << i << " ";
    ++i; // Increment counter
}
std::cout << std::endl; // End of line; "0 1 2 3 4 5 6 7 8 9" is printed to the console
```

```
int i = 0;
while (i < 10)
{
    std::cout << i << " ";
    ++i; // Increment counter
}
std::cout << std::endl; // End of line; "0 1 2 3 4 5 6 7 8 9" is printed to the console
```

whiledo...while。 **do-while** 。

forwhileo

```
for (int i = 0; i < 5; ++i) {
```

```

        do_something(i);
    }
    // i is no longer in scope.

for (auto& a : some_container) {
    a.do_something();
}
// a is no longer in scope.

while(std::shared_ptr<Object> p = get_object()) {
    p->do_something();
}
// p is no longer in scope.

```

do...while do...while ;

```

for (int i = 0; i < 5; ++i) {
    do_something(i);
}
// i is no longer in scope.

for (auto& a : some_container) {
    a.do_something();
}
// a is no longer in scope.

while(std::shared_ptr<Object> p = get_object()) {
    p->do_something();
}
// p is no longer in scope.

```

do...while while o

## Do-while

*do-while* while o o

0 false

```

int i =0;
do
{
    std::cout << i;
    ++i; // Increment counter
}
while (i < 0);
std::cout << std::endl; // End of line; 0 is printed to the console

```

while (condition); while (condition); while (condition); **do-while** o

**do-while** false

```

int i =0;
do
{

```

```
    std::cout << i;
    ++i; // Increment counter
}
while (i < 0);
std::cout << std::endl; // End of line; 0 is printed to the console
```

break gotoreturn**whilefalse**。

```
int i = 0;
do
{
    std::cout << i;
    ++i; // Increment counter
}
while (i < 0);
std::cout << std::endl; // End of line; 0 is printed to the console
```

## do-while

```
int i = 0;
do
{
    std::cout << i;
    ++i; // Increment counter
}
while (i < 0);
std::cout << std::endl; // End of line; 0 is printed to the console
```

◦◦ breakcontinue◦

break◦

```
for (int i = 0; i < 10; i++)
{
    if (i == 4)
        break; // this will immediately exit our loop
    std::cout << i << '\n';
}
```

```
for (int i = 0; i < 10; i++)
{
    if (i == 4)
        break; // this will immediately exit our loop
    std::cout << i << '\n';
}
```

continue◦

```
for (int i = 0; i < 10; i++)
{
    if (i == 4)
        break; // this will immediately exit our loop
    std::cout << i << '\n';
}
```

```
for (int i = 0; i < 10; i++)
{
    if (i == 4)
        break; // this will immediately exit our loop
    std::cout << i << '\n';
}
```

breakcontinue。 。 breakfor

```
for (int i = 0; i < 10; i++)
{
    if (i == 4)
        break; // this will immediately exit our loop
    std::cout << i << '\n';
}
```

continue

```
for (int i = 0; i < 10; i++)
{
    if (i == 4)
        break; // this will immediately exit our loop
    std::cout << i << '\n';
}
```

-

for。

```
template<class Iterator, class Sentinel=Iterator>
struct range_t {
    Iterator b;
    Sentinel e;
    Iterator begin() const { return b; }
    Sentinel end() const { return e; }
    bool empty() const { return begin()==end(); }
    range_t without_front( std::size_t count=1 ) const {
        if (std::is_same< std::random_access_iterator_tag, typename
std::iterator_traits<Iterator>::iterator_category >{} ) {
            count = (std::min)(std::size_t(std::distance(b,e)), count);
        }
        return {std::next(b, count), e};
    }
    range_t without_back( std::size_t count=1 ) const {
        if (std::is_same< std::random_access_iterator_tag, typename
std::iterator_traits<Iterator>::iterator_category >{} ) {
            count = (std::min)(std::size_t(std::distance(b,e)), count);
        }
        return {b, std::prev(e, count)};
    }
};

template<class Iterator, class Sentinel>
range_t<Iterator, Sentinel> range( Iterator b, Sentinel e ) {
    return {b,e};
}
```

```

template<class Iterable>
auto range( Iterable& r ) {
    using std::begin; using std::end;
    return range(begin(r),end(r));
}

template<class C>
auto except_first( C& c ) {
    auto r = range(c);
    if (r.empty()) return r;
    return r.without_front();
}

```

```

template<class Iterator, class Sentinel=Iterator>
struct range_t {
    Iterator b;
    Sentinel e;
    Iterator begin() const { return b; }
    Sentinel end() const { return e; }
    bool empty() const { return begin()==end(); }
    range_t without_front( std::size_t count=1 ) const {
        if (std::is_same< std::random_access_iterator_tag, typename
std::iterator_traits<Iterator>::iterator_category >{} ) {
            count = (std::min)(std::size_t(std::distance(b,e)), count);
        }
        return {std::next(b, count), e};
    }
    range_t without_back( std::size_t count=1 ) const {
        if (std::is_same< std::random_access_iterator_tag, typename
std::iterator_traits<Iterator>::iterator_category >{} ) {
            count = (std::min)(std::size_t(std::distance(b,e)), count);
        }
        return {b, std::prev(e, count)};
    }
};

```

```

template<class Iterator, class Sentinel>
range_t<Iterator, Sentinel> range( Iterator b, Sentinel e ) {
    return {b,e};
}

```

```

template<class Iterable>
auto range( Iterable& r ) {
    using std::begin; using std::end;
    return range(begin(r),end(r));
}

```

```

template<class C>
auto except_first( C& c ) {
    auto r = range(c);
    if (r.empty()) return r;
    return r.without_front();
}

```

```

template<class Iterator, class Sentinel=Iterator>
struct range_t {
    Iterator b;
    Sentinel e;
    Iterator begin() const { return b; }
    Sentinel end() const { return e; }

```

```

bool empty() const { return begin()==end(); }
range_t without_front( std::size_t count=1 ) const {
    if (std::is_same< std::random_access_iterator_tag, typename
std::iterator_traits<Iterator>::iterator_category >{} ) {
        count = (std::min)(std::size_t(std::distance(b,e)), count);
    }
    return {std::next(b, count), e};
}
range_t without_back( std::size_t count=1 ) const {
    if (std::is_same< std::random_access_iterator_tag, typename
std::iterator_traits<Iterator>::iterator_category >{} ) {
        count = (std::min)(std::size_t(std::distance(b,e)), count);
    }
    return {b, std::prev(e, count)};
}
};

template<class Iterator, class Sentinel>
range_t<Iterator, Sentinel> range( Iterator b, Sentinel e ) {
    return {b,e};
}
template<class Iterable>
auto range( Iterable& r ) {
    using std::begin; using std::end;
    return range(begin(r),end(r));
}

template<class C>
auto except_first( C& c ) {
    auto r = range(c);
    if (r.empty()) return r;
    return r.without_front();
}

```

forfor(:range\_expression)for◦

<https://riptutorial.com/zh-TW/cplusplus/topic/589/>

# 79:

+	-	*	/	\		<<	>>				
+ =	- =	* =	/ =	=	\ =	=	=	<< =	>> =	=	
==	=	<	>	<=	>=	&&			*.	- >	*

&&	

## Examples

◦

- (... op pack)

```
((Pack1 op Pack2) op ...) op PackN
```

- **Unary Right Fold** (pack op ...)

```
((Pack1 op Pack2) op ...) op PackN
```

```
((Pack1 op Pack2) op ...) op PackN
```

◦

- - (value op ... op pack) -

```
((Value op Pack1) op Pack2) op ... op PackN
```

- **Binary Right Fold** (pack op ... op value) -

```
((Value op Pack1) op Pack2) op ... op PackN
```

```
((Value op Pack1) op Pack2) op ... op PackN
```

◦ C++ 11

```
template <class... Ts>
void print_all(std::ostream& os, Ts const&... args) {
```

```
using expander = int[];
(void)expander{0,
    (void(os << args), 0)...
};

}
```

## fold

```
template <class... Ts>
void print_all(std::ostream& os, Ts const&... args) {
    using expander = int[];
    (void)expander{0,
        (void(os << args), 0)...
    };
}


```

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/2676/>

# 80:

- Class.....

- type \* ptr =Class :: member; //
- type Class :: \* ptr =Class :: member; //
- ◦ ;
- Class \* p =instance;
- ◦ ptr =Class :: i; //i
- instance. \* ptr = 1; //i
- p - > \* ptr = 1; //pi
- ◦ ptr =Class :: F; //F'
- \* PTR5; //F
- - > \* PTR6; //pF.

## Examples

### static C / C ++

- class ;
- public protectedprivate◦

staticclass

```
typedef int Fn(int); // Fn is a type-of function that accepts an int and returns an int

// Note that MyFn() is of type 'Fn'
int MyFn(int i) { return 2*i; }

class Class {
public:
    // Note that Static() is of type 'Fn'
    static int Static(int i) { return 3*i; }
}; // Class

int main() {
    Fn *fn;      // fn is a pointer to a type-of Fn

    fn = &MyFn;           // Point to one function
    fn(3);            // Call it
    fn = &Class::Static; // Point to the other function
    fn(4);            // Call it
} // main()
```

“”。 ...

```
typedef int Fn(int); // Fn is a type-of function that accepts an int and returns an int
```

```

class Class {
public:
    // Note that A() is of type 'Fn'
    int A(int a) { return 2*a; }
    // Note that B() is of type 'Fn'
    int B(int b) { return 3*b; }
}; // Class

int main() {
    Class c;           // Need a Class instance to play with
    Class *p = &c;    // Need a Class pointer to play with

    Fn Class::*fn;   // fn is a pointer to a type-of Fn within Class

    fn = &Class::A;  // fn now points to A within any Class
    (c.*fn)(5);     // Pass 5 to c's function A (via fn)
    fn = &Class::B;  // fn now points to B within any Class
    (p->*fn)(6);   // Pass 6 to c's (via p) function B (via fn)
} // main()

```

.\*>\*

class " " . class...

```

class Class {
public:
    int x, y, z;
    char m, n, o;
}; // Class

int x; // Global variable

int main() {
    Class c;           // Need a Class instance to play with
    Class *p = &c;    // Need a Class pointer to play with

    int *p_i;          // Pointer to an int

    p_i = &x;          // Now pointing to x
    p_i = &c.x;        // Now pointing to c's x

    int Class::*p_C_i; // Pointer to an int within Class

    p_C_i = &Class::x; // Point to x within any Class
    int i = c.*p_C_i; // Use p_c_i to fetch x from c's instance
    p_C_i = &Class::y; // Point to y within any Class
    i = c.*p_C_i;     // Use p_c_i to fetch y from c's instance

    p_C_i = &Class::m; // ERROR! m is a char, not an int!

    char Class::*p_C_c = &Class::m; // That's better...
} // main()

```

- int Class::\*ptr; ◦
- .\*.◦
- ->\*>-◦

static C / C ++

- class ;
- public protectedprivate◦

staticclass

```
class Class {  
public:  
    static int i;  
}; // Class  
  
int Class::i = 1; // Define the value of i (and where it's stored!)  
  
int j = 2; // Just another global variable  
  
int main() {  
    int k = 3; // Local variable  
  
    int *p;  
  
    p = &k; // Point to k  
    *p = 2; // Modify it  
    p = &j; // Point to j  
    *p = 3; // Modify it  
    p = &Class::i; // Point to Class::i  
    *p = 4; // Modify it  
} // main()
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2130/>

# 81:

◦ ◦ ◦

"\* " & " -> \* -> &

- <> \* <>;
- <> \* <> = <>;
- <> \* <> = <>;
- int \* foo; //
- int \* bar = myIntVar;
- long \* bar [2];
- long \* bar [] = {myLongVar1 myLongVar2}; //long \* bar [2]

◦

```
int* a, b, c; //Only a is a pointer, the others are regular ints.  
int* a, *b, *c; //These are three pointers!  
int *foo[2]; //Both *foo[0] and *foo[1] are pointers.
```

## Examples

### C ++ 11

C ++ 11 nullptr ◦ nullptr NULL ◦

```
*int *pointer_to_int; ◦  
int * ◦ int ◦  
  
// Declare a struct type `big_struct` that contains  
// three long long ints.  
typedef struct {  
    long long int fool;  
    long long int foo2;  
    long long int foo3;  
} big_struct;  
  
// Create a variable `bar` of type `big_struct`  
big_struct bar;  
// Create a variable `p_bar` of type `pointer to big_struct`.  
// Initialize it to `nullptr` (a null pointer).  
big_struct *p_bar0 = nullptr;  
  
// Print the size of `bar`  
std::cout << "sizeof(bar) = " << sizeof(bar) << std::endl;  
// Print the size of `p_bar`.  
std::cout << "sizeof(p_bar0) = " << sizeof(p_bar0) << std::endl;
```

```
/* Produces:  
 sizeof(bar) = 24  
 sizeof(p_bar0) = 8  
 */
```

;

• `nullptr` • `nullptr`

### operator &◦ & ◦

```
// Declare a struct type `big_struct` that contains  
// three long long ints.  
typedef struct {  
    long long int foo1;  
    long long int foo2;  
    long long int foo3;  
} big_struct;  
  
// Create a variable `bar` of type `big_struct`  
big_struct bar;  
// Create a variable `p_bar` of type `pointer to big_struct`.  
// Initialize it to `nullptr` (a null pointer).  
big_struct *p_bar0 = nullptr;  
  
// Print the size of `bar`  
std::cout << "sizeof(bar) = " << sizeof(bar) << std::endl;  
// Print the size of `p_bar`.  
std::cout << "sizeof(p_bar0) = " << sizeof(p_bar0) << std::endl;  
  
/* Produces:  
 sizeof(bar) = 24  
 sizeof(p_bar0) = 8  
 */
```

- ;
- `nullptr`
- `◦`

• `& *◦` • `const` •

```
// Declare a struct type `big_struct` that contains  
// three long long ints.  
typedef struct {  
    long long int foo1;  
    long long int foo2;  
    long long int foo3;  
} big_struct;  
  
// Create a variable `bar` of type `big_struct`  
big_struct bar;  
// Create a variable `p_bar` of type `pointer to big_struct`.  
// Initialize it to `nullptr` (a null pointer).
```

```

big_struct *p_bar0 = nullptr;

// Print the size of `bar`
std::cout << "sizeof(bar) = " << sizeof(bar) << std::endl;
// Print the size of `p_bar`.
std::cout << "sizeof(p_bar0) = " << sizeof(p_bar0) << std::endl;

/* Produces:
   sizeof(bar) = 24
   sizeof(p_bar0) = 8
*/

```

\*.->

```

// Declare a struct type `big_struct` that contains
// three long long ints.
typedef struct {
    long long int foo1;
    long long int foo2;
    long long int foo3;
} big_struct;

// Create a variable `bar` of type `big_struct`
big_struct bar;
// Create a variable `p_bar` of type `pointer to big_struct`.
// Initialize it to `nullptr` (a null pointer).
big_struct *p_bar0 = nullptr;

// Print the size of `bar`
std::cout << "sizeof(bar) = " << sizeof(bar) << std::endl;
// Print the size of `p_bar`.
std::cout << "sizeof(p_bar0) = " << sizeof(p_bar0) << std::endl;

/* Produces:
   sizeof(bar) = 24
   sizeof(p_bar0) = 8
*/

```

◦ ◦

```

// Declare a struct type `big_struct` that contains
// three long long ints.
typedef struct {
    long long int foo1;
    long long int foo2;
    long long int foo3;
} big_struct;

// Create a variable `bar` of type `big_struct`
big_struct bar;
// Create a variable `p_bar` of type `pointer to big_struct`.
// Initialize it to `nullptr` (a null pointer).
big_struct *p_bar0 = nullptr;

// Print the size of `bar`
std::cout << "sizeof(bar) = " << sizeof(bar) << std::endl;
// Print the size of `p_bar`.
std::cout << "sizeof(p_bar0) = " << sizeof(p_bar0) << std::endl;

```

```
/* Produces:  
 sizeof(bar) = 24  
 sizeof(p_bar0) = 8  
 */
```

g++clang++

```
// Declare a struct type `big_struct` that contains  
// three long long ints.  
typedef struct {  
    long long int fool;  
    long long int foo2;  
    long long int foo3;  
} big_struct;  
  
// Create a variable `bar` of type `big_struct`  
big_struct bar;  
// Create a variable `p_bar` of type `pointer to big_struct`.  
// Initialize it to `nullptr` (a null pointer).  
big_struct *p_bar0 = nullptr;  
  
// Print the size of `bar`  
std::cout << "sizeof(bar) = " << sizeof(bar) << std::endl;  
// Print the size of `p_bar`.  
std::cout << "sizeof(p_bar0) = " << sizeof(p_bar0) << std::endl;  
  
/* Produces:  
 sizeof(bar) = 24  
 sizeof(p_bar0) = 8  
 */
```

## null

```
// Declare a struct type `big_struct` that contains  
// three long long ints.  
typedef struct {  
    long long int fool;  
    long long int foo2;  
    long long int foo3;  
} big_struct;  
  
// Create a variable `bar` of type `big_struct`  
big_struct bar;  
// Create a variable `p_bar` of type `pointer to big_struct`.  
// Initialize it to `nullptr` (a null pointer).  
big_struct *p_bar0 = nullptr;  
  
// Print the size of `bar`  
std::cout << "sizeof(bar) = " << sizeof(bar) << std::endl;  
// Print the size of `p_bar`.  
std::cout << "sizeof(p_bar0) = " << sizeof(p_bar0) << std::endl;  
  
/* Produces:  
 sizeof(bar) = 24  
 sizeof(p_bar0) = 8  
 */
```

## Address-of operator。 Contents-of Dereference\*。

```
int var = 20;
int *ptr;
ptr = &var;

cout << var << endl;
//Outputs 20 (The value of var)

cout << ptr << endl;
//Outputs 0x234f119 (var's memory location)

cout << *ptr << endl;
//Outputs 20(The value of the variable stored in the pointer ptr
```

### \*。 dereference。 。



◦ ◦ ◦

◦ void◦

```
char* str = new char[10]; // str = 0x010
++str;                  // str = 0x011 in this case sizeof(char) = 1 byte

int* arr = new int[10];   // arr = 0x00100
++arr;                  // arr = 0x00104 if sizeof(int) = 4 bytes

void* ptr = (void*)new char[10];
++ptr;      // void is incomplete.
```

◦ ◦

◦

1◦



;1◦ /◦

```
char* str = new char[10]; // str = 0x010
++str;                  // str = 0x011 in this case sizeof(char) = 1 byte

int* arr = new int[10];   // arr = 0x00100
++arr;                  // arr = 0x00104 if sizeof(int) = 4 bytes

void* ptr = (void*)new char[10];
++ptr;      // void is incomplete.
```

◦ ;◦

PQ PiQjP - Qi - j◦ std::ptrdiff\_t <cstddef> ◦

```
char* str = new char[10]; // str = 0x010
++str;                  // str = 0x011 in this case sizeof(char) = 1 byte

int* arr = new int[10];   // arr = 0x00100
++arr;                  // arr = 0x00104 if sizeof(int) = 4 bytes

void* ptr = (void*)new char[10];
++ptr;      // void is incomplete.
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3056/>

```
std::sort algorithm°
```

## Examples

std::sort

### C++ 11

```
#include <vector>
#include <algorithm>
#include <functional>

std::vector<int> v = {5,1,2,4,3};

//sort in ascending order (1,2,3,4,5)
std::sort(v.begin(), v.end(), std::less<int>());

// Or just:
std::sort(v.begin(), v.end());

//sort in descending order (5,4,3,2,1)
std::sort(v.begin(), v.end(), std::greater<int>());

//Or just:
std::sort(v.rbegin(), v.rend());
```

### C++ 14

### C++ 14

```
#include <vector>
#include <algorithm>
#include <functional>

std::vector<int> v = {5,1,2,4,3};

//sort in ascending order (1,2,3,4,5)
std::sort(v.begin(), v.end(), std::less<int>());

// Or just:
std::sort(v.begin(), v.end());

//sort in descending order (5,4,3,2,1)
std::sort(v.begin(), v.end(), std::greater<int>());

//Or just:
std::sort(v.rbegin(), v.rend());
```

std::sort operator< **ON** std::sort bool bool ° °

sort°

```

// Include sequence containers
#include <vector>
#include <deque>
#include <list>

// Insert sorting algorithm
#include <algorithm>

class Base {
public:

    // Constructor that set variable to the value of v
    Base(int v) : variable(v) {
    }

    // Use variable to provide total order operator less
    // `this` always represents the left-hand side of the compare.
    bool operator<(const Base &b) const {
        return this->variable < b.variable;
    }

    int variable;
};

int main() {
    std::vector <Base> vector;
    std::deque <Base> deque;
    std::list <Base> list;

    // Create 2 elements to sort
    Base a(10);
    Base b(5);

    // Insert them into backs of containers
    vector.push_back(a);
    vector.push_back(b);

    deque.push_back(a);
    deque.push_back(b);

    list.push_back(a);
    list.push_back(b);

    // Now sort data using operator<(const Base &b) function
    std::sort(vector.begin(), vector.end());
    std::sort(deque.begin(), deque.end());
    // List must be sorted differently due to its design
    list.sort();

    return 0;
}

```

```

// Include sequence containers
#include <vector>
#include <deque>
#include <list>

// Insert sorting algorithm
#include <algorithm>

```

```

class Base {
public:

    // Constructor that set variable to the value of v
    Base(int v) : variable(v) {
    }

    int variable;
};

bool compare(const Base &a, const Base &b) {
    return a.variable < b.variable;
}

int main() {
    std::vector <Base> vector;
    std::deque <Base> deque;
    std::list <Base> list;

    // Create 2 elements to sort
    Base a(10);
    Base b(5);

    // Insert them into backs of containers
    vector.push_back(a);
    vector.push_back(b);

    deque.push_back(a);
    deque.push_back(b);

    list.push_back(a);
    list.push_back(b);

    // Now sort data using comparing function
    std::sort(vector.begin(), vector.end(), compare);
    std::sort(deque.begin(), deque.end(), compare);
    list.sort(compare);

    return 0;
}

```

## lambdaC ++ 11

### C ++ 11

```

// Include sequence containers
#include <vector>
#include <deque>
#include <list>
#include <array>
#include <forward_list>

// Include sorting algorithm
#include <algorithm>

class Base {
public:

    // Constructor that set variable to the value of v

```

```

Base(int v): variable(v) {
}

    int variable;
};

int main() {
    // Create 2 elements to sort
    Base a(10);
    Base b(5);

    // We're using C++11, so let's use initializer lists to insert items.
    std::vector <Base> vector = {a, b};
    std::deque <Base> deque = {a, b};
    std::list <Base> list = {a, b};
    std::array <Base, 2> array = {a, b};
    std::forward_list<Base> flist = {a, b};

    // We can sort data using an inline lambda expression
    std::sort(std::begin(vector), std::end(vector),
              [] (const Base &a, const Base &b) { return a.variable < b.variable; });

    // We can also pass a lambda object as the comparator
    // and reuse the lambda multiple times
    auto compare = [] (const Base &a, const Base &b) {
        return a.variable < b.variable;};
    std::sort(std::begin(deque), std::end(deque), compare);
    std::sort(std::begin(array), std::end(array), compare);
    list.sort(compare);
    flist.sort(compare);

    return 0;
}

```

std::sort algorithm。 std::sort; std::sort 。

◦ ◦

std::sortstd::sort

## C++ 11

```

#include <vector>
#include <algorithm>

std::vector<int> MyVector = {3, 1, 2}

//Default comparison of <
std::sort(MyVector.begin(), MyVector.end());

```

std::sort。 std::liststd::forward\_list C++ 11std::sort。 sort。

## C++ 11

```

#include <vector>
#include <algorithm>

```

```
std::vector<int> MyVector = {3, 1, 2}

//Default comparison of <
std::sort(MyVector.begin(), MyVector.end());
```

sort. listforward\_list

```
#include <vector>
#include <algorithm>

std::vector<int> MyVector = {3, 1, 2}

//Default comparison of <
std::sort(MyVector.begin(), MyVector.end());
```

## std :: map

◦ std::string ◦

```
#include <iostream>
#include <utility>
#include <map>

int main()
{
    std::map<double, std::string> sorted_map;
    // Sort the names of the planets according to their size
    sorted_map.insert(std::make_pair(0.3829, "Mercury"));
    sorted_map.insert(std::make_pair(0.9499, "Venus"));
    sorted_map.insert(std::make_pair(1, "Earth"));
    sorted_map.insert(std::make_pair(0.532, "Mars"));
    sorted_map.insert(std::make_pair(10.97, "Jupiter"));
    sorted_map.insert(std::make_pair(9.14, "Saturn"));
    sorted_map.insert(std::make_pair(3.981, "Uranus"));
    sorted_map.insert(std::make_pair(3.865, "Neptune"));

    for (auto const& entry: sorted_map)
    {
        std::cout << entry.second << " (" << entry.first << " of Earth's radius)" << '\n';
    }
}
```

```
#include <iostream>
#include <utility>
#include <map>

int main()
{
    std::map<double, std::string> sorted_map;
    // Sort the names of the planets according to their size
    sorted_map.insert(std::make_pair(0.3829, "Mercury"));
    sorted_map.insert(std::make_pair(0.9499, "Venus"));
    sorted_map.insert(std::make_pair(1, "Earth"));
    sorted_map.insert(std::make_pair(0.532, "Mars"));
    sorted_map.insert(std::make_pair(10.97, "Jupiter"));
    sorted_map.insert(std::make_pair(9.14, "Saturn"));
    sorted_map.insert(std::make_pair(3.981, "Uranus"));
```

```
sorted_map.insert(std::make_pair(3.865, "Neptune"));

for (auto const& entry: sorted_map)
{
    std::cout << entry.second << " (" << entry.first << " of Earth's radius)" << '\n';
}

}
```

multimapmap °

std::greater<>

```
#include <iostream>
#include <utility>
#include <map>

int main()
{
    std::map<double, std::string> sorted_map;
    // Sort the names of the planets according to their size
    sorted_map.insert(std::make_pair(0.3829, "Mercury"));
    sorted_map.insert(std::make_pair(0.9499, "Venus"));
    sorted_map.insert(std::make_pair(1, "Earth"));
    sorted_map.insert(std::make_pair(0.532, "Mars"));
    sorted_map.insert(std::make_pair(10.97, "Jupiter"));
    sorted_map.insert(std::make_pair(9.14, "Saturn"));
    sorted_map.insert(std::make_pair(3.981, "Uranus"));
    sorted_map.insert(std::make_pair(3.865, "Neptune"));

    for (auto const& entry: sorted_map)
    {
        std::cout << entry.second << " (" << entry.first << " of Earth's radius)" << '\n';
    }
}
```

```
#include <iostream>
#include <utility>
#include <map>

int main()
{
    std::map<double, std::string> sorted_map;
    // Sort the names of the planets according to their size
    sorted_map.insert(std::make_pair(0.3829, "Mercury"));
    sorted_map.insert(std::make_pair(0.9499, "Venus"));
    sorted_map.insert(std::make_pair(1, "Earth"));
    sorted_map.insert(std::make_pair(0.532, "Mars"));
    sorted_map.insert(std::make_pair(10.97, "Jupiter"));
    sorted_map.insert(std::make_pair(9.14, "Saturn"));
    sorted_map.insert(std::make_pair(3.981, "Uranus"));
    sorted_map.insert(std::make_pair(3.865, "Neptune"));

    for (auto const& entry: sorted_map)
    {
        std::cout << entry.second << " (" << entry.first << " of Earth's radius)" << '\n';
    }
}
```

sortsort . C°

## C ++ 11

```
int arr1[] = {36, 24, 42, 60, 59};

// sort numbers in ascending order
sort(std::begin(arr1), std::end(arr1));

// sort numbers in descending order
sort(std::begin(arr1), std::end(arr1), std::greater<int>());
```

## C ++ 11“”

## C ++ 11

```
int arr1[] = {36, 24, 42, 60, 59};

// sort numbers in ascending order
sort(std::begin(arr1), std::end(arr1));

// sort numbers in descending order
sort(std::begin(arr1), std::end(arr1), std::greater<int>());
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1675/>

83:

## Examples

1

- 7237498123.
  - 237498123237499123.
  - 23749912320249472.

C++14 Simple Quotation Mark ' .. .

C ++ 14

```
long long decn = 1'000'000'0001l;
long long hexn = 0xFFFF'FFFF1l;
long long octn = 00'23'001l;
long long binn = 0b1010'0011ll;
```

1

- 1048576 1'048'576 0x100000 0x10'00000'004'000'000.
  - 1.602'176'565e-191.602176565e-19.

8

C ++ 14

```
long long decn = 1'000'000'0001;  
long long hexn = 0xFFFF'FFFF11;  
long long octn = 00'23'0011;  
long long binn = 0b1010'001111;
```

user-defined

C++ 14

```
long long decn = 1'000'000'0001;
long long hexn = 0xFFFF'FFFF1l;
long long octn = 00'23'001l;
long long binn = 0b1010'00111l;
```

<https://riptutorial.com/zh-TW/cplusplus/topic/10595/>

◦ ◦

◦

## Examples

◦

```
#include <stddef.h>      // size_t, ptrdiff_t

//----- Machinery:

using Size = ptrdiff_t;

template< class Item, size_t n >
constexpr auto n_items( Item (&) [n] ) noexcept
    -> Size
{ return n; }

//----- Usage:

#include <iostream>
using namespace std;
auto main()
    -> int
{
    int const a[]     = {3, 1, 4, 1, 5, 9, 2, 6, 5, 4};
    Size const n      = n_items( a );
    int         b[n]   = {};           // An array of the same size as a.

    (void) b;
    cout << "Size = " << n << "\n";
}
```

**C idiom** `sizeof(a)/sizeof(a[0])`

**C++ 11**

**C++ 11**

```
#include <stddef.h>      // size_t, ptrdiff_t

//----- Machinery:

using Size = ptrdiff_t;

template< class Item, size_t n >
constexpr auto n_items( Item (&) [n] ) noexcept
    -> Size
{ return n; }
```

```

//----- Usage:

#include <iostream>
using namespace std;
auto main()
    -> int
{
    int const a[]      = {3, 1, 4, 1, 5, 9, 2, 6, 5, 4};
    Size const n       = n_items( a );
    int         b[n]    = {};           // An array of the same size as a.

    (void) b;
    cout << "Size = " << n << "\n";
}

```

```

#include <stddef.h>      // size_t, ptrdiff_t

//----- Machinery:

using Size = ptrdiff_t;

template< class Item, size_t n >
constexpr auto n_items( Item (&) [n] ) noexcept
    -> Size
{ return n; }

//----- Usage:

#include <iostream>
using namespace std;
auto main()
    -> int
{
    int const a[]      = {3, 1, 4, 1, 5, 9, 2, 6, 5, 4};
    Size const n       = n_items( a );
    int         b[n]    = {};           // An array of the same size as a.

    (void) b;
    cout << "Size = " << n << "\n";
}

```

## C++17 `size_t`

### C++17 `std::size`

```

// Example of raw dynamic size array. It's generally better to use std::vector.
#include <algorithm>          // std::sort
#include <iostream>
using namespace std;

auto int_from( istream& in ) -> int { int x; in >> x; return x; }

auto main()
    -> int
{

```

```

cout << "Sorting n integers provided by you.\n";
cout << "n? ";
int const n = int_from( cin );
int* a = new int[n]; // ← Allocation of array of n items.

for( int i = 1; i <= n; ++i )
{
    cout << "The #" << i << " number, please: ";
    a[i-1] = int_from( cin );
}

sort( a, a + n );
for( int i = 0; i < n; ++i ) { cout << a[i] << ' '; }
cout << '\n';

delete[] a;
}

```

## T a[n];nC99 VLA。 C ++VLA。 new[] -expression

```

// Example of raw dynamic size array. It's generally better to use std::vector.
#include <algorithm> // std::sort
#include <iostream>
using namespace std;

auto int_from( istream& in ) -> int { int x; in >> x; return x; }

auto main()
-> int
{
    cout << "Sorting n integers provided by you.\n";
    cout << "n? ";
    int const n = int_from( cin );
    int* a = new int[n]; // ← Allocation of array of n items.

    for( int i = 1; i <= n; ++i )
    {
        cout << "The #" << i << " number, please: ";
        a[i-1] = int_from( cin );
    }

    sort( a, a + n );
    for( int i = 0; i < n; ++i ) { cout << a[i] << ' '; }
    cout << '\n';

    delete[] a;
}

```

## ...delete[] -expression

```

// Example of raw dynamic size array. It's generally better to use std::vector.
#include <algorithm> // std::sort
#include <iostream>
using namespace std;

auto int_from( istream& in ) -> int { int x; in >> x; return x; }

auto main()
-> int

```

```

{
    cout << "Sorting n integers provided by you.\n";
    cout << "n? ";
    int const n = int_from( cin );
    int* a = new int[n];           // ← Allocation of array of n items.

    for( int i = 1; i <= n; ++i )
    {
        cout << "The #" << i << " number, please: ";
        a[i-1] = int_from( cin );
    }

    sort( a, a + n );
    for( int i = 0; i < n; ++i ) { cout << a[i] << ' '; }
    cout << '\n';

    delete[] a;
}

() new int[n] () ° °

delete[]new[] ° std::unique_ptr° std::vector std::vector°

```

## std :: vector°

```

// Example of std::vector as an expanding dynamic size array.
#include <algorithm>           // std::sort
#include <iostream>
#include <vector>                // std::vector
using namespace std;

int int_from( std::istream& in ) { int x = 0; in >> x; return x; }

int main()
{
    cout << "Sorting integers provided by you.\n";
    cout << "You can indicate EOF via F6 in Windows or Ctrl+D in Unix-land.\n";
    vector<int> a;           // ← Zero size by default.

    while( cin )
    {
        cout << "One number, please, or indicate EOF: ";
        int const x = int_from( cin );
        if( !cin.fail() ) { a.push_back( x ); } // Expands as necessary.
    }

    sort( a.begin(), a.end() );
    int const n = a.size();
    for( int i = 0; i < n; ++i ) { cout << a[i] << ' '; }
    cout << '\n';
}

```

std::vector° &v[0]v.data() API° push\_backvector°

$n$  push\_back $O(n)$  ° “”°

- $1n = 17$ push\_back $1 + 2 + 4 + 8 + 16 = 312 \times n = 34$ ◦  $2 \times n$ ◦

```
vector< >
```

## 2D。

```
// A fixed size raw array matrix (that is, a 2D raw array).
#include <iostream>
#include <iomanip>
using namespace std;

auto main() -> int
{
    int const n_rows = 3;
    int const n_cols = 7;
    int const m[n_rows][n_cols] =           // A raw array matrix.
    {
        { 1, 2, 3, 4, 5, 6, 7 },
        { 8, 9, 10, 11, 12, 13, 14 },
        { 15, 16, 17, 18, 19, 20, 21 }
    };

    for( int y = 0; y < n_rows; ++y )
    {
        for( int x = 0; x < n_cols; ++x )
        {
            cout << setw( 4 ) << m[y][x];      // Note: do NOT use m[y,x]!
        }
        cout << '\n';
    }
}
```

```
// A fixed size raw array matrix (that is, a 2D raw array).
#include <iostream>
#include <iomanip>
using namespace std;

auto main() -> int
{
    int const n_rows = 3;
    int const n_cols = 7;
    int const m[n_rows][n_cols] =           // A raw array matrix.
    {
        { 1, 2, 3, 4, 5, 6, 7 },
        { 8, 9, 10, 11, 12, 13, 14 },
        { 15, 16, 17, 18, 19, 20, 21 }
    };

    for( int y = 0; y < n_rows; ++y )
    {
        for( int x = 0; x < n_cols; ++x )
        {
            cout << setw( 4 ) << m[y][x];      // Note: do NOT use m[y,x]!
        }
        cout << '\n';
    }
}
```

C++ [ i ] [ y ] my y [ x ] y x .

“”。

C++。 m[y][x] operator[]。 m(x,y)m.at(x,y)m.item(x,y)。

std :: vector。

C++ 14C++。 Boost。

BoostC++

```
vector<vector<int>> m( 3, vector<int>( 7 ) );
```

...vectorstd::vector。 nn3.m[y][x]。

x y

```
vector<vector<int>> m( 3, vector<int>( 7 ) );
```

```
vector<vector<int>> m( 3, vector<int>( 7 ) );
```

C++。

operator()。

◦ [ ]◦

typ int arrayOfInts[5]

```
int arrayOfInts[5];
```

```
int arrayOfInts[5];
```

◦ ◦ 5◦

```
int arrayOfInts[5];
```

◦ ◦ 5◦

```
int arrayOfInts[5];
```

◦

```
int arrayOfInts[5];
```

0◦

```
int arrayOfInts[5];
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3017/>

# 85: I / O.

## C ++I / O.

```
std::istream
```

```
std::ostream
```

```
std::streambuf
```

```
operator>>
```

```
operator<<
```

**Streams** std::locale

[<iostream> Library](#)

## Examples

3 ifstream ofstream fstream

```
std::ifstream ifs("foo.txt"); // ifstream: Opens file "foo.txt" for reading only.  
std::ofstream ofs("foo.txt"); // ofstream: Opens file "foo.txt" for writing only.  
std::fstream iofs("foo.txt"); // fstream: Opens file "foo.txt" for reading and writing.
```

open()

```
std::ifstream ifs("foo.txt"); // ifstream: Opens file "foo.txt" for reading only.  
std::ofstream ofs("foo.txt"); // ofstream: Opens file "foo.txt" for writing only.  
std::fstream iofs("foo.txt"); // fstream: Opens file "foo.txt" for reading and writing.
```

◦ ...

```
std::ifstream ifs("foo.txt"); // ifstream: Opens file "foo.txt" for reading only.  
std::ofstream ofs("foo.txt"); // ofstream: Opens file "foo.txt" for writing only.  
std::fstream iofs("foo.txt"); // fstream: Opens file "foo.txt" for reading and writing.
```

## Windows

```
std::ifstream ifs("foo.txt"); // ifstream: Opens file "foo.txt" for reading only.  
std::ofstream ofs("foo.txt"); // ofstream: Opens file "foo.txt" for writing only.
```

```
std::fstream iofs("foo.txt"); // fstream: Opens file "foo.txt" for reading and writing.
```

## C++ 11

```
std::ifstream ifs("foo.txt"); // ifstream: Opens file "foo.txt" for reading only.  
std::ofstream ofs("foo.txt"); // ofstream: Opens file "foo.txt" for writing only.  
std::fstream iofs("foo.txt"); // fstream: Opens file "foo.txt" for reading and writing.
```

```
std::ifstream ifs("foo.txt"); // ifstream: Opens file "foo.txt" for reading only.  
std::ofstream ofs("foo.txt"); // ofstream: Opens file "foo.txt" for writing only.  
std::fstream iofs("foo.txt"); // fstream: Opens file "foo.txt" for reading and writing.
```

## C++ 11

### WindowsASCII

```
std::ifstream ifs("foo.txt"); // ifstream: Opens file "foo.txt" for reading only.  
std::ofstream ofs("foo.txt"); // ofstream: Opens file "foo.txt" for writing only.  
std::fstream iofs("foo.txt"); // fstream: Opens file "foo.txt" for reading and writing.
```

◦

>> ◦ *foo.txt*

```
John Doe 25 4 6 1987  
Jane Doe 15 5 24 1976
```

```
John Doe 25 4 6 1987  
Jane Doe 15 5 24 1976
```

>>

- \n ◦
- ◦

*foo.txt*

```
John Doe 25 4 6 1987  
Jane Doe 15 5 24 1976
```

>> ◦ while◦ bool◦ bool()true◦ bool()false◦ while◦

```
John Doe 25 4 6 1987  
Jane Doe 15 5 24 1976
```

```
string<operator>
```

```
getline()
```

```
John Doe 25 4 6 1987  
Jane Doe 15 5 24 1976
```

```
read()
```

```
John Doe 25 4 6 1987  
Jane Doe 15 5 24 1976
```

```
failbit | fail()
```

```
John Doe 25 4 6 1987  
Jane Doe 15 5 24 1976
```

```
o ofstream <<
```

```
std::ofstream os("foo.txt");  
if(os.is_open()) {  
    os << "Hello World!";  
}
```

```
<< write()
```

```
std::ofstream os("foo.txt");  
if(os.is_open()) {  
    os << "Hello World!";  
}
```

```
badbit | bad()
```

```
std::ofstream os("foo.txt");  
if(os.is_open()) {  
    os << "Hello World!";  
}
```

```
o o
```

```
std::ios<operator>
```

```
open()
```

```
std::ofstream os("foo.txt", std::ios::out | std::ios::trunc);  
std::ifstream is;  
is.open("foo.txt", std::ios::in | std::ios::binary);
```

```
ios::outios::inios::out iostream<operator>
```

- ifstream - in
- ofstream out
- fstream - inout

app		o	
binary			o
in		o	
out			o
trunc		o	
ate			o

binary /; '\n' o

Windows CRLF "\r\n" o

"\n" => "\r\n"  
 "\r\n" => "\n"

C++ o o {}

```
std::string const prepared_data = prepare_data();
{
    // Open a file for writing.
    std::ofstream output("foo.txt");

    // Write data.
    output << prepared_data;
} // The ofstream will go out of scope here.
// Its destructor will take care of closing the file properly.
```

fstreamclose()

```
std::string const prepared_data = prepare_data();
{
    // Open a file for writing.
    std::ofstream output("foo.txt");

    // Write data.
    output << prepared_data;
} // The ofstream will go out of scope here.
// Its destructor will take care of closing the file properly.
```

o o oder o flush() std::flush

```
std::ofstream os("foo.txt");
os << "Hello World!" << std::flush;
```

```

char data[3] = "Foo";
os.write(data, 3);
os.flush();

std::endl

std::ofstream os("foo.txt");
os << "Hello World!" << std::flush;

char data[3] = "Foo";
os.write(data, 3);
os.flush();

```

◦◦ I/O◦

## ASCIIstd :: string

```

std::ifstream f("file.txt");

if (f)
{
    std::stringstream buffer;
    buffer << f.rdbuf();
    f.close();

    // The content of "file.txt" is available in the string `buffer.str()`
}

```

`rdbuf()` streambuf`stringstream::operator<<` member`buffer`◦

## Scott MeyersEffective STL

```

std::ifstream f("file.txt");

if (f)
{
    std::stringstream buffer;
    buffer << f.rdbuf();
    f.close();

    // The content of "file.txt" is available in the string `buffer.str()`
}

```

STL◦

◦

```

std::ifstream f("file.txt");

if (f)
{

```

```
    std::stringstream buffer;
    buffer << f.rdbuf();
    f.close();

    // The content of "file.txt" is available in the string `buffer.str()`
}
```

o

std::string operator>>

```
std::ifstream file("file3.txt");

std::vector<std::string> v;

std::string s;
while(file >> s) // keep reading until we run out
{
    v.push_back(s);
}
```

operator>>”。 std::istream\_iterator”。

```
std::ifstream file("file3.txt");

std::vector<std::string> v;

std::string s;
while(file >> s) // keep reading until we run out
{
    v.push_back(s);
}
```

std::istream\_iterator

```
std::ifstream file("file3.txt");

std::vector<std::string> v;

std::string s;
while(file >> s) // keep reading until we run out
{
    v.push_back(s);
}
```

`struct`。

C++ 11

```
struct info_type
{
    std::string name;
    int age;
    float height;
```

```

// we define an overload of operator>> as a friend function which
// gives in privileged access to private data members
friend std::istream& operator>>(std::istream& is, info_type& info)
{
    // skip whitespace
    is >> std::ws;
    std::getline(is, info.name);
    is >> info.age;
    is >> info.height;
    return is;
}
};

void func4()
{
    auto file = std::ifstream("file4.txt");

    std::vector<info_type> v;

    for(info_type info; file >> info;) // keep reading until we run out
    {
        // we only get here if the read succeeded
        v.push_back(info);
    }

    for(auto const& info: v)
    {
        std::cout << " name: " << info.name << '\n';
        std::cout << " age: " << info.age << " years" << '\n';
        std::cout << "height: " << info.height << "lbs" << '\n';
        std::cout << '\n';
    }
}
}

```

## file4.txt

```

struct info_type
{
    std::string name;
    int age;
    float height;

    // we define an overload of operator>> as a friend function which
    // gives in privileged access to private data members
    friend std::istream& operator>>(std::istream& is, info_type& info)
    {
        // skip whitespace
        is >> std::ws;
        std::getline(is, info.name);
        is >> info.age;
        is >> info.height;
        return is;
    }
};

void func4()
{
    auto file = std::ifstream("file4.txt");

```

```

std::vector<info_type> v;

for(info_type info; file >> info;) // keep reading until we run out
{
    // we only get here if the read succeeded
    v.push_back(info);
}

for(auto const& info: v)
{
    std::cout << " name: " << info.name << '\n';
    std::cout << " age: " << info.age << " years" << '\n';
    std::cout << "height: " << info.height << "lbs" << '\n';
    std::cout << '\n';
}
}

```

```

struct info_type
{
    std::string name;
    int age;
    float height;

    // we define an overload of operator>> as a friend function which
    // gives in privileged access to private data members
    friend std::istream& operator>>(std::istream& is, info_type& info)
    {
        // skip whitespace
        is >> std::ws;
        std::getline(is, info.name);
        is >> info.age;
        is >> info.height;
        return is;
    }
};

void func4()
{
    auto file = std::ifstream("file4.txt");

    std::vector<info_type> v;

    for(info_type info; file >> info;) // keep reading until we run out
    {
        // we only get here if the read succeeded
        v.push_back(info);
    }

    for(auto const& info: v)
    {
        std::cout << " name: " << info.name << '\n';
        std::cout << " age: " << info.age << " years" << '\n';
        std::cout << "height: " << info.height << "lbs" << '\n';
        std::cout << '\n';
    }
}

```

```

std::ifstream src("source_filename", std::ios::binary);
std::ofstream dst("dest_filename", std::ios::binary);

```

```
dst << src.rdbuf();
```

## C++ 17

### C++ 17<filesystem>copy\_file

```
std::ifstream src("source_filename", std::ios::binary);
std::ofstream dst("dest_filename",   std::ios::binary);
dst << src.rdbuf();
```

boost.filesystem C++ 17|ISO C++。

eoftrue ◦ ◦

```
while (!f.eof())
{
    // Everything is OK

    f >> buffer;

    // What if *only* now the eof / fail bit is set?

    /* Use `buffer` */
}
```

```
while (!f.eof())
{
    // Everything is OK

    f >> buffer;

    // What if *only* now the eof / fail bit is set?

    /* Use `buffer` */
}
```

```
while (!f.eof())
{
    // Everything is OK

    f >> buffer;

    // What if *only* now the eof / fail bit is set?

    /* Use `buffer` */
}
```

◦

- std::ws
- std::basic\_ios::fail true

std::localestd::basic\_ios::imbue()

-

◦  
• ◦

## UTF-8。UTF-8BOM;

```
#include <iostream>
#include <fstream>
#include <locale>

int main()
{
    std::cout << "User-preferred locale setting is "
        << std::locale("").name().c_str() << std::endl;

    // Write a floating-point value using the user's preferred locale.
    std::ofstream ofs1;
    ofs1.imbue(std::locale(""));
    ofs1.open("file1.txt");
    ofs1 << 78123.456 << std::endl;

    // Use a specific locale (names are system-dependent)
    std::ofstream ofs2;
    ofs2.imbue(std::locale("en_US.UTF-8"));
    ofs2.open("file2.txt");
    ofs2 << 78123.456 << std::endl;

    // Switch to the classic "C" locale
    std::ofstream ofs3;
    ofs3.imbue(std::locale::classic());
    ofs3.open("file3.txt");
    ofs3 << 78123.456 << std::endl;
}
```

“C”。 “C”

```
#include <iostream>
#include <fstream>
#include <locale>

int main()
{
    std::cout << "User-preferred locale setting is "
        << std::locale("").name().c_str() << std::endl;

    // Write a floating-point value using the user's preferred locale.
    std::ofstream ofs1;
    ofs1.imbue(std::locale(""));
    ofs1.open("file1.txt");
    ofs1 << 78123.456 << std::endl;

    // Use a specific locale (names are system-dependent)
    std::ofstream ofs2;
    ofs2.imbue(std::locale("en_US.UTF-8"));
    ofs2.open("file2.txt");
    ofs2 << 78123.456 << std::endl;
```

```

// Switch to the classic "C" locale
std::ofstream ofs3;
ofs3.imbue(std::locale::classic());
ofs3.open("file3.txt");
ofs3 << 78123.456 << std::endl;
}

#include <iostream>
#include <fstream>
#include <locale>

int main()
{
    std::cout << "User-preferred locale setting is "
        << std::locale("").name().c_str() << std::endl;

    // Write a floating-point value using the user's preferred locale.
    std::ofstream ofs1;
    ofs1.imbue(std::locale(""));
    ofs1.open("file1.txt");
    ofs1 << 78123.456 << std::endl;

    // Use a specific locale (names are system-dependent)
    std::ofstream ofs2;
    ofs2.imbue(std::locale("en_US.UTF-8"));
    ofs2.open("file2.txt");
    ofs2 << 78123.456 << std::endl;

    // Switch to the classic "C" locale
    std::ofstream ofs3;
    ofs3.imbue(std::locale::classic());
    ofs3.open("file3.txt");
    ofs3 << 78123.456 << std::endl;
}

```

◦

I / O. <https://riptutorial.com/zh-TW/cplusplus/topic/496/i---o->

- std::shared\_ptr<ClassType> variableName = std::make\_shared<ClassType>(arg1, arg2, ...);
- std::shared\_ptr<ClassType> variableName (new ClassType(arg1, arg2, ...));
- std::unique\_ptr<ClassType> variableName = std::make\_unique<ClassType>(arg1, arg2, ...); // C++ 14
- std::unique\_ptr<ClassType> variableName (new ClassType(arg1, arg2, ...));

C++ new/delete “”。

◦

“”。

```
#include <memory>
```

## Examples

### std :: shared\_ptr

```
std::shared_ptr< T> std::unique_ptr< T>
```

◦ std::shared\_ptr< T>

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

shared\_ptr< T>

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

secondSharedFoo firstShared< T>

◦ \* →

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

shared\_ptr< Foo>

```
shared_ptr<bad_alloc> shared_ptr<T> shared_ptr<T>(new T(args)) T make_shared<T>(args)  
allocate_shared<T>(alloc, args)。
```

### shared\_ptr[]

## C++ 11 C++ 17

```
make_shared<>。  
newstd::default_deleteshared_ptr<>。
```

### 10

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

```
std::default_deletedelete[]。
```

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

```
make_shared_array<int[10]>10shared_ptr<int>。
```

## C++ 17

### C++ 17 shared\_ptr 。 array-deleter[]

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

---

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

---

```
p2p1Foop2intx。 p1Foop2。
```

---

```
shared_ptrshared_ptr 。 Fooshared_ptr
```

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

## shared\_ptr

```
shared_ptr。 std::move
```

```
// Creation: 'firstShared' is a shared pointer for a new instance of 'Foo'  
std::shared_ptr<Foo> firstShared = std::make_shared<Foo>(/*args*/);
```

## std :: weak\_ptr

```
std::weak_ptrstd::shared_ptr。 。
```

---

```
std::weak_ptr
```

```

#include <memory>
#include <vector>

struct TreeNode {
    std::weak_ptr<TreeNode> parent;
    std::vector< std::shared_ptr<TreeNode> > children;
};

int main() {
    // Create a TreeNode to serve as the root/parent.
    std::shared_ptr<TreeNode> root(new TreeNode);

    // Give the parent 100 child nodes.
    for (size_t i = 0; i < 100; ++i) {
        std::shared_ptr<TreeNode> child(new TreeNode);
        root->children.push_back(child);
        child->parent = root;
    }

    // Reset the root shared pointer, destroying the root object, and
    // subsequently its child nodes.
    root.reset();
}

```

`std::weak_ptr` `parent` `main()` `std::shared_ptr` `children` `。`

### **shared\_ptr** `shared_ptr` `weak_ptr` `。`

```

#include <memory>
#include <vector>

struct TreeNode {
    std::weak_ptr<TreeNode> parent;
    std::vector< std::shared_ptr<TreeNode> > children;
};

int main() {
    // Create a TreeNode to serve as the root/parent.
    std::shared_ptr<TreeNode> root(new TreeNode);

    // Give the parent 100 child nodes.
    for (size_t i = 0; i < 100; ++i) {
        std::shared_ptr<TreeNode> child(new TreeNode);
        root->children.push_back(child);
        child->parent = root;
    }

    // Reset the root shared pointer, destroying the root object, and
    // subsequently its child nodes.
    root.reset();
}

```

---

`std::weak_ptr` `std::weak_ptr` `lock()` `std::shared_ptr`

```

#include <memory>
#include <vector>

```

```

struct TreeNode {
    std::weak_ptr<TreeNode> parent;
    std::vector< std::shared_ptr<TreeNode> > children;
};

int main() {
    // Create a TreeNode to serve as the root/parent.
    std::shared_ptr<TreeNode> root(new TreeNode);

    // Give the parent 100 child nodes.
    for (size_t i = 0; i < 100; ++i) {
        std::shared_ptr<TreeNode> child(new TreeNode);
        root->children.push_back(child);
        child->parent = root;
    }

    // Reset the root shared pointer, destroying the root object, and
    // subsequently its child nodes.
    root.reset();
}

```

## std :: unique\_ptr

### C ++ 11

`std::unique_ptr` `std::shared_ptr``std::unique_ptr`

---

```

// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);

```

### C ++ 11 `std::unique_ptr` C ++ 14 `std::make_unique`

`ptr` `int` `std::unique_ptr`

`std::unique_ptr``std::make_unique`

```

// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);

```

`std::unique_ptr` `int`

---

`std::move``nullptr` `int`

```

// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);

```

`unique_ptr` `int`

---

```

// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);

```

unique\_ptr。 C++ 11 unique\_ptr。

```
// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);
```

```
// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);
```

## C++ 14

### C++ 14 make\_unique。 C++ 11

```
// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);
```

## C++ 11

std::auto\_ptr unique\_ptr std::vector<sup>o</sup> 10 int[] int

```
// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);
```

```
// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);
```

arr\_ptr

```
// Creates a dynamic int with value of 20 owned by a unique pointer
std::unique_ptr<int> ptr = std::make_unique<int>(20);
```

o o unique\_ptr vector - o

### C++ 11 std::auto\_ptr<sup>o</sup> unique\_ptr auto\_ptr<sup>o</sup>

## C

### CSDL2。

```
std::unique_ptr<SDL_Surface> a; // won't work, UNSAFE!
```

o [SDL\\_Surface](#) [SDL\\_FreeSurface\(\)](#) C。

```
std::unique_ptr<SDL_Surface> a; // won't work, UNSAFE!
```

operator()

```
std::unique_ptr<SDL_Surface> a; // won't work, UNSAFE!
```

```
unique_ptr °  
  
unique_ptr° std::unique_ptr<SDL_Surface, SurfaceDeleteer>std::unique_ptr<SDL_Surface,  
void(*) (SDL_Surface*)> SDL_Surface*°  
  
shared_ptrunique_ptr ° shared_ptrAPI° shared_ptrunique_ptr°
```

```
std::unique_ptr<SDL_Surface> a; // won't work, UNSAFE!
```

## C++ 17

```
template auto  
  
std::unique_ptr<SDL_Surface> a; // won't work, UNSAFE!  
  
std::unique_ptr<SDL_Surface> a; // won't work, UNSAFE!
```

```
auto void SDL_FreeSurface fclose °
```

## auto\_ptr

### C++ 11

```
std::auto_ptrC++ 11C++ 17° C++ 03° std::moveunique_ptrstd::auto_ptr°  
  
std::unique_ptrstd::auto_ptr ° std::auto_ptr°  
  
std::auto_ptrRAII
```

```
{  
    std::auto_ptr<int> p(new int(42));  
    std::cout << *p;  
} // p is deleted here, no memory leaked
```

```
{  
    std::auto_ptr<int> p(new int(42));  
    std::cout << *p;  
} // p is deleted here, no memory leaked
```

## std :: auto\_ptr

```
{  
    std::auto_ptr<int> p(new int(42));  
    std::cout << *p;  
} // p is deleted here, no memory leaked
```

```
""° auto_ptrconst°
```

```
{
```

```
    std::auto_ptr<int> p(new int(42));
    std::cout << *p;
} // p is deleted here, no memory leaked
```

◦ T

```
{  
    std::auto_ptr<int> p(new int(42));  
    std::cout << *p;  
} // p is deleted here, no memory leaked
```

auto\_ptr ◦ auto\_ptr◦

## shared\_ptr

enable\_shared\_from\_thisshared\_ptrthis◦

enable\_shared\_from\_this shared\_from\_thisshared\_ptrthis◦

shared\_ptr

```
#include <memory>
class A: public enable_shared_from_this<A> {
};
A* ap1 =new A();
shared_ptr<A> ap2(ap1); // First prepare a shared pointer to the object and hold it!
// Then get a shared pointer to the object from the object itself
shared_ptr<A> ap3 = ap1->shared_from_this();
int c3 =ap3.use_count(); // =2: pointing to the same object
```

2enable\_shared\_from\_this◦

```
#include <memory>
class A: public enable_shared_from_this<A> {
};
A* ap1 =new A();
shared_ptr<A> ap2(ap1); // First prepare a shared pointer to the object and hold it!
// Then get a shared pointer to the object from the object itself
shared_ptr<A> ap3 = ap1->shared_from_this();
int c3 =ap3.use_count(); // =2: pointing to the same object
```

shared\_ptrshared\_from\_this()shared\_from\_this()◦ C ++ 17std::bad\_alloc◦

shared\_from\_this()shared\_ptr shared\_ptr◦

## std :: shared\_ptr

```
std::shared_ptrstatic_cast const_cast dynamic_castreinterpret_cast◦ std::static_pointer_cast
std::const_pointer_cast std::dynamic_pointer_cast std::const_pointer_cast
std::dynamic_pointer_caststd::reinterpret_pointer_cast
```

```

struct Base { virtual ~Base() noexcept {}; };
struct Derived: Base {};
auto derivedPtr(std::make_shared<Derived>());
auto basePtr(std::static_pointer_cast<Base>(derivedPtr));
auto constBasePtr(std::const_pointer_cast<Base const>(basePtr));
auto constDerivedPtr(std::dynamic_pointer_cast<Derived const>(constBasePtr));

```

std::reinterpret\_pointer\_cast C++ 11C++ 14N392020142 Library Fundamentals TS 。

```

struct Base { virtual ~Base() noexcept {}; };
struct Derived: Base {};
auto derivedPtr(std::make_shared<Derived>());
auto basePtr(std::static_pointer_cast<Base>(derivedPtr));
auto constBasePtr(std::const_pointer_cast<Base const>(basePtr));
auto constDerivedPtr(std::dynamic_pointer_cast<Derived const>(constBasePtr));

```

## value\_ptr

value\_ptr o o

```

// Like std::default_delete:
template<class T>
struct default_copier {
    // a copier must handle a null T const* in and return null:
    T* operator()(T const* tin) const {
        if (!tin) return nullptr;
        return new T(*tin);
    }
    void operator()(void* dest, T const* tin) const {
        if (!tin) return;
        return new(dest) T(*tin);
    }
};

// tag class to handle empty case:
struct empty_ptr_t {};
constexpr empty_ptr_t empty_ptr{};
// the value pointer type itself:
template<class T, class Copier=default_copier<T>, class Deleter=std::default_delete<T>,
         class Base=std::unique_ptr<T, Deleter>
>
struct value_ptr:Base, private Copier {
    using copier_type=Copier;
    // also typedefs from unique_ptr

    using Base::Base;

    value_ptr( T const& t ):
        Base( std::make_unique<T>(t) ),
        Copier()
    {}
    value_ptr( T && t ):
        Base( std::make_unique<T>(std::move(t)) ),
        Copier()
    {}
    // almost-never-empty:
    value_ptr():
        Base( std::make_unique<T>() ),

```

```

Copier()
{}
value_ptr( empty_ptr_t ) {}

value_ptr( Base b, Copier c={} ) :
    Base(std::move(b)),
    Copier(std::move(c))
{}

Copier const& get_copier() const {
    return *this;
}

value_ptr clone() const {
    return (
        Base(
            get_copier()(this->get()),
            this->get_deleter()
        ),
        get_copier()
    );
}
value_ptr(value_ptr&&)=default;
value_ptr& operator=(value_ptr&&)=default;

value_ptr(value_ptr const& o):value_ptr(o.clone()) {}
value_ptr& operator=(value_ptr const&o) {
    if (o && *this) {
        // if we are both non-null, assign contents:
        **this = *o;
    } else {
        // otherwise, assign a clone (which could itself be null):
        *this = o.clone();
    }
    return *this;
}
value_ptr& operator=( T const& t ) {
    if (*this) {
        **this = t;
    } else {
        *this = value_ptr(t);
    }
    return *this;
}
value_ptr& operator=( T && t ) {
    if (*this) {
        **this = std::move(t);
    } else {
        *this = value_ptr(std::move(t));
    }
    return *this;
}
T& get() { return **this; }
T const& get() const { return **this; }
T* get_pointer() {
    if (!*this) return nullptr;
    return std::addressof(get());
}
T const* get_pointer() const {
    if (!*this) return nullptr;
    return std::addressof(get());
}

```

```
}

// operator-> from unique_ptr
};

template<class T, class...Args>
value_ptr<T> make_value_ptr( Args&&... args ) {
    return {std::make_unique<T>(std::forward<Args>(args)...)};
}
```

empty\_ptr\_t**value\_ptr**。unique\_ptr的explicit operator bool() const。.get().get\_pointer()。

pImplpImpl。

Copier。

<https://riptutorial.com/zh-TW/cplusplus/topic/509/>

- ◦
- ◦

## Examples

- 

```
// Forward declaration of functions.
void friend_function();
void non_friend_function();

class PrivateHolder {
public:
    PrivateHolder(int val) : private_value(val) {}
private:
    int private_value;
    // Declare one of the function as a friend.
    friend void friend_function();
};

void non_friend_function() {
    PrivateHolder ph(10);
    // Compilation error: private_value is private.
    std::cout << ph.private_value << std::endl;
}

void friend_function() {
    // OK: friends may access private values.
    PrivateHolder ph(10);
    std::cout << ph.private_value << std::endl;
}
```

- ◦

- PrivateHolderPrivateHolder

```
// Forward declaration of functions.
void friend_function();
void non_friend_function();

class PrivateHolder {
public:
    PrivateHolder(int val) : private_value(val) {}
private:
    int private_value;
    // Declare one of the function as a friend.
    friend void friend_function();
};

void non_friend_function() {
    PrivateHolder ph(10);
```

```

// Compilation error: private_value is private.
std::cout << ph.private_value << std::endl;
}

void friend_function() {
    // OK: friends may access private values.
    PrivateHolder ph(10);
    std::cout << ph.private_value << std::endl;
}

```

```

// Forward declaration of functions.
void friend_function();
void non_friend_function();

class PrivateHolder {
public:
    PrivateHolder(int val) : private_value(val) {}
private:
    int private_value;
    // Declare one of the function as a friend.
    friend void friend_function();
};

void non_friend_function() {
    PrivateHolder ph(10);
    // Compilation error: private_value is private.
    std::cout << ph.private_value << std::endl;
}

void friend_function() {
    // OK: friends may access private values.
    PrivateHolder ph(10);
    std::cout << ph.private_value << std::endl;
}

```

PrivateHolderDerivedPrivateHolder::private\_value °

```

class Accesser {
public:
    void private_accesser();
};

class PrivateHolder {
public:
    PrivateHolder(int val) : private_value(val) {}
    friend void Accesser::private_accesser();
private:
    int private_value;
};

void Accesser::private_accesser() {
    PrivateHolder ph(10);
    // OK: this method is declares as friend.
    std::cout << ph.private_value << std::endl;
}

```

°

```

class Accesser {
public:
    void private_accesser1();
    void private_accesser2();
};

class PrivateHolder {
public:
    PrivateHolder(int val) : private_value(val) {}
    friend class Accesser;
private:
    int private_value;
};

void Accesser::private_accesser1() {
    PrivateHolder ph(10);
    // OK.
    std::cout << ph.private_value << std::endl;
}

void Accesser::private_accesser2() {
    PrivateHolder ph(10);
    // OK.
    std::cout << ph.private_value + 1 << std::endl;
}

```

◦ ◦

```

class Accesser {
public:
    void private_accesser1();
    void private_accesser2();
};

class PrivateHolder {
public:
    PrivateHolder(int val) : private_value(val) {}
    friend class Accesser;
private:
    int private_value;
};

void Accesser::private_accesser1() {
    PrivateHolder ph(10);
    // OK.
    std::cout << ph.private_value << std::endl;
}

void Accesser::private_accesser2() {
    PrivateHolder ph(10);
    // OK.
    std::cout << ph.private_value + 1 << std::endl;
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/3275/>

## PromisesFutures

```
std::promise<
```

```
std::future<
```

## Examples

### std :: futurestd :: promise

```
{
    auto promise = std::promise<std::string>();

    auto producer = std::thread([&]
    {
        promise.set_value("Hello World");
    });

    auto future = promise.get_future();

    auto consumer = std::thread([&]
    {
        std::cout << future.get();
    });

    producer.join();
    consumer.join();
}
```

```
std::asyncdeferedasynco  asyncfuture;future<
```

```
template<typename F>
auto async_deferred(F&& func) -> std::future<decltype(func())>
{
    using result_type = decltype(func());

    auto promise = std::promise<result_type>();
    auto future = promise.get_future();

    std::thread(std::bind([=] (std::promise<result_type>& promise)
    {
        try
        {
            promise.set_value(func());
            // Note: Will not work with std::promise<void>. Needs some meta-template
            programming which is out of scope for this example.
        }
        catch(...)
        {
            promise.set_exception(std::current_exception());
        }
    }, std::move(promise))).detach();
```

```
        return future;
    }
```

## std :: packaged\_taskstd :: future

std::packaged\_task

```
template<typename F>
auto async_deferred(F&& func) -> std::future<decltype(func())>
{
    auto task = std::packaged_task<decltype(func())()>(std::forward<F>(func));
    auto future = task.get_future();

    std::thread(std::move(task)).detach();

    return std::move(future);
}
```

◦◦◦ std :: thread◦◦◦

std::asyncstd::future◦◦◦

## std :: future\_errorstd :: future\_errc

std :: promisesstd :: futuresstd :: future\_error◦◦◦

std :: future\_errc

```
enum class future_errc {
    broken_promise           = /* the task is no longer shared */,
    future_already_retrieved = /* the answer was already retrieved */,
    promise_already_satisfied = /* the answer was stored already */,
    no_state                 = /* access to a promise in non-shared state */
};
```

```
enum class future_errc {
    broken_promise           = /* the task is no longer shared */,
    future_already_retrieved = /* the answer was already retrieved */,
    promise_already_satisfied = /* the answer was stored already */,
    no_state                 = /* access to a promise in non-shared state */
};
```

```
enum class future_errc {
    broken_promise           = /* the task is no longer shared */,
    future_already_retrieved = /* the answer was already retrieved */,
    promise_already_satisfied = /* the answer was stored already */,
    no_state                 = /* access to a promise in non-shared state */
};
```

```
enum class future_errc {
    broken_promise           = /* the task is no longer shared */,
    future_already_retrieved = /* the answer was already retrieved */,
    promise_already_satisfied = /* the answer was stored already */,
    no_state                 = /* access to a promise in non-shared state */
};
```

```

promise_already_satisfied = /* the answer was stored already */,
no_state                  = /* access to a promise in non-shared state */
};


```

## std :: promise

```

enum class future_errc {
    broken.promise          = /* the task is no longer shared */,
    future_already_retrieved = /* the answer was already retrieved */,
    promise_already_satisfied = /* the answer was stored already */,
    no.state                 = /* access to a promise in non-shared state */
};


```

## std :: future std :: async

std::async`merge_sort` std::future

```

#include <iostream>
using namespace std;

void merge(int low,int mid,int high, vector<int>&num)
{
    vector<int> copy(num.size());
    int h,i,j,k;
    h=low;
    i=low;
    j=mid+1;

    while( (h<=mid) && (j<=high) )
    {
        if (num[h]<=num[j])
        {
            copy[i]=num[h];
            h++;
        }
        else
        {
            copy[i]=num[j];
            j++;
        }
        i++;
    }
    if (h>mid)
    {
        for (k=j;k<=high;k++)
        {
            copy[i]=num[k];
            i++;
        }
    }
    else
    {
        for (k=h;k<=mid;k++)
        {
            copy[i]=num[k];
            i++;
        }
    }
}


```

```

    }
    for(k=low;k<=high;k++)
        swap(num[k],copy[k]);
}

void merge_sort(int low,int high,vector<int>& num)
{
    int mid;
    if(low<high)
    {
        mid = low + (high-low)/2;
        auto future1      = std::async(std::launch::deferred, [&]()
                                         {
                                             merge_sort(low,mid,num);
                                         });
        auto future2      = std::async(std::launch::deferred, [&]()
                                         {
                                             merge_sort(mid+1,high,num) ;
                                         });
        future1.get();
        future2.get();
        merge(low,mid,high,num);
    }
}

```

`std::launch_deferred``std::async`  $\circ$  `std::async``std::future::get()`  $\circ$

`std::launch_async`

`std::launch::deferred`| `std::launch::async`  $\circ$

- `std :: async`。
- `std :: future`。
- `std :: promise`。
- `std :: packaged_task`。

<https://riptutorial.com/zh-TW/cplusplus/topic/9840/>

## Examples

◦

```
struct {
    int foo;
    double bar;
} foobar;

foobar.foo = 5;
foobar.bar = 4.0;

class {
    int baz;
public:
    int buzz;

    void setBaz(int v) {
        baz = v;
    }
} barbar;

barbar.setBaz(15);
barbar.buzz = 2;
```

### C ++◦

```
struct Example {
    struct {
        int inner_b;
    };
    int outer_b;

    //The anonymous struct's members are accessed as if members of the parent struct
    Example() : inner_b(2), outer_b(4) {
        inner_b = outer_b + 2;
    }
};

Example ex;

//The same holds true for external code referencing the struct
ex.inner_b -= ex.outer_b;
```

typedefusing

### C ++ 11

```
using vec2d = struct {
    float x;
    float y;
```

```
};
```

```
using vec2d = struct {
    float x;
    float y;
};
```

```
using vec2d = struct {
    float x;
    float y;
};
```

- “**struct**”。

```
struct Sample {
    union {
        int a;
        int b;
    };
    int c;
};

int main()
{
    Sample sa;
    sa.a =3;
    sa.b =4;
    sa.c =5;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2704/>

90:

UBISO C ++§1.3.24N4296”。

UB.

UB.

C ++.

- ○  
● ○  
● ○  
● ○

1

1

C++””。。

- -
  - **gccclang“Undefined Behavior Sanitizer”** -fsanitize=undefined ◦
  - **lint**◦

C ++ 14ISO / IEC 14882:2014 1.9

1. °

2. `sizeof(int)` . . .

3-8888

#### 4. const. [ ° - ]

## Examples

```
int *ptr = nullptr;  
*ptr = 1; // Undefined behavior
```

`*ptr`。

6

## voidreturn

void return

◦

```
int function() {
    // Missing return statement
}

int main() {
    function(); //Undefined Behavior
}
```

◦

---

```
main◦ mainreturnreturn 0;◦
```

## C ++ 11

```
char *str = "hello world";
str[0] = 'H';
```

```
"hello world"◦
```

```
strC ++ 03◦ 2003◦ 2003◦
```

## C ++ 11

```
C ++ 11◦
```

```
char *str = "hello world";
str[0] = 'H';
```

```
int array[] = {1, 2, 3, 4, 5};
array[5] = 0; // Undefined behavior
```

```
array + 5 ◦
```

```
int array[] = {1, 2, 3, 4, 5};
array[5] = 0; // Undefined behavior
```

◦ ◦

```
int x = 5 / 0; // Undefined behavior
```

```
0◦
```

```
int x = 5 / 0; // Undefined behavior
```

**IEEE-754**NaN 0.0f infinity -infinity ◦

```
int x = INT_MAX + 1;  
  
// x can be anything -> Undefined behavior
```

◦

## C++ 115/4

◦

```
int x = INT_MAX + 1;  
  
// x can be anything -> Undefined behavior
```

2<sup>n</sup>◦

## C++ 113.9.1 / 4

```
int x = INT_MAX + 1;  
  
// x can be anything -> Undefined behavior
```

“if”

```
int a;  
std::cout << a; // Undefined behavior!
```

a◦

“”◦ a◦ “”◦

◦ - ◦

## C++ 14

unsigned char

- ;
- ;
- unsigned char;
- unsigned char;
- unsigned char;

◦ ◦

static

```
int a;  
std::cout << a; // Undefined behavior!
```

## ODR。

foo.h

```
class Foo {  
public:  
    double x;  
private:  
    int y;  
};  
  
Foo get_foo();
```

foo.cpp

```
class Foo {  
public:  
    double x;  
private:  
    int y;  
};  
  
Foo get_foo();
```

main.cpp

```
class Foo {  
public:  
    double x;  
private:  
    int y;  
};  
  
Foo get_foo();
```

**class ::Foo** °

newdelete° newdelete°

newdelete[]° delete[]new°

mallocfree°

```
int* p1 = new int;  
delete p1;      // correct  
// delete[] p1; // undefined  
// free(p1);    // undefined  
  
int* p2 = new int[10];  
delete[] p2;    // correct  
// delete p2;   // undefined  
// free(p2);    // undefined  
  
int* p3 = static_cast<int*>(malloc(sizeof(int)));  
free(p3);       // correct
```

```
// delete p3; // undefined  
// delete[] p3; // undefined
```

C++~~mallocfree std::vector<std::string> raw newdelete[]。~~

CV◦

```
float x = 42;  
int y = reinterpret_cast<int&>(x);
```

◦

- ◦
- char unsigned char **char**◦
- ◦

◦ ◦

```
float x = 42;  
int y = reinterpret_cast<int&>(x);
```

C++IEEE 754◦

```
float x = 1.0;  
for (int i = 0; i < 10000; i++) {  
    x *= 10.0; // will probably overflow eventually; undefined behavior  
}
```

## 10.4

;10.3◦

C++Scott Meyers dstructors◦

```
class transaction  
{  
public:  
    transaction() { log_it(); }  
    virtual void log_it() const = 0;  
};  
  
class sell_transaction : public transaction  
{  
public:  
    virtual void log_it() const { /* Do something */ }  
};  
  
sell_transaction
```

```
class transaction  
{
```

```

public:
    transaction() { log_it(); }
    virtual void log_it() const = 0;
};

class sell_transaction : public transaction
{
public:
    virtual void log_it() const { /* Do something */ }
};

```

sell\_transaction sell\_transaction transaction. transactions sell\_transaction transaction.

transaction::transaction() log\_it - sell\_transaction::log\_it.

- log\_it.
- log\_it transaction::log\_it.

◦

```

class base {};
class derived: public base {};

int main() {
    base* p = new derived();
    delete p; // The is undefined behavior!
}

```

[expr.delete]§5.3.5/ 3 virtual delete

◦  
◦  
◦ ◦

```

#include <iostream>
int& getX() {
    int x = 42;
    return x;
}
int main() {
    int& r = getX();
    std::cout << r << "\n";
}

```

getXx. ◦ r. 42. ◦

`std` `posix`

17.6.4.2.1 / 1 std

## C ++stdstd。

posix 17.6.4.2.2 / 1

## C ++posixposix。

```
#include <algorithm>

namespace std
{
    int foo() {}
}
```

algorithm °

◦ ◦ ◦

```
#include <algorithm>

namespace std
{
    int foo() {}
}
```

```
#include <algorithm>

namespace std
{
    int foo() {}
}
```

•  
•  
•

◦

```
double x = 1e100;
int y = x; // int probably cannot hold numbers that large, so this is UB
```

static\_cast<resp> resp = resp.reference °

◦ ◦

```
int f();
void (*p)() = reinterpret_cast<void(*)()>(f);
p(); // undefined
```

## const

const ° const constconst ° constmutableconst °

```
const_cast
```

```
const int x = 123;
const_cast<int&>(x) = 456;
std::cout << x << '\n';
```

const int123 ◦ const◦ ◦

```
const int x = 123;
const_cast<int&>(x) = 456;
std::cout << x << '\n';
```

getFooconst Foom\_x123 ◦ do\_evil foo.m\_x**456.**

do\_evil; Foo\* **setter**◦ const\_cast const Foo◦ Foo◦ constconst const Foo\* thisFoo\* const Foo\*◦

◦

◦ static\_cast◦

```
struct Base { int x; };
struct Derived : Base { int y; };
int Derived::*pdy = &Derived::y;
int Base::*pby = static_cast<int Base::*>(pdy);

Base* b1 = new Derived;
b1->*pby = 42; // ok; sets y in Derived object to 42
Base* b2 = new Base;
b2->*pby = 42; // undefined; there is no y member in Base
```

static\_cast TD::\*TB::\* B◦ ◦

• ◦ ◦

```
int a[10];
int* p1 = &a[5];
int* p2 = p1 + 4; // ok; p2 points to a[9]
int* p3 = p1 + 5; // ok; p2 points to one past the end of a
int* p4 = p1 + 6; // UB
int* p5 = p1 - 5; // ok; p2 points to a[0]
int* p6 = p1 - 6; // UB
int* p7 = p3 - 5; // ok; p7 points to a[5]
```

• ◦ ◦ 0◦

```
int a[10];
int* p1 = &a[5];
int* p2 = p1 + 4; // ok; p2 points to a[9]
int* p3 = p1 + 5; // ok; p2 points to one past the end of a
int* p4 = p1 + 6; // UB
int* p5 = p1 - 5; // ok; p2 points to a[0]
int* p6 = p1 - 6; // UB
int* p7 = p3 - 5; // ok; p7 points to a[5]
```

- std::ptrdiff\_t。
- cv-qualification。 “[。 ”

```
int a[10];
int* p1 = &a[5];
int* p2 = p1 + 4; // ok; p2 points to a[9]
int* p3 = p1 + 5; // ok; p2 points to one past the end of a
int* p4 = p1 + 6; // UB
int* p5 = p1 - 5; // ok; p2 points to a[0]
int* p6 = p1 - 6; // UB
int* p7 = p3 - 5; // ok; p7 points to a[5]
```

◦ ◦

```
const int a = 42;
const int b = a << -1; // UB
const int c = a << 0; // ok
const int d = a << 32; // UB if int is 32 bits or less
const int e = a >> 32; // also UB if int is 32 bits or less
const signed char f = 'x';
const int g = f << 10; // ok even if signed char is 10 bits or less;
                      // int must be at least 16 bits
```

## [[noreturn]]

C++11

### [dcl.attr.noreturn]

```
[[ noreturn ]] void f() {
    throw "error"; // OK
}
[[ noreturn ]] void q(int i) { // behavior is undefined if called with an argument <= 0
    if (i > 0)
        throw "positive";
}
```

◦

```
struct S {
    ~S() { std::cout << "destroying S\n"; }
};
int main() {
    S s;
    s.~S();
} // UB: s destroyed a second time here
```

std::unique\_ptr<T> T。

```
struct S {
    ~S() { std::cout << "destroying S\n"; }
};
```

```
int main() {
    S s;
    s.~S();
} // UB: s destroyed a second time here
```

shared\_ptr<sup>o</sup>

```
struct S {
    ~S() { std::cout << "destroying S\n"; }
};

int main() {
    S s;
    s.~S();
} // UB: s destroyed a second time here
```

[temp.inst] / 17

```
template<class T> class X {
    X<T*>* p; // OK
    X<T*> a; // implicit generation of X<T> requires
                // the implicit instantiation of X<T*> which requires
                // the implicit instantiation of X<T**> which ...
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1812/>

◦ ◦

## Examples

TU

◦

- Foo.cpp

```
#include <iostream>

int dummyFoo = ((std::cout << "foo"), 0);
```

- bar.cpp

```
#include <iostream>

int dummyFoo = ((std::cout << "foo"), 0);
```

- main.cpp

```
#include <iostream>

int dummyFoo = ((std::cout << "foo"), 0);
```

```
#include <iostream>

int dummyFoo = ((std::cout << "foo"), 0);
```

```
#include <iostream>

int dummyFoo = ((std::cout << "foo"), 0);
```

*Fiasco* ◦

◦

```
enum class E {
    X = 1,
    Y = 1000,
};

// assume 1000 does not fit into a char
char c1 = static_cast<char>(E::X); // c1 is 1
char c2 = static_cast<char>(E::Y); // c2 has an unspecified value
```

◦

```
enum class E {
    X = 1,
    Y = 1000,
};

// assume 1000 does not fit into a char
char c1 = static_cast<char>(E::X); // c1 is 1
char c2 = static_cast<char>(E::Y); // c2 has an unspecified value
```

```
enum class E {
    X = 1,
    Y = 1000,
};

// assume 1000 does not fit into a char
char c1 = static_cast<char>(E::X); // c1 is 1
char c2 = static_cast<char>(E::Y); // c2 has an unspecified value
```

s3ONE TWOFOUR ◦

\*

void\*T\*T◦

```
// Suppose that alignof(int) is 4
int x = 42;
void* p1 = &x;
// Do some pointer arithmetic...
void* p2 = static_cast<char*>(p1) + 2;
int* p3 = static_cast<int*>(p2);
```

p3p2int◦

## reinterpret\_cast

reinterpret\_cast◦

```
int f();
auto fp = reinterpret_cast<int(*)(int)>(&f); // fp has unspecified value
```

## C++ 03

reinterpret\_cast◦

```
int f();
auto fp = reinterpret_cast<int(*)(int)>(&f); // fp has unspecified value
```

static\_cast<char\*>(static\_cast<void\*>(&x))px◦ C++ 11◦ ◦

< > <=>=

•

◦ 1◦

```
int x;
int y;
const bool b1 = &x < &y;           // unspecified
int a[10];
const bool b2 = &a[0] < &a[1];     // true
const bool b3 = &a[0] < &x;       // unspecified
const bool b4 = (a + 9) < (a + 10); // true
                                // note: a+10 points past the end of the array
```

• ◦

```
int x;
int y;
const bool b1 = &x < &y;           // unspecified
int a[10];
const bool b2 = &a[0] < &a[1];     // true
const bool b3 = &a[0] < &x;       // unspecified
const bool b4 = (a + 9) < (a + 10); // true
                                // note: a+10 points past the end of the array
```

◦ ◦

- sizeof◦
- ◦
- ◦
- offsetof◦
- ◦

◦

```
void f() {
    int x;
    int& r = x;
    // do something with r
}
```

rxfrx r◦

◦ x = 1, y = 2x = 2, y = 1◦

```
int f(int x, int y) {
    printf("x = %d, y = %d\n", x, y);
}
int get_val() {
    static int x = 0;
    return ++x;
}
int main() {
    f(get_val(), get_val());
}
```

## C++ 17

### C++ 17.

◦

```
int f(int x, int y) {
    printf("x = %d, y = %d\n", x, y);
}
int get_val() {
    static int x = 0;
    return ++x;
}
int main() {
    f(get_val(), get_val());
}
```

```
int f(int x, int y) {
    printf("x = %d, y = %d\n", x, y);
}
int get_val() {
    static int x = 0;
    return ++x;
}
int main() {
    f(get_val(), get_val());
}
```

```
int f(int x, int y) {
    printf("x = %d, y = %d\n", x, y);
}
int get_val() {
    static int x = 0;
    return ++x;
}
int main() {
    f(get_val(), get_val());
}
```

barmakefrom

```
int f(int x, int y) {
    printf("x = %d, y = %d\n", x, y);
}
int get_val() {
    static int x = 0;
    return ++x;
}
int main() {
    f(get_val(), get_val());
}
```

◦ make\_int C++ 17 barfrom\_intmake\_int from\_int .

## C++ 11

- v2{1, 2, 3, 4} v1◦

```
int main() {  
    std::vector<int> v1{1, 2, 3, 4};  
    std::vector<int> v2 = std::move(v1);  
}
```

- std::unique\_ptr<T> null◦

<https://riptutorial.com/zh-TW/cplusplus/topic/4939/>

# 92:

CC ++。 C ++。 。

C ++。 /GNU / LinuxGNU MakeVisual C ++ / Visual StudioNMAKE。

IDEIDE。 IDE/CMake for EclipseMicrosoft Visual Studio 2012。

## Examples

### CMake

CMakeIDE。 CMake“Hello World”C ++。

CMake“CMakeLists.txt”。 CMakeLists.txt

```
cmake_minimum_required(VERSION 2.4)
project(HelloWorld)
add_executable(HelloWorld main.cpp)
```

Coliru。

CMakemain.cpp“HelloWorld”。

/ IDE

```
cmake_minimum_required(VERSION 2.4)
project(HelloWorld)
add_executable(HelloWorld main.cpp)
```

```
cmake_minimum_required(VERSION 2.4)
project(HelloWorld)
add_executable(HelloWorld main.cpp)
```

◦ “out-of-source”

```
cmake_minimum_required(VERSION 2.4)
project(HelloWorld)
add_executable(HelloWorld main.cpp)
```

CMakeshell

```
cmake_minimum_required(VERSION 2.4)

project(HelloWorld)

add_executable(HelloWorld main.cpp)
```

## CMakeIDE。Visual Studio nmake makefile

```
cmake_minimum_required(VERSION 2.4)

project(HelloWorld)

add_executable(HelloWorld main.cpp)
```

## GNU make

GNU Make make shell。 GNU Make Make。 Unix POSIX Linux Mac OS X BSD。

GNU Make GNUGNUGNU / Linux。 GNU Make Windows Mac OS X。 GNU Make CC ++。

make Makefile。 Makefile

## Makefile

```
# Set some variables to use in our command
# First, we set the compiler to be g++
CXX=g++

# Then, we say that we want to compile with g++'s recommended warnings and some extra ones.
CXXFLAGS=-Wall -Wextra -pedantic

# This will be the output file
EXE=app

SRCS=main.cpp

# When you call `make` at the command line, this "target" is called.
# The $(EXE) at the right says that the `all` target depends on the `$(EXE)` target.
# $(EXE) expands to be the content of the EXE variable
# Note: Because this is the first target, it becomes the default target if `make` is called
# without target
all: $(EXE)

# This is equivalent to saying
# app: $(SRCS)
# $(SRCS) can be separated, which means that this target would depend on each file.
# Note that this target has a "method body": the part indented by a tab (not four spaces).
# When we build this target, make will execute the command, which is:
# g++ -Wall -Wextra -pedantic -o app main.cpp
# I.E. Compile main.cpp with warnings, and output to the file ./app
$(EXE): $(SRCS)
    @$(CXX) $(CXXFLAGS) -o $@ $(SRCS)
```

```

# This target should reverse the `all` target. If you call
# make with an argument, like `make clean`, the corresponding target
# gets called.
clean:
    @rm -f $(EXE)

o Makefile:10: *** missing separator. Stop. Makefile:10: *** missing separator.
Stop.

# Set some variables to use in our command
# First, we set the compiler to be g++
CXX=g++

# Then, we say that we want to compile with g++'s recommended warnings and some extra ones.
CXXFLAGS=-Wall -Wextra -pedantic

# This will be the output file
EXE=app

SRCS=main.cpp

# When you call `make` at the command line, this "target" is called.
# The $(EXE) at the right says that the `all` target depends on the `$(EXE)` target.
# $(EXE) expands to be the content of the EXE variable
# Note: Because this is the first target, it becomes the default target if `make` is called
# without target
all: $(EXE)

# This is equivalent to saying
# app: $(SRCS)
# $(SRCS) can be separated, which means that this target would depend on each file.
# Note that this target has a "method body": the part indented by a tab (not four spaces).
# When we build this target, make will execute the command, which is:
# g++ -Wall -Wextra -pedantic -o app main.cpp
# I.E. Compile main.cpp with warnings, and output to the file ./app
$(EXE): $(SRCS)
    @$(CXX) $(CXXFLAGS) -o $@ $(SRCS)

# This target should reverse the `all` target. If you call
# make with an argument, like `make clean`, the corresponding target
# gets called.
clean:
    @rm -f $(EXE)

```

## make a.cpp b.cpp

```

# Set some variables to use in our command
# First, we set the compiler to be g++
CXX=g++

# Then, we say that we want to compile with g++'s recommended warnings and some extra ones.
CXXFLAGS=-Wall -Wextra -pedantic

# This will be the output file
EXE=app

SRCS=main.cpp

```

```

# When you call `make` at the command line, this "target" is called.
# The $(EXE) at the right says that the `all` target depends on the `$(EXE)` target.
# $(EXE) expands to be the content of the EXE variable
# Note: Because this is the first target, it becomes the default target if `make` is called
# without target
all: $(EXE)

# This is equivalent to saying
# app: $(SRCS)
# $(SRCS) can be separated, which means that this target would depend on each file.
# Note that this target has a "method body": the part indented by a tab (not four spaces).
# When we build this target, make will execute the command, which is:
# g++ -Wall -Wextra -pedantic -o app main.cpp
# I.E. Compile main.cpp with warnings, and output to the file ./app
$(EXE): $(SRCS)
    @$(CXX) $(CXXFLAGS) -o $@ $(SRCS)

# This target should reverse the `all` target. If you call
# make with an argument, like `make clean`, the corresponding target
# gets called.
clean:
    @rm -f $(EXE)

```

## Makefile

### Makefile

```

# Set some variables to use in our command
# First, we set the compiler to be g++
CXX=g++

# Then, we say that we want to compile with g++'s recommended warnings and some extra ones.
CXXFLAGS=-Wall -Wextra -pedantic

# This will be the output file
EXE=app

SRCS=main.cpp

# When you call `make` at the command line, this "target" is called.
# The $(EXE) at the right says that the `all` target depends on the `$(EXE)` target.
# $(EXE) expands to be the content of the EXE variable
# Note: Because this is the first target, it becomes the default target if `make` is called
# without target
all: $(EXE)

# This is equivalent to saying
# app: $(SRCS)
# $(SRCS) can be separated, which means that this target would depend on each file.
# Note that this target has a "method body": the part indented by a tab (not four spaces).
# When we build this target, make will execute the command, which is:
# g++ -Wall -Wextra -pedantic -o app main.cpp
# I.E. Compile main.cpp with warnings, and output to the file ./app
$(EXE): $(SRCS)
    @$(CXX) $(CXXFLAGS) -o $@ $(SRCS)

# This target should reverse the `all` target. If you call
# make with an argument, like `make clean`, the corresponding target
# gets called.

```

```
clean:  
  @rm -f $(EXE)
```

- Makefile◦
- 

make **stackoverflowdmckeestackoverflow**◦

## SCons

**Scons -A Python -language“Hello World”C ++**◦

SConstruct **SCons**◦ hello.cpp◦

```
Program('hello.cpp')
```

scons◦

```
Program('hello.cpp')
```

◦

EnvironmentGlob◦ SConstruct

```
Program('hello.cpp')
```

srccpphello◦ CPPPATH/usr/include/boost **C ++ 11**◦

---

Ninja“”。 NinjaCMakeMeson◦

NinjaC ++PythonChromiumSCons◦

## NMAKEMicrosoft

---

NMAKEMicrosoftMicrosoft Visual Studio/Visual C ++◦

NMAKEMakeUnixMakeWindowsUnix◦

## AutotoolsGNU

---

AutotoolsGNU◦ MakefileGNU Make◦ Autotools◦

Autotools

- Autoconf
- Automake<sub>make</sub>

## Autotools Unix Makefile

```
./configure && make && make install
```

## Autotools POSIX。

<https://riptutorial.com/zh-TW/cplusplus/topic/8200/>

## Examples

### std :: for\_each

```
template<class InputIterator, class Function>
Function for_each(InputIterator first, InputIterator last, Function f);
```

`ffirst [first, last)last - 1[first, last)◦`

`first, last - f◦`

`f - [first, last)◦`

`f C++ 11 std::move(f) C++ 11◦`

`last - firstf last - first◦`

### C++ 11

```
template<class InputIterator, class Function>
Function for_each(InputIterator first, InputIterator last, Function f);
```

`vvstd::cout ◦`

### std :: next\_permutation

```
template< class Iterator >
bool next_permutation( Iterator first, Iterator last );
template< class Iterator, class Compare >
bool next_permutation( Iterator first, Iterator last, Compare cmpFun );
```

`[firstlast]◦ cmpFun ◦`

`first -`

`last -`

`true◦`

`false◦`

`Onnfirstlast◦`

```

template< class Iterator >
bool next_permutation( Iterator first, Iterator last );
template< class Iterator, class Compare >
bool next_permutation( Iterator first, Iterator last, Compare cmpFun );

```

1,2,3°

```

template< class Iterator >
bool next_permutation( Iterator first, Iterator last );
template< class Iterator, class Compare >
bool next_permutation( Iterator first, Iterator last, Compare cmpFun );

```

**std ::**

<numeric>

```

template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)

```

**std :: accumulate**[first, last)f[first, last]init°

```

template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)

```

1operator+f °

```

first, last - f°
init - °
f - °

```

f°

**On**xk nfirstlast Okf°

```

template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)

```

```

template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)

```

## C++ 11

```
template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)
```

```
template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)
```

## C++ 11

```
template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)
```

```
template<class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init); // (1)

template<class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init, BinaryOperation f); // (2)
```

## std ::

```
template <class InputIterator, class T>
InputIterator find (InputIterator first, InputIterator last, const T& val);
```

val [firstlast

first => iterator last => iterator val =>

==val val last =

```
template <class InputIterator, class T>
InputIterator find (InputIterator first, InputIterator last, const T& val);
```

```
template <class InputIterator, class T>
InputIterator find (InputIterator first, InputIterator last, const T& val);
```

## std ::

```
template <class InputIterator, class T>
typename iterator_traits<InputIterator>::difference_type
count (InputIterator first, InputIterator last, const T& val);
```

val

first =>

last =>

val =>

==val.

```
template <class InputIterator, class T>
typename iterator_traits<InputIterator>::difference_type
count (InputIterator first, InputIterator last, const T& val);
```

```
template <class InputIterator, class T>
typename iterator_traits<InputIterator>::difference_type
count (InputIterator first, InputIterator last, const T& val);
```

## std :: count\_if

```
template <class InputIterator, class UnaryPredicate>
typename iterator_traits<InputIterator>::difference_type
count_if (InputIterator first, InputIterator last, UnaryPredicate red);
```

true

first => iterator<sub>last</sub> => iterator<sub>red</sub> =>truefalse

true.

```
template <class InputIterator, class UnaryPredicate>
typename iterator_traits<InputIterator>::difference_type
count_if (InputIterator first, InputIterator last, UnaryPredicate red);
```

```
template <class InputIterator, class UnaryPredicate>
typename iterator_traits<InputIterator>::difference_type
count_if (InputIterator first, InputIterator last, UnaryPredicate red);
```

## std :: find\_if

```
template <class InputIterator, class UnaryPredicate>
InputIterator find_if (InputIterator first, InputIterator last, UnaryPredicate pred);
```

predtrue.

first => iterator<sub>last</sub> => iterator<sub>pred</sub> =>truefalse

predtrue. val

```
template <class InputIterator, class UnaryPredicate>
InputIterator find_if (InputIterator first, InputIterator last, UnaryPredicate pred);
```

```
template <class InputIterator, class UnaryPredicate>
InputIterator find_if (InputIterator first, InputIterator last, UnaryPredicate pred);
```

## std :: min\_element

```
template <class ForwardIterator>
ForwardIterator min_element (ForwardIterator first, ForwardIterator last);

template <class ForwardIterator, class Compare>
ForwardIterator min_element (ForwardIterator first, ForwardIterator last, Compare comp);
```

first -  
last - comp - truefalse2.

◦

```
template <class ForwardIterator>
ForwardIterator min_element (ForwardIterator first, ForwardIterator last);

template <class ForwardIterator, class Compare>
ForwardIterator min_element (ForwardIterator first, ForwardIterator last, Compare comp);
```

```
template <class ForwardIterator>
ForwardIterator min_element (ForwardIterator first, ForwardIterator last);

template <class ForwardIterator, class Compare>
ForwardIterator min_element (ForwardIterator first, ForwardIterator last, Compare comp);
```

## std :: nth\_element

std::nth\_element n◦ nn◦ ;◦

- ◦

n◦ n/2◦ 536◦

◦

```
std::vector<int> v{5, 1, 2, 3, 4};

std::vector<int>::iterator b = v.begin();
std::vector<int>::iterator e = v.end();

std::vector<int>::iterator med = b;
std::advance(med, v.size() / 2);

// This makes the 2nd position hold the median.
std::nth_element(b, med, e);

// The median is now at v[2].
```

p

```
std::vector<int> v{5, 1, 2, 3, 4};

std::vector<int>::iterator b = v.begin();
std::vector<int>::iterator e = v.end();

std::vector<int>::iterator med = b;
std::advance(med, v.size() / 2);

// This makes the 2nd position hold the median.
std::nth_element(b, med, e);

// The median is now at v[2].
```

pos°

<https://riptutorial.com/zh-TW/cplusplus/topic/3177/>

# 94:

C++ 14. . . std::vector std::vector<int> .

- template <template-parameter-list>
- <template-parameter-list> / \*C++ 11 \*/
- <>
- 
- extern / \*C++ 11 \*/
- template <template-parameter-list> class ... opt identifier opt
- template <template-parameter-list> opt = id-expression
- template <template-parameter-list> typename ... opt identifier opt / \*C++ 17 \*/
- template <template-parameter-list> typename identifier opt = id-expression / \*C++ 17 \*/
- . id-expression
- postfix-expression -> template id-expression
- nested-name-specifier template simple-template-id ::

template C++.

1. <> .

```
template <class T>
void increment(T& x) { ++x; }
```

2. <> .

```
template <class T>
void increment(T& x) { ++x; }
```

3. <> .

```
template <class T>
void increment(T& x) { ++x; }
```

4. .

```
template <class T>
void increment(T& x) { ++x; }
```

5. :: .-> .

```
template <class T>
void increment(T& x) { ++x; }
```

C++ 11 export . .

foo.h

```
template <class T>
void increment(T& x) { ++x; }
```

foo.cpp

```
template <class T>
void increment(T& x) { ++x; }
```

main.cpp

```
template <class T>
void increment(T& x) { ++x; }
```

export C++11; export .

## Examples

◦

```
// 'T' stands for the unknown type
// Both of our arguments will be of the same type.
template<typename T>
void printSum(T add1, T add2)
{
    std::cout << (add1 + add2) << std::endl;
}
```

◦

```
// 'T' stands for the unknown type
// Both of our arguments will be of the same type.
template<typename T>
void printSum(T add1, T add2)
{
    std::cout << (add1 + add2) << std::endl;
}
```

template;C++.

◦

```
// 'T' stands for the unknown type
// Both of our arguments will be of the same type.
template<typename T>
void printSum(T add1, T add2)
{
    std::cout << (add1 + add2) << std::endl;
}
```

◦ make\_X() template structure X

```
// 'T' stands for the unknown type
// Both of our arguments will be of the same type.
template<typename T>
void printSum(T add1, T add2)
{
    std::cout << (add1 + add2) << std::endl;
}
```

```
// 'T' stands for the unknown type
// Both of our arguments will be of the same type.
template<typename T>
void printSum(T add1, T add2)
{
    std::cout << (add1 + add2) << std::endl;
}
```

◦ ◦ ◦

```
template <typename T>
void f(T &&t);
```

t

```
template <typename T>
void f(T &&t);
```

TX X& tX TXt◦ X X&& ◦

decltype(t) T◦

tstd::forward

```
template <typename T>
void f(T &&t);
```

```
template <typename T>
void f(T &&t);
```

v

```
template <typename T>
void f(T &&t);
```

◦ ◦ ◦ main()

```
#include <iostream>
using std::cout;

template <typename T>           // A simple class to hold one number of any type
class Number {
public:
    void setNum(T n);          // Sets the class field to the given number
```

```

    T plus1() const;           // returns class field's "follower"
private:
    T num;                  // Class field
};

template <typename T>           // Set the class field to the given number
void Number<T>::setNum(T n) {
    num = n;
}

template <typename T>           // returns class field's "follower"
T Number<T>::plus1() const {
    return num + 1;
}

int main() {
    Number<int> anInt;        // Test with an integer (int replaces T in the class)
    anInt.setNum(1);
    cout << "My integer + 1 is " << anInt.plus1() << "\n";      // Prints 2

    Number<double> aDouble;   // Test with a double
    aDouble.setNum(3.1415926535897);
    cout << "My double + 1 is " << aDouble.plus1() << "\n";      // Prints 4.14159

    Number<float> aFloat;     // Test with a float
    aFloat.setNum(1.4);
    cout << "My float + 1 is " << aFloat.plus1() << "\n";      // Prints 2.4

    return 0; // Successful completion
}

```

/.

```

template <typename T>
T sqrt(T t) { /* Some generic implementation */ }

```

```

template <typename T>
T sqrt(T t) { /* Some generic implementation */ }

```

sqrt(4.0)sqrt(4)◦

◦ /

```

// Common case:
template<typename T, typename U>
struct S {
    T t_val;
    U u_val;
};

// Special case when the first template argument is fixed to int
template<typename V>
struct S<int, V> {
    double another_value;
    int foo(double arg) { // Do something}
};

```

6

```
// Common case:  
template<typename T, typename U>  
struct S {  
    T t_val;  
    U u_val;  
};  
  
// Special case when the first template argument is fixed to int  
template<typename V>  
struct S<int, V> {  
    double another_value;  
    int foo(double arg) { // Do something}  
};
```

```
// Common case:  
template<typename T, typename U>  
struct S {  
    T t_val;  
    U u_val;  
};  
  
// Special case when the first template argument is fixed to int  
template<typename V>  
struct S<int, V> {  
    double another_value;  
    int foo(double arg) { // Do something}  
};
```

```
// Common case:  
template<typename T, typename U>  
struct S {  
    T t_val;  
    U u_val;  
};  
  
// Special case when the first template argument is fixed to int  
template<typename V>  
struct S<int, V> {  
    double another_value;  
    int foo(double arg) { // Do something}  
};
```

```
// Common case:  
template<typename T, typename U>  
struct S {  
    T t_val;  
    U u_val;  
};  
  
// Special case when the first template argument is fixed to int  
template<typename V>  
struct S<int, V> {  
    double another_value;
```

```
int foo(double arg) { // Do something}
};
```

◦ ◦ ◦

```
template <class T, size_t N = 10>
struct my_array {
    T arr[N];
};

int main() {
    /* Default parameter is ignored, N = 5 */
    my_array<int, 5> a;

    /* Print the length of a.arr: 5 */
    std::cout << sizeof(a.arr) / sizeof(int) << std::endl;

    /* Last parameter is omitted, N = 10 */
    my_array<int> b;

    /* Print the length of a.arr: 10 */
    std::cout << sizeof(b.arr) / sizeof(int) << std::endl;
}
```

## C++11

```
template<typename T> using pointer = T*;
```

pointer<T>T\*

```
template<typename T> using pointer = T*;
```

◦

```
template<typename T> using pointer = T*;
```

◦ ◦

```
template <class T>
struct Tag1 { };

template <class T>
struct Tag2 { };

template <template <class> class Tag>
struct IntTag {
    typedef Tag<int> type;
};

int main() {
    IntTag<Tag1>::type t;
}
```

## C++11

```

template <class T>
struct Tag1 { };

template <class T>
struct Tag2 { };

template <template <class> class Tag>
struct IntTag {
    typedef Tag<int> type;
};

int main() {
    IntTag<Tag1>::type t;
}

```

## auto

### C++ 17.

```

template <class T, T N>
struct integral_constant {
    using type = T;
    static constexpr T value = N;
};

using five = integral_constant<int, 5>;

```

`decltype(expr), expr decltype(auto)`

### C++ 17

```

template <class T, T N>
struct integral_constant {
    using type = T;
    static constexpr T value = N;
};

using five = integral_constant<int, 5>;

```

## unique\_ptr

### unique\_ptr C API -

```

template <class T, T N>
struct integral_constant {
    using type = T;
    static constexpr T value = N;
};

using five = integral_constant<int, 5>;

```

`Tunique_ptr`

```
template <class T, T N>
```

```

struct integral_constant {
    using type = T;
    static constexpr T value = N;
};

using five = integral_constant<int, 5>;

```

- 
- 
- 
- 
- std::nullptr\_t •

## Template Argument Deduction◦

```

#include <iostream>

template<typename T, std::size_t size>
std::size_t size_of(T (&anArray)[size]) // Pass array by reference. Requires.
{                                         // an exact size. We allow all sizes
    return size;                         // by using a template "size".
}

int main()
{
    char anArrayOfChar[15];
    std::cout << "anArrayOfChar: " << size_of(anArrayOfChar) << "\n";

    int anArrayOfData[] = {1,2,3,4,5,6,7,8,9};
    std::cout << "anArrayOfData: " << size_of(anArrayOfData) << "\n";
}

```

```

#include <iostream>

template<typename T, std::size_t size>
std::size_t size_of(T (&anArray)[size]) // Pass array by reference. Requires.
{                                         // an exact size. We allow all sizes
    return size;                         // by using a template "size".
}

int main()
{
    char anArrayOfChar[15];
    std::cout << "anArrayOfChar: " << size_of(anArrayOfChar) << "\n";

    int anArrayOfData[] = {1,2,3,4,5,6,7,8,9};
    std::cout << "anArrayOfData: " << size_of(anArrayOfData) << "\n";
}

```

◦

## C ++ 14

- std::tuple ◦ std::tuple◦

```

template<typename ... T>

```

```

struct DataStructure {};

DataStructure<> data;

template<typename ... T>
struct DataStructure {};

```

DataStructure<int, float, std::string> data(1, 2.1, "hello") .

T Rest . T first . DataStructure<Rest ... > rest . rest .

DataStructure<int, float> data . int firstDataStructure<float> rest . rest . float first

DataStructure<> rest . rest .

```

template<typename ... T>
struct DataStructure {};

```

DataStructure<int, float, std::string> data . DataStructure<int, float, std::string> data .

data . rest . rest . first . get . get

```

template<typename ... T>
struct DataStructure {};

```

get = data . get<1> () . std::tuple . GetHelper . DataStructure::get . idx = . C++14 . auto .

. .

```

template<typename ... T>
struct DataStructure {};

```

idx == 0 . first

```

template<typename ... T>
struct DataStructure {};

```

idx . rest . GetHelper

```

template<typename ... T>
struct DataStructure {};

```

DataStructure<int, float> data . data . get<1> () . GetHelper<1, DataStructure<int, float>>::get (data)

GetHelper<0, DataStructure<float>>::get (data . rest) . idx = 0 . data . rest . first .

main

```

template<typename ... T>
struct DataStructure {};

```

◦ ◦ ◦

```
// print_string.h
template <class T>
void print_string(const T* str);

// print_string.cpp
#include "print_string.h"
template void print_string(const char* );
template void print_string(const wchar_t* );
```

print\_string<char>print\_string<wchar\_t>print\_string.cppprint\_string◦ ◦

## C++ 11

extern ◦ ◦ TU◦

foo.h

```
// print_string.h
template <class T>
void print_string(const T* str);

// print_string.cpp
#include "print_string.h"
template void print_string(const char* );
template void print_string(const wchar_t* );
```

foo.cpp

```
// print_string.h
template <class T>
void print_string(const T* str);

// print_string.cpp
#include "print_string.h"
template void print_string(const char* );
template void print_string(const wchar_t* );
```

main.cpp

```
// print_string.h
template <class T>
void print_string(const T* str);

// print_string.cpp
#include "print_string.h"
template void print_string(const char* );
template void print_string(const wchar_t* );
```

<https://riptutorial.com/zh-TW/cplusplus/topic/460/>

autoo

## C ++ 98C。 C ++

```
int main()
{
    auto int i = 5; // removing auto has no effect
}
```

◦

## Examples

autoo

```
std::map< std::string, std::shared_ptr< Widget > > table;
// C++98
std::map< std::string, std::shared_ptr< Widget > >::iterator i = table.find( "42" );
// C++11/14/17
auto j = table.find( "42" );
```

## for

```
std::map< std::string, std::shared_ptr< Widget > > table;
// C++98
std::map< std::string, std::shared_ptr< Widget > >::iterator i = table.find( "42" );
// C++11/14/17
auto j = table.find( "42" );
```

## lambdas

```
std::map< std::string, std::shared_ptr< Widget > > table;
// C++98
std::map< std::string, std::shared_ptr< Widget > >::iterator i = table.find( "42" );
// C++11/14/17
auto j = table.find( "42" );
```

```
std::map< std::string, std::shared_ptr< Widget > > table;
// C++98
std::map< std::string, std::shared_ptr< Widget > >::iterator i = table.find( "42" );
// C++11/14/17
auto j = table.find( "42" );
```

```
std::map< std::string, std::shared_ptr< Widget > > table;
// C++98
std::map< std::string, std::shared_ptr< Widget > >::iterator i = table.find( "42" );
// C++11/14/17
auto j = table.find( "42" );
```

```
std::pair<const int, float>

auto

auto mult(int c) {
    return c * std::valarray<int>{1};
}

auto v = mult(3);
std::cout << v[0]; // some value that could be, but almost certainly is not, 3.
```

valarrayoperator\*valarray auto. multstd::valarray<int> 3.

## autoconstreferences

autointchar. const&const. .

sstd::string fors .

```
std::vector<std::string> strings = { "stuff", "things", "misc" };
for(auto s : strings) {
    std::cout << s << std::endl;
}
```

s s.append(" and stuff") strings.

sauto&std::string&

```
std::vector<std::string> strings = { "stuff", "things", "misc" };
for(auto s : strings) {
    std::cout << s << std::endl;
}
```

sstrings.

sconst auto& constconst

```
std::vector<std::string> strings = { "stuff", "things", "misc" };
for(auto s : strings) {
    std::cout << s << std::endl;
}
```

sconst.

forautoconst auto&.

auto

```
auto main() -> int {}
```

```
auto main() -> int {}
```

```
decltype std::declval<T>
```

```
auto main() -> int {}
```

## lambdaC ++ 14

### C ++ 14

#### C ++ 14 lambda

```
auto print = [](const auto& arg) { std::cout << arg << std::endl; };  
  
print(42);  
print("hello world");
```

#### lambda

```
auto print = [](const auto& arg) { std::cout << arg << std::endl; };  
  
print(42);  
print("hello world");
```

```
auto print = [](const auto& arg) { std::cout << arg << std::endl; };  
  
print(42);  
print("hello world");
```

```
auto o o
```

```
std::vector<bool> flags{true, true, false};  
auto flag = flags[0];  
flags.push_back(true);
```

```
flag bool std::vector<bool>::reference vector<bool> operator [] operator bool operator bool
```

```
flags.push_back(true) o
```

```
std::vector<bool> flags{true, true, false};  
auto flag = flags[0];  
flags.push_back(true);
```

```
vector<flag> foo o
```

```
auto
```

```
std::vector<bool> flags{true, true, false};  
auto flag = flags[0];  
flags.push_back(true);
```

```
bool auto o
```

o o

<https://riptutorial.com/zh-TW/cplusplus/topic/2421/>

# 96:

```
operator >>operator << o  
  
#include <iomanip> o  
  
1. os.width(n);os << std::setw(n);  
is.width(n);is >> std::setw(n);  
  
2. os.precision(n);os << std::setprecision(n);  
is.precision(n);is >> std::setprecision(n);  
  
3. os.fill(c);os << std::setfill(c);  
  
4. str >> std::setbase(base);str << std::setbase(base);
```

```
str.setf(base == 8 ? std::ios_base::oct :  
         base == 10 ? std::ios_base::dec :  
         base == 16 ? std::ios_base::hex :  
                     std::ios_base::fmtflags(0),  
         std::ios_base::basefield);
```

```
5. os.setf(std::ios_base::flag);os << std::flag;  
is.setf(std::ios_base::flag);is >> std::flag;  
os.unsetf(std::ios_base::flag);os << std::no ## flag;  
is.unsetf(std::ios_base::flag);is >> std::no ## flag;  
## -  
flag S boolalpha showbase showpoint showpos skipws uppercase.
```

```
6. std::ios_base::basefield o  
flag S dec hexoct  
• os.setf(std::ios_base::flag, std::ios_base::basefield);os << std::flag;  
is.setf(std::ios_base::flag, std::ios_base::basefield);is >> std::flag;  
1  
• str.unsetf(std::ios_base::flag, std::ios_base::basefield);  
str.setf(std::ios_base::fmtflags(0), std::ios_base::basefield);  
2
```

```
7. std::ios_base::adjustfield
```

- 
- flag **S left rightinternal**
- os.setf(std::ios\_base::flag, std::ios\_base::adjustfield);os << std::flag;  
is.setf(std::ios\_base::flag, std::ios\_base::adjustfield);is >> std::flag;
- 1**
- str.unsetf(std::ios\_base::flag, std::ios\_base::adjustfield);  
str.setf(std::ios\_base::fmtflags(0), std::ios\_base::adjustfield);
- 2**

**1**unsetf◦

**2**flag◦

#### 8. **std::ios\_base::floatfield**◦

- os.setf(std::ios\_base::flag, std::ios\_base::floatfield);os << std::flag;  
is.setf(std::ios\_base::flag, std::ios\_base::floatfield);is >> std::flag;
- flag **S fixedscientific**◦
- os.setf(std::ios\_base::fmtflags(0), std::ios\_base::floatfield);os << std::defaultfloat;  
is.setf(std::ios\_base::fmtflags(0), std::ios\_base::floatfield);is >> std::defaultfloat;

#### 9. str.setf(std::ios\_base::fmtflags(0), std::ios\_base::flag);str.unsetf(std::ios\_base::flag) flag **S basefield adjustfield floatfield**◦

#### 10. os.setf(mask)os << setiosflags(mask); is.setf(mask) is >> setiosflags(mask); os.unsetf(mask)os << resetiosflags(mask); is.unsetf(mask) is >> resetiosflags(mask); std::ios\_base::fmtflagsmask◦

## Examples

```
std::boolalphastd::noboolalpha = °
```

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

```
std::showbase std::noshowbase - o

std::dec std::hex std::oct - o

std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

```
std::ios_base::noshowbase std::ios_base::dec o

std::istringstream< sstream >o
```

```
std::uppercase std::nouppercase - o o

std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0

std::nouppercase o
```

```
std::setw(n) - /n o
```

```
width n o
```

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
```

```

        << "\"" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0

std::setw(0) .

std::left std::right std::internal - std::ios_base::adjustfield std::ios_base::left
std::ios_base::right std::ios_base::internal std::ios_base::internal o std::left std::right
std::internal - o .

std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \"\" << std::boolalpha << boolValue
        << "\"" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0

```

std::left .

**std::fixed std::scientific std::hexfloat [C++ 11] std::defaultfloat [C++ 11] - /o**

```

std::fixed std::ios_base::floatfield std::ios_base::fixed
std::scientific - to std::ios_base::scientific
std::hexfloat - std::ios_base::fixed | std::ios_base::scientific
std::defaultfloat - std::ios_base::fmtflags(0) .

```

#### fmtflags

```

std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \"\" << std::boolalpha << boolValue
        << "\"" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0

std::ios_base::fmtflags(0) .

```

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

std::showpointstd::noshowpoint - . .

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

std::showpoint .

std::showposstd::noshowpos - +. .

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

std::noshowpos.

std::unitbuf std::nounitbuf - . . std::unitbuf.

```
std::setbase(base) - o

std::setbase(8) std::ios_base::basefieldstd::ios_base::oct
std::setbase(16) - std::ios_base::hex
std::setbase(10) - std::ios_base::dec .

base8 1016std::ios_base::basefieldstd::ios_base::fmtflags(0)。

std::ios_base::basefieldstd::ios_base::dec std::setbase(10) 。
```

```
std::setprecision(n) - o

std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

```
std::setprecision(6) 。
```

```
std::setiosflags(mask) std::resetiosflags(mask) - std::ios_base::fmtflagsmasksetclear。
```

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

```
std::skipwsstd::noskipws - o 。
```

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
```

```
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

```
std::ios_base::skipws °
```

```
std::quoted(s[, delim[, escape]]) [C++ 14] - °
```

```
s = °
delim = °
escape = \ °
```

```
std::cout << std::boolalpha << 1;
// Output: true

std::cout << std::noboolalpha << false;
// Output: 0

bool boolValue;
std::cin >> std::boolalpha >> boolValue;
std::cout << "Value \" " << std::boolalpha << boolValue
    << "\" was parsed as " << std::noboolalpha << boolValue;
// Input: true
// Output: Value "true" was parsed as 0
```

```
°
```

```
std::ends = '\0' °
```

```
template <class charT, class traits>
std::basic_ostream<charT, traits>& ends(std::basic_ostream<charT, traits>& os);
```

```
os.put(charT())
os << std::ends;
```

```
std::endl std::flush out.flush() out. ° std::endl '\n' °
```

```
template <class charT, class traits>
std::basic_ostream<charT, traits>& ends(std::basic_ostream<charT, traits>& os);
```

```
std::setfill(c) = c ° std::setw °
```

```
template <class charT, class traits>
std::basic_ostream<charT, traits>& ends(std::basic_ostream<charT, traits>& os);
```

**std::put\_money(mon[, intl]) [C ++ 11]**  $\circ$  out << std::put\_money(mon, intl) mon long double  
std::basic\_string std::money\_put facet out. intltrue.

```
template <class charT, class traits>
std::basic_ostream<charT, traits>& ends(std::basic_ostream<charT, traits>& os);
```

**std::put\_time(tmb, fmt) [C ++ 11]** - fmt/std::tm.

tmb - localtime() gmtime() const std::tm\*.  
fmt - nullconst CharT\*.

```
template <class charT, class traits>
std::basic_ostream<charT, traits>& ends(std::basic_ostream<charT, traits>& os);
```

$\circ$

std::ws - . std::skipws.

```
#include <sstream>
...
std::string str;
std::istringstream(" \v\n\r\t Wow!There is no whitespaces!") >> std::ws >> str;
std::cout << str;
// Output: Wow!There is no whitespaces!
```

**std::get\_money(mon[, intl]) [C ++ 11]**  $\circ$  in >> std::get\_money(mon, intl) std::money\_get in mon long  
double std::basic\_string. intltrue.

```
#include <sstream>
...
std::string str;
std::istringstream(" \v\n\r\t Wow!There is no whitespaces!") >> std::ws >> str;
std::cout << str;
// Output: Wow!There is no whitespaces!
```

**std::get\_time(tmb, fmt) [C ++ 11]** - fmt tmb.

tmb - const std::tm\*.

```
fmt - nullconst CharT*o

#include <sstream>
...
std::string str;
std::istringstream(" \v\n\r\t    Wow!There    is no whitespaces!") >> std::ws >> str;
std::cout << str;
// Output: Wow!There    is no whitespaces!
```

◦

<https://riptutorial.com/zh-TW/cplusplus/topic/10699/>

◦

## Examples

switchcase◦ switchcase◦

```
char c = getchar();
bool confirmed;
switch (c) {
    case 'y':
        confirmed = true;
        break;
    case 'n':
        confirmed = false;
        break;
    default:
        std::cout << "invalid response!\n";
        abort();
}
```

## C ++

switch◦  
switchcasedefault◦ switchcasedefault◦

◦

```
char c = getchar();
bool confirmed;
switch (c) {
    case 'y':
        confirmed = true;
        break;
    case 'n':
        confirmed = false;
        break;
    default:
        std::cout << "invalid response!\n";
        abort();
}
```

catch◦ catch ...◦ ◦

```
try {
    std::vector<int> v(N);
    // do something
} catch (const std::bad_alloc&) {
    std::cout << "failed to allocate memory for vector!" << std::endl;
} catch (const std::runtime_error& e) {
```

```
    std::cout << "runtime error: " << e.what() << std::endl;
} catch (...) {
    std::cout << "unexpected exception!" << std::endl;
    throw;
}
```

## switchcase◦

```
char c = getchar();
bool confirmed;
switch (c) {
    case 'y':
        confirmed = true;
        break;
    case 'n':
        confirmed = false;
        break;
    default:
        std::cout << "invalid response!\n";
        abort();
}
```

## C ++ 11

◦

```
char c = getchar();
bool confirmed;
switch (c) {
    case 'y':
        confirmed = true;
        break;
    case 'n':
        confirmed = false;
        break;
    default:
        std::cout << "invalid response!\n";
        abort();
}
```

## if◦ if◦ ◦

```
int x;
std::cout << "Please enter a positive number." << std::endl;
std::cin >> x;
if (x <= 0) {
    std::cout << "You didn't enter a positive number!" << std::endl;
    abort();
}
```

## ifelse◦ else◦

```
int x;
std::cin >> x;
if (x%2 == 0) {
```

```
    std::cout << "The number is even\n";
} else {
    std::cout << "The number is odd\n";
}
```

◦

```
bool f(int arg) {
    bool result = false;
    hWidget widget = get_widget(arg);
    if (!g()) {
        // we can't continue, but must do cleanup still
        goto end;
    }
    // ...
    result = true;
end:
    release_widget(widget);
    return result;
}
```

◦

return◦

```
int f() {
    return 42;
}
int x = f(); // x is 42
int g() {
    return 3.14;
}
int y = g(); // y is 3
```

returnvoid◦ void void -returning◦

```
int f() {
    return 42;
}
int x = f(); // x is 42
int g() {
    return 3.14;
}
int y = g(); // y is 3
```

mainstd::exit◦ mainstd::exit◦

```
int f() {
    return 42;
}
int x = f(); // x is 42
int g() {
    return 3.14;
}
int y = g(); // y is 3
```

1. throw。

```
void print_asterisks(int count) {
    if (count < 0) {
        throw std::invalid_argument("count cannot be negative!");
    }
    while (count--) { putchar('*'); }
}
```

2. throw。 std::terminate。

```
void print_asterisks(int count) {
    if (count < 0) {
        throw std::invalid_argument("count cannot be negative!");
    }
    while (count--) { putchar('*'); }
}
```

3. throwthrow。

```
void print_asterisks(int count) {
    if (count < 0) {
        throw std::invalid_argument("count cannot be negative!");
    }
    while (count--) { putchar('*'); }
}
```

C++ 11。

throw。 **throw** void。

```
void print_asterisks(int count) {
    if (count < 0) {
        throw std::invalid_argument("count cannot be negative!");
    }
    while (count--) { putchar('*'); }
}
```

try。 **trycatch**。 trytrycatch。

```
std::vector<int> v(N);      // if an exception is thrown here,
                            // it will not be caught by the following catch block
try {
    std::vector<int> v(N); // if an exception is thrown here,
                            // it will be caught by the following catch block
    // do something with v
} catch (const std::bad_alloc&) {
    // handle bad_alloc exceptions from the try block
}
```

**ifif..else**

**ifelse**

## truefalse

```
if (condition) statement
```

## C ++/

```
if (condition) statement
```

```
if (condition) statement
```

```
if (condition) statement
```

## **if / ifelse / else**

```
if (condition) statement
```

```
if (condition) statement
```

## “ switch ”

break continue goto exit。

break。

```
break;
```

switch break i。

```
break;
```

1111231break

```
break;
```

◦

break if while for ;

```
break;
```

continue。

```
break;
```

```
break;
```

```
break;
```

```
i%2==0 continue@i;
```

```
for (initialisation; condition; increment/decrement)
{
    ...
    if (True Condition)
        continue;
    ...
}
```

Continues Loop with the Next Value

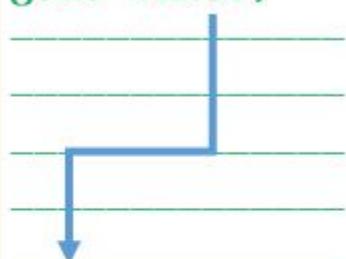
## goto

```
.. goto .. :)
```

```
break;
```

```
goto ..
```

goto label;



label :

```
statement 1;  
statement 2;  
statement 3;
```

label :

```
statement 1;  
statement 2;  
statement 3;
```

```
goto label;
```

Forward Reference

Backward Reference

```
break;
```

```
break;
```

```
num%2==0 goto do-while.
```

```
exitcstdlib. exit.
```

```
break;
```

cstdlibEXIT\_SUCCESS EXIT\_FAILURE。

<https://riptutorial.com/zh-TW/cplusplus/topic/7837/>

## Examples

```
float total = 0;
for(float a = 0; a != 2; a += 0.01f) {
    total += a;
}
```

0, 0.01, 0.02, 0.03, ..., 1.97, 1.98, 1.99 199 -the.

1. a2.
2. a < 2 199 IEEE754<sub>201</sub>.

.

```
float total = 0;
for(float a = 0; a != 2; a += 0.01f) {
    total += a;
}
```

base10. 0.1 0.2 0.310 --10-2-1/3100.3333333333333333...

```
float total = 0;
for(float a = 0; a != 2; a += 0.01f) {
    total += a;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/5115/>

99:

## Examples

Base. . .

virtual . .

```
class Base {
public:
    virtual ~Base() = default;

private:
    // data members etc.
};

class Derived : public Base { // models Is-A relationship
public:
    // some methods

private:
    // more data members
};

// virtual destructor in Base ensures that derived destructors
// are also called when the object is destroyed
std::unique_ptr<Base> base = std::make_unique<Derived>();
base = nullptr; // safe, doesn't leak Derived's members
```

protected.

```
class Base {
public:
    virtual ~Base() = default;

private:
    // data members etc.
};

class Derived : public Base { // models Is-A relationship
public:
    // some methods

private:
    // more data members
};

// virtual destructor in Base ensures that derived destructors
// are also called when the object is destroyed
std::unique_ptr<Base> base = std::make_unique<Derived>();
base = nullptr; // safe, doesn't leak Derived's members
```

○

```

private° virtual° protected°

° final ° °

class Base {
public:
    virtual ~Base() = default;

private:
    //      data members etc.
};

class Derived : public Base { //  models Is-A relationship
public:
    //      some methods

private:
    //      more data members
};

//      virtual destructor in Base ensures that derived destructors
//      are also called when the object is destroyed
std::unique_ptr<Base> base = std::make_unique<Derived>();
base = nullptr; //      safe, doesn't leak Derived's members

```

° °  
°

```

class Movable {
public:
    virtual ~Movable() noexcept = default;

    //      compiler won't generate these unless we tell it to
    //      because we declared a destructor
    Movable(Movable&&) noexcept = default;
    Movable& operator=(Movable&&) noexcept = default;

    //      declaring move operations will suppress generation
    //      of copy operations unless we explicitly re-enable them
    Movable(const Movable&) = default;
    Movable& operator=(const Movable&) = default;
};

```

## Rule of Three / Five / Zero °

```

person(const person &other)
: name(new char[strlen(other.name) + 1])
, age(other.age)
{
    std::strcpy(name, other.name);
}

person& operator=(person const& rhs) {
    if (this != &other) {
        delete [] name;
}

```

```

        name = new char[std::strlen(other.name) + 1];
        std::strcpy(name, other.name);
        age = other.age;
    }

    return *this;
}

```

◦ -new[] this◦ ◦ ◦

```

person(const person &other)
: name(new char[std::strlen(other.name) + 1])
, age(other.age)
{
    std::strcpy(name, other.name);
}

person& operator=(person const& rhs) {
    if (this != &other) {
        delete [] name;
        name = new char[std::strlen(other.name) + 1];
        std::strcpy(name, other.name);
        age = other.age;
    }

    return *this;
}

```

```

person(const person &other)
: name(new char[std::strlen(other.name) + 1])
, age(other.age)
{
    std::strcpy(name, other.name);
}

person& operator=(person const& rhs) {
    if (this != &other) {
        delete [] name;
        name = new char[std::strlen(other.name) + 1];
        std::strcpy(name, other.name);
        age = other.age;
    }

    return *this;
}

```

p2rhs◦ operator=p1◦ \*thisrhsrhs◦ operator= this◦ -◦

## C++11

◦

```

person(const person &other)
: name(new char[std::strlen(other.name) + 1])
, age(other.age)
{
    std::strcpy(name, other.name);
}

```

```

}

person& operator=(person const& rhs) {
    if (this != &other) {
        delete [] name;
        name = new char[strlen(other.name) + 1];
        std::strcpy(name, other.name);
        age = other.age;
    }

    return *this;
}

```

p2rhs . .

```

. .

class C{
    int i;
public:
    // the default constructor definition
    C()
    : i(0){ // member initializer list -- initialize i to 0
        // constructor function body -- can do more complex things here
    }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

```

"

```

class C{
    int i;
public:
    // the default constructor definition
    C()
    : i(0){ // member initializer list -- initialize i to 0
        // constructor function body -- can do more complex things here
    }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

```

```

class C{
    int i;

```

```

public:
    // the default constructor definition
    C()
    : i(0){ // member initializer list -- initialize i to 0
        // constructor function body -- can do more complex things here
    }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

```

```

class C{
    int i;
public:
    // the default constructor definition
    C()
    : i(0){ // member initializer list -- initialize i to 0
        // constructor function body -- can do more complex things here
    }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

```

## C++ 11

private . .

.

## C++ 11

### C++ 11<sub>delete</sub>

```

class C{
    int i;
public:
    // the default constructor definition
    C()
    : i(0){ // member initializer list -- initialize i to 0
        // constructor function body -- can do more complex things here
    }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

```

```

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

.

class C{
    int i;
public:
    // the default constructor definition
    C()
    : i(0){ // member initializer list -- initialize i to 0
        // constructor function body -- can do more complex things here
    }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

```

## C++ 14

<type\_traits> std::is\_default\_constructible

```

class C{
    int i;
public:
    // the default constructor definition
    C()
    : i(0){ // member initializer list -- initialize i to 0
        // constructor function body -- can do more complex things here
    }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

```

## C++ 11

**C++ 11** std::is\_default\_constructible

```

class C{
    int i;
public:
    // the default constructor definition
    C()
    : i(0){ // member initializer list -- initialize i to 0
        // constructor function body -- can do more complex things here
    }
};

```

```

    }
};

C c1; // calls default constructor of C to create object c1
C c2 = C(); // calls default constructor explicitly
C c3(); // ERROR: this intuitive version is not possible due to "most vexing parse"
C c4{}; // but in C++11 {} CAN be used in a similar way

C c5[2]; // calls default constructor for both array elements
C* c6 = new C[2]; // calls default constructor for both array elements

```

◦ ~◦

```

class C{
    int* is;
    string s;
public:
    C()
    : is( new int[10] ){
    }

    ~C(){ // destructor definition
        delete[] is;
    }
};

class C_child : public C{
    string s_ch;
public:
    C_child(){}
    ~C_child(){} // child destructor
};

void f(){
    C c1; // calls default constructor
    C c2[2]; // calls default constructor for both elements
    C* c3 = new C[2]; // calls default constructor for both array elements

    C_child c_ch; // when destructed calls destructor of s_ch and of C base (and in turn s)

    delete[] c3; // calls destructors on c3[0] and c3[1]
} // automatic variables are destroyed here -- i.e. c1, c2 and c_ch

```

```

class C{
    int* is;
    string s;
public:
    C()
    : is( new int[10] ){

    }

    ~C(){ // destructor definition
        delete[] is;
    }
};

class C_child : public C{
    string s_ch;
public:

```

```

C_child(){}
~C_child(){} // child destructor
};

void f(){
    C c1; // calls default constructor
    C c2[2]; // calls default constructor for both elements
    C* c3 = new C[2]; // calls default constructor for both array elements

    C_child c_ch; // when destructed calls destructor of s_ch and of C base (and in turn s)

    delete[] c3; // calls destructors on c3[0] and c3[1]
} // automatic variables are destroyed here -- i.e. c1, c2 and c_ch

```

```

class C{
    int* is;
    string s;
public:
    C()
        : is( new int[10] ){
    }

    ~C(){ // destructor definition
        delete[] is;
    }
};

class C_child : public C{
    string s_ch;
public:
    C_child(){}
    ~C_child(){} // child destructor
};

void f(){
    C c1; // calls default constructor
    C c2[2]; // calls default constructor for both elements
    C* c3 = new C[2]; // calls default constructor for both array elements

    C_child c_ch; // when destructed calls destructor of s_ch and of C base (and in turn s)

    delete[] c3; // calls destructors on c3[0] and c3[1]
} // automatic variables are destroyed here -- i.e. c1, c2 and c_ch

```

## C++ 11

### C++ 11.

```

class C{
    int* is;
    string s;
public:
    C()
        : is( new int[10] ){

    }

    ~C(){ // destructor definition

```

```

        delete[] is;
    }
};

class C_child : public C{
    string s_ch;
public:
    C_child(){}
    ~C_child(){} // child destructor
};

void f(){
    C c1; // calls default constructor
    C c2[2]; // calls default constructor for both elements
    C* c3 = new C[2]; // calls default constructor for both array elements

    C_child c_ch; // when destructed calls destructor of s_ch and of C base (and in turn s)

    delete[] c3; // calls destructors on c3[0] and c3[1]
} // automatic variables are destroyed here -- i.e. c1, c2 and c_ch

```

◦

```

class C{
    int* is;
    string s;
public:
    C()
    : is( new int[10] ){
    }

    ~C(){ // destructor definition
        delete[] is;
    }
};

class C_child : public C{
    string s_ch;
public:
    C_child(){}
    ~C_child(){} // child destructor
};

void f(){
    C c1; // calls default constructor
    C c2[2]; // calls default constructor for both elements
    C* c3 = new C[2]; // calls default constructor for both array elements

    C_child c_ch; // when destructed calls destructor of s_ch and of C base (and in turn s)

    delete[] c3; // calls destructors on c3[0] and c3[1]
} // automatic variables are destroyed here -- i.e. c1, c2 and c_ch

```

## C++ 11

<type\_traits> std::is\_destructible

```
class C{
```

```

int* is;
string s;
public:
C()
: is( new int[10] ){
}

~C(){ // destructor definition
    delete[] is;
}
};

class C_child : public C{
    string s_ch;
public:
C_child(){}
~C_child(){} // child destructor
};

void f(){
    C c1; // calls default constructor
    C c2[2]; // calls default constructor for both elements
    C* c3 = new C[2]; // calls default constructor for both array elements

    C_child c_ch; // when destructed calls destructor of s_ch and of C base (and in turn s)

    delete[] c3; // calls destructors on c3[0] and c3[1]
} // automatic variables are destroyed here -- i.e. c1, c2 and c_ch

```

<https://riptutorial.com/zh-TW/cplusplus/topic/1476/>

# 100: C ++

C++

1

## Examples

## Observer Pattern.

9

Gamma™.

```
#include <iostream>
#include <vector>

class Subject;

class Observer
{
public:
    virtual ~Observer() = default;
    virtual void Update(Subject&) = 0;
};

class Subject
{
public:
    virtual ~Subject() = default;
    void Attach(Observer& o) { observers.push_back(&o); }
    void Detach(Observer& o)
    {
        observers.erase(std::remove(observers.begin(), observers.end(), &o));
    }
    void Notify()
    {
        for (auto* o : observers) {
            o->Update(*this);
        }
    }
private:
    std::vector<Observer*> observers;
};

class ClockTimer : public Subject
{
public:

    void SetTime(int hour, int minute, int second)
    {
        this->hour = hour;
        this->minute = minute;
        this->second = second;
    }
};
```

```

        Notify();
    }

    int GetHour() const { return hour; }
    int GetMinute() const { return minute; }
    int GetSecond() const { return second; }

private:
    int hour;
    int minute;
    int second;
};

class DigitalClock: public Observer
{
public:
    explicit DigitalClock(ClockTimer& s) : subject(s) { subject.Attach(*this); }
    ~DigitalClock() { subject.Detach(*this); }
    void Update(Subject& theChangedSubject) override
    {
        if (&theChangedSubject == &subject) {
            Draw();
        }
    }

    void Draw()
    {
        int hour = subject.GetHour();
        int minute = subject.GetMinute();
        int second = subject.GetSecond();

        std::cout << "Digital time is " << hour << ":"
            << minute << ":"
            << second << std::endl;
    }

private:
    ClockTimer& subject;
};

class AnalogClock: public Observer
{
public:
    explicit AnalogClock(ClockTimer& s) : subject(s) { subject.Attach(*this); }
    ~AnalogClock() { subject.Detach(*this); }
    void Update(Subject& theChangedSubject) override
    {
        if (&theChangedSubject == &subject) {
            Draw();
        }
    }

    void Draw()
    {
        int hour = subject.GetHour();
        int minute = subject.GetMinute();
        int second = subject.GetSecond();

        std::cout << "Analog time is " << hour << ":"
            << minute << ":"
            << second << std::endl;
    }
}

```

```

private:
    ClockTimer& subject;
};

int main()
{
    ClockTimer timer;

    DigitalClock digitalClock(timer);
    AnalogClock analogClock(timer);

    timer.SetTime(14, 41, 36);
}

```

```

#include <iostream>
#include <vector>

class Subject;

class Observer
{
public:
    virtual ~Observer() = default;
    virtual void Update(Subject&) = 0;
};

class Subject
{
public:
    virtual ~Subject() = default;
    void Attach(Observer& o) { observers.push_back(&o); }
    void Detach(Observer& o)
    {
        observers.erase(std::remove(observers.begin(), observers.end(), &o));
    }
    void Notify()
    {
        for (auto* o : observers) {
            o->Update(*this);
        }
    }
private:
    std::vector<Observer*> observers;
};

class ClockTimer : public Subject
{
public:

    void SetTime(int hour, int minute, int second)
    {
        this->hour = hour;
        this->minute = minute;
        this->second = second;

        Notify();
    }

    int GetHour() const { return hour; }
    int GetMinute() const { return minute; }

```

```

int GetSecond() const { return second; }

private:
    int hour;
    int minute;
    int second;
};

class DigitalClock: public Observer
{
public:
    explicit DigitalClock(ClockTimer& s) : subject(s) { subject.Attach(*this); }
    ~DigitalClock() { subject.Detach(*this); }
    void Update(Subject& theChangedSubject) override
    {
        if (&theChangedSubject == &subject) {
            Draw();
        }
    }

    void Draw()
    {
        int hour = subject.GetHour();
        int minute = subject.GetMinute();
        int second = subject.GetSecond();

        std::cout << "Digital time is " << hour << ":"
            << minute << ":"
            << second << std::endl;
    }

private:
    ClockTimer& subject;
};

class AnalogClock: public Observer
{
public:
    explicit AnalogClock(ClockTimer& s) : subject(s) { subject.Attach(*this); }
    ~AnalogClock() { subject.Detach(*this); }
    void Update(Subject& theChangedSubject) override
    {
        if (&theChangedSubject == &subject) {
            Draw();
        }
    }

    void Draw()
    {
        int hour = subject.GetHour();
        int minute = subject.GetMinute();
        int second = subject.GetSecond();

        std::cout << "Analog time is " << hour << ":"
            << minute << ":"
            << second << std::endl;
    }

private:
    ClockTimer& subject;
};

int main()

```

```

{
    ClockTimer timer;

    DigitalClock digitalClock(timer);
    AnalogClock analogClock(timer);

    timer.SetTime(14, 41, 36);
}

```

1. DigitalClockAnalogClock **Subject** Attach() Detach() subject.Attach(\*this);  
subject.Detach(\*this);

2. vector<Observer\*> observers; °

3. **Observer**° Update() **Subject** Update(Subject &)

4. Attach() Detach() Notify()° Subject Subject::Attach (Observer&) void  
Subject::Detach(Observer&) void Subject::Notify()°

5. **Concrete**°

6. **Observer**° subject.Attach(\*this); °

7. °

QtObserver° ° Qt° /° /I / O° //QtMOC°

C° Qt°

° ° °

1. °

2. **Adapter**°

3. **Adaptee**°

4. STLpushvector :: push\_back°

```

#include <iostream>

// Desired interface (Target)
class Rectangle
{
public:
    virtual void draw() = 0;
};

// Legacy component (Adaptee)
class LegacyRectangle
{
public:
    LegacyRectangle(int x1, int y1, int x2, int y2) {
        x1_ = x1;
    }
}
```

```

    y1_ = y1;
    x2_ = x2;
    y2_ = y2;
    std::cout << "LegacyRectangle(x1,y1,x2,y2)\n";
}
void oldDraw() {
    std::cout << "LegacyRectangle: oldDraw().\n";
}
private:
    int x1_;
    int y1_;
    int x2_;
    int y2_;
};

// Adapter wrapper
class RectangleAdapter: public Rectangle, private LegacyRectangle
{
public:
    RectangleAdapter(int x, int y, int w, int h):
        LegacyRectangle(x, y, x + w, y + h) {
            std::cout << "RectangleAdapter(x,y,x+w,y+h)\n";
    }

    void draw() {
        std::cout << "RectangleAdapter: draw().\n";
        oldDraw();
    }
};

int main()
{
    int x = 20, y = 50, w = 300, h = 200;
    Rectangle *r = new RectangleAdapter(x,y,w,h);
    r->draw();
}

//Output:
//LegacyRectangle(x1,y1,x2,y2)
//RectangleAdapter(x,y,x+w,y+h)

```

## 1. Rectangle

## 2. Rectangle

```

#include <iostream>

// Desired interface (Target)
class Rectangle
{
public:
    virtual void draw() = 0;
};

// Legacy component (Adaptee)
class LegacyRectangle
{
public:
    LegacyRectangle(int x1, int y1, int x2, int y2) {

```

```

        x1_ = x1;
        y1_ = y1;
        x2_ = x2;
        y2_ = y2;
        std::cout << "LegacyRectangle(x1,y1,x2,y2)\n";
    }
    void oldDraw() {
        std::cout << "LegacyRectangle: oldDraw().\n";
    }
private:
    int x1_;
    int y1_;
    int x2_;
    int y2_;
};

// Adapter wrapper
class RectangleAdapter: public Rectangle, private LegacyRectangle
{
public:
    RectangleAdapter(int x, int y, int w, int h):
        LegacyRectangle(x, y, x + w, y + h) {
        std::cout << "RectangleAdapter(x,y,x+w,x+h)\n";
    }

    void draw() {
        std::cout << "RectangleAdapter: draw().\n";
        oldDraw();
    }
};

int main()
{
    int x = 20, y = 50, w = 300, h = 200;
    Rectangle *r = new RectangleAdapter(x,y,w,h);
    r->draw();
}

//Output:
//LegacyRectangle(x1,y1,x2,y2)
//RectangleAdapter(x,y,x+w,x+h)

```

### 3.◦

```

#include <iostream>

// Desired interface (Target)
class Rectangle
{
public:
    virtual void draw() = 0;
};

// Legacy component (Adaptee)
class LegacyRectangle
{
public:
    LegacyRectangle(int x1, int y1, int x2, int y2) {
        x1_ = x1;

```

```

        y1_ = y1;
        x2_ = x2;
        y2_ = y2;
        std::cout << "LegacyRectangle(x1,y1,x2,y2)\n";
    }
    void oldDraw() {
        std::cout << "LegacyRectangle: oldDraw().\n";
    }
private:
    int x1_;
    int y1_;
    int x2_;
    int y2_;
};

// Adapter wrapper
class RectangleAdapter: public Rectangle, private LegacyRectangle
{
public:
    RectangleAdapter(int x, int y, int w, int h):
        LegacyRectangle(x, y, x + w, y + h) {
        std::cout << "RectangleAdapter(x,y,x+w,x+h)\n";
    }

    void draw() {
        std::cout << "RectangleAdapter: draw().\n";
        oldDraw();
    }
};

int main()
{
    int x = 20, y = 50, w = 300, h = 200;
    Rectangle *r = new RectangleAdapter(x,y,w,h);
    r->draw();
}

//Output:
//LegacyRectangle(x1,y1,x2,y2)
//RectangleAdapter(x,y,x+w,x+h)

```

4. Adapter RectangleAdapter LegacyRectangle BOTH Rectangle draw()。

5. LegacyRectangle Rectangle draw() Adapter (RectangleAdapter) Rectangle LegacyRectangle oldDraw()

- 

```

#include <iostream>

// Desired interface (Target)
class Rectangle
{
public:
    virtual void draw() = 0;
};

// Legacy component (Adaptee)
class LegacyRectangle
{
public:

```

```

LegacyRectangle(int x1, int y1, int x2, int y2) {
    x1_ = x1;
    y1_ = y1;
    x2_ = x2;
    y2_ = y2;
    std::cout << "LegacyRectangle(x1,y1,x2,y2)\n";
}
void oldDraw() {
    std::cout << "LegacyRectangle: oldDraw().\n";
}
private:
    int x1_;
    int y1_;
    int x2_;
    int y2_;
};

// Adapter wrapper
class RectangleAdapter: public Rectangle, private LegacyRectangle
{
public:
    RectangleAdapter(int x, int y, int w, int h):
        LegacyRectangle(x, y, x + w, y + h) {
        std::cout << "RectangleAdapter(x,y,x+w,y+h)\n";
    }

    void draw() {
        std::cout << "RectangleAdapter: draw().\n";
        oldDraw();
    }
};

int main()
{
    int x = 20, y = 50, w = 300, h = 200;
    Rectangle *r = new RectangleAdapter(x,y,w,h);
    r->draw();
}

//Output:
//LegacyRectangle(x1,y1,x2,y2)
//RectangleAdapter(x,y,x+w,y+h)

```

◦ ◦ ◦

**API**◦ **API**◦

**Bridge**◦ **Bridge**◦ ◦

```

class Animal{
public:
    virtual std::shared_ptr<Animal> clone() const = 0;
    virtual std::string getname() const = 0;
};

class Bear: public Animal{
public:
    virtual std::shared_ptr<Animal> clone() const override
    {

```

```

        return std::make_shared<Bear>(*this);
    }
    virtual std::string getname() const override
    {
        return "bear";
    }
};

class Cat: public Animal{
public:
    virtual std::shared_ptr<Animal> clone() const override
    {
        return std::make_shared<Cat>(*this);
    }
    virtual std::string getname() const override
    {
        return "cat";
    }
};

class AnimalFactory{
public:
    static std::shared_ptr<Animal> getAnimal( const std::string& name )
    {
        if ( name == "bear" )
            return std::make_shared<Bear>();
        if ( name == "cat" )
            return std::shared_ptr<Cat>();

        return nullptr;
    }
};

```

## Fluent APIBuilder

Builder Pattern◦◦◦

[CEmail Builder](#)C ++◦ Email◦

```

#include <iostream>
#include <sstream>
#include <string>

using namespace std;

// Forward declaring the builder
class EmailBuilder;

class Email
{
public:
    friend class EmailBuilder; // the builder can access Email's privates

    static EmailBuilder make();

```

```

string to_string() const {
    stringstream stream;
    stream << "from: " << m_from
        << "\nto: " << m_to
        << "\nsubject: " << m_subject
        << "\nbody: " << m_body;
    return stream.str();
}

private:
    Email() = default; // restrict construction to builder

    string m_from;
    string m_to;
    string m_subject;
    string m_body;
};

class EmailBuilder
{
public:
    EmailBuilder& from(const string &from) {
        m_email.m_from = from;
        return *this;
    }

    EmailBuilder& to(const string &to) {
        m_email.m_to = to;
        return *this;
    }

    EmailBuilder& subject(const string &subject) {
        m_email.m_subject = subject;
        return *this;
    }

    EmailBuilder& body(const string &body) {
        m_email.m_body = body;
        return *this;
    }

    operator Email&&() {
        return std::move(m_email); // notice the move
    }
}

private:
    Email m_email;
};

EmailBuilder Email::make()
{
    return EmailBuilder();
}

// Bonus example!
std::ostream& operator <<(std::ostream& stream, const Email& email)
{
    stream << email.to_string();
    return stream;
}

```

```

int main()
{
    Email mail = Email::make().from("me@mail.com")
        .to("you@mail.com")
        .subject("C++ builders")
        .body("I like this API, don't you?");

    cout << mail << endl;
}

```

## C ++std::move&&。

operator Email&&()。◦ Email EmailBuilder::build() { ... }◦

---

## Builder Patternactor◦ actoractor◦

```

#include <iostream>
#include <sstream>
#include <string>

using namespace std;

// Forward declaring the builder
class EmailBuilder;

class Email
{
public:
    friend class EmailBuilder; // the builder can access Email's privates

    static EmailBuilder make();

    string to_string() const {
        stringstream stream;
        stream << "from: " << m_from
            << "\nto: " << m_to
            << "\nsubject: " << m_subject
            << "\nbody: " << m_body;
        return stream.str();
    }

private:
    Email() = default; // restrict construction to builder

    string m_from;
    string m_to;
    string m_subject;
    string m_body;
};

class EmailBuilder
{
public:
    EmailBuilder& from(const string &from) {
        m_email.m_from = from;
        return *this;
    }
}
```

```

}

EmailBuilder& to(const string &to) {
    m_email.m_to = to;
    return *this;
}

EmailBuilder& subject(const string &subject) {
    m_email.m_subject = subject;
    return *this;
}

EmailBuilder& body(const string &body) {
    m_email.m_body = body;
    return *this;
}

operator Email&&() {
    return std::move(m_email); // notice the move
}

private:
    Email m_email;
};

EmailBuilder Email::make()
{
    return EmailBuilder();
}

// Bonus example!
std::ostream& operator <<(std::ostream& stream, const Email& email)
{
    stream << email.to_string();
    return stream;
}

int main()
{
    Email mail = Email::make().from("me@mail.com")
        .to("you@mail.com")
        .subject("C++ builders")
        .body("I like this API, don't you?");

    cout << mail << endl;
}

```

o o

Email◦◦setter◦◦setter◦◦

C ++ <https://riptutorial.com/zh-TW/cplusplus/topic/4335/c-plusplus>

# 101:

## Examples

```
#include <iostream>

long double operator"" _km(long double val)
{
    return val * 1000.0;
}

long double operator"" _mi(long double val)
{
    return val * 1609.344;
}

int main()
{
    std::cout << "3 km = " << 3.0_km << " m\n";
    std::cout << "3 mi = " << 3.0_mi << " m\n";
    return 0;
}
```

```
#include <iostream>

long double operator"" _km(long double val)
{
    return val * 1000.0;
}

long double operator"" _mi(long double val)
{
    return val * 1609.344;
}

int main()
{
    std::cout << "3 km = " << 3.0_km << " m\n";
    std::cout << "3 mi = " << 3.0_mi << " m\n";
    return 0;
}
```

## C++ 14

```
namespace std::literals::chrono_literals literals chrono_literals。 using namespace std::literals
using namespace std::chrono_literals using namespace std::literals::chrono_literals。
```

```
#include <chrono>
#include <iostream>

int main()
{
    using namespace std::literals::chrono_literals;
```

```

    std::chrono::nanoseconds t1 = 600ns;
    std::chrono::microseconds t2 = 42us;
    std::chrono::milliseconds t3 = 51ms;
    std::chrono::seconds t4 = 61s;
    std::chrono::minutes t5 = 88min;
    auto t6 = 2 * 0.5h;

    auto total = t1 + t2 + t3 + t4 + t5 + t6;

    std::cout.precision(13);
    std::cout << total.count() << " nanoseconds" << std::endl; // 8941051042600 nanoseconds
    std::cout << std::chrono::duration_cast<std::chrono::hours>(total).count()
        << " hours" << std::endl; // 2 hours
}

```

## C++ 14

```

namespace std::literals::string_literals literals string_literals。 using namespace std::literals
using namespace std::string_literals using namespace std::literals::string_literals。

```

```

#include <codecvt>
#include <iostream>
#include <locale>
#include <string>

int main()
{
    using namespace std::literals::string_literals;

    std::string s = "hello world"s;
    std::u16string s16 = u"hello world"s;
    std::u32string s32 = U"hello world"s;
    std::wstring ws = L"hello world"s;

    std::cout << s << std::endl;

    std::wstring_convert<std::codecvt_utf8_utf16<char16_t>, char16_t> utf16conv;
    std::cout << utf16conv.to_bytes(s16) << std::endl;

    std::wstring_convert<std::codecvt_utf8_utf16<char32_t>, char32_t> utf32conv;
    std::cout << utf32conv.to_bytes(s32) << std::endl;

    std::wcout << ws << std::endl;
}

```

\0

```

#include <codecvt>
#include <iostream>
#include <locale>
#include <string>

int main()
{
    using namespace std::literals::string_literals;

    std::string s = "hello world"s;
    std::u16string s16 = u"hello world"s;

```

```

    std::u32string s32 = U"hello world"s;
    std::wstring ws = L"hello world"s;

    std::cout << s << std::endl;

    std::wstring_convert<std::codecvt_utf8_utf16<char16_t>, char16_t> utf16conv;
    std::cout << utf16conv.to_bytes(s16) << std::endl;

    std::wstring_convert<std::codecvt_utf8_utf16<char32_t>, char32_t> utf32conv;
    std::cout << utf32conv.to_bytes(s32) << std::endl;

    std::wcout << ws << std::endl;
}

```

## C++ 14

```

namespace std::literals::complex_literals literalscomplex_literals {
    using namespace std::literals::complex_literals using namespace
    std::literals::complex_literals .

```

```

#include <complex>
#include <iostream>

int main()
{
    using namespace std::literals::complex_literals;

    std::complex<double> c = 2.0 + 1i;           // {2.0, 1.}
    std::complex<float> cf = 2.0f + 1if;         // {2.0f, 1.f}
    std::complex<long double> cl = 2.0L + 1il; // {2.0L, 1.L}

    std::cout << "abs" << c << " = " << abs(c) << std::endl; // abs(2,1) = 2.23607
    std::cout << "abs" << cf << " = " << abs(cf) << std::endl; // abs(2,1) = 2.23607
    std::cout << "abs" << cl << " = " << abs(cl) << std::endl; // abs(2,1) = 2.23607
}

```

## C++ 14

```

int number = 0b0001'0101; // ==21

.

template< char FIRST, char... REST > struct binary
{
    static_assert( FIRST == '0' || FIRST == '1', "invalid binary digit" );
    enum { value = ( ( FIRST - '0' ) << sizeof...(REST) ) + binary<REST...>::value } ;
};

template<> struct binary<'0'> { enum { value = 0 } ; };
template<> struct binary<'1'> { enum { value = 1 } ; };

// raw literal operator
template< char... LITERAL > inline
constexpr unsigned int operator "" _b() { return binary<LITERAL...>::value ; }

// raw literal operator

```

```
template< char... LITERAL > inline
constexpr unsigned int operator "" _B() { return binary<LITERAL...>::value ; }

#include <iostream>

int main()
{
    std::cout << 10101_B << ", " << 011011000111_b << '\n' ; // prints 21, 1735
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2745/>

# 102: std :: integer\_sequence

```
std::integer_sequence<Type, Values...>Type Typeo . 。 “”。
```

## Examples

### std :: tuple

```
std::tuple<T...>o .
```

```
#include <array>
#include <iostream>
#include <string>
#include <tuple>
#include <utility>

// -----
// Example functions to be called:
void f(int i, std::string const& s) {
    std::cout << "f(" << i << ", " << s << ")" \n";
}
void f(int i, double d, std::string const& s) {
    std::cout << "f(" << i << ", " << d << ", " << s << ")" \n";
}
void f(char c, int i, double d, std::string const& s) {
    std::cout << "f(" << c << ", " << i << ", " << d << ", " << s << ")" \n";
}
void f(int i, int j, int k) {
    std::cout << "f(" << i << ", " << j << ", " << k << ")" \n";
}

// -----
// The actual function expanding the tuple:
template <typename Tuple, std::size_t... I>
void process(Tuple const& tuple, std::index_sequence<I...>) {
    f(std::get<I>(tuple)...);
}

// The interface to call. Sadly, it needs to dispatch to another function
// to deduce the sequence of indices created from std::make_index_sequence<N>
template <typename Tuple>
void process(Tuple const& tuple) {
    process(tuple, std::make_index_sequence<std::tuple_size<Tuple>::value>());
}

// -----
int main() {
    process(std::make_tuple(1, 3.14, std::string("foo")));
    process(std::make_tuple('a', 2, 2.71, std::string("bar")));
    process(std::make_pair(3, std::string("pair")));
    process(std::array<int, 3>{ 1, 2, 3 });
}
```

```
std::get<I>(object) std::tuple_size<T>::value process()o .
```

```

std::integer_sequence< " " >

#include <iostream>
#include <initializer_list>
#include <utility>

template <typename T, T... I>
void print_sequence(std::integer_sequence<T, I...>) {
    std::initializer_list<bool>{ bool(std::cout << I << ' ')... };
    std::cout << '\n';
}

template <int Offset, typename T, T... I>
void print_offset_sequence(std::integer_sequence<T, I...>) {
    print_sequence(std::integer_sequence<T, T(I + Offset)...>());
}

int main() {
    // explicitly specify sequences:
    print_sequence(std::integer_sequence<int, 1, 2, 3>());
    print_sequence(std::integer_sequence<char, 'f', 'o', 'o'>());

    // generate sequences:
    print_sequence(std::make_index_sequence<10>());
    print_sequence(std::make_integer_sequence<short, 10>());
    print_offset_sequence('A'(std::make_integer_sequence<char, 26>()));
}

```

print\_sequence () std::initializer\_list<bool>[array]◦

- gccclang gccvoid

```

#include <algorithm>
#include <array>
#include <iostream>
#include <iterator>
#include <string>
#include <utility>

template <typename T, std::size_t... I>
std::array<T, sizeof...(I)> make_array(T const& value, std::index_sequence<I...>) {
    return std::array<T, sizeof...(I)>{ (I, value)... };
}

template <int N, typename T>
std::array<T, N> make_array(T const& value) {
    return make_array(value, std::make_index_sequence<N>());
}

int main() {
    auto array = make_array<20>(std::string("value"));
    std::copy(array.begin(), array.end(),
              std::ostream_iterator<std::string>(std::cout, " "));
    std::cout << "\n";
}

```

std :: integer\_sequence <https://riptutorial.com/zh-TW/cplusplus/topic/8315/std---integer-sequence>

# 103: std :: string

◦ `stringchar`◦ `C ++ stringstd1998`◦

• //

```
std :: string s;
```

• //*const char \*c-string*

```
std :: string s“Hello”;
```

```
std :: string s = “”;
```

• //

```
std :: string s1“Hello”;
```

```
std :: string s2s1;
```

• //

```
std :: string s1“Hello”;
```

```
std :: string s2s1,0,4; //s104s2
```

• //

```
std :: string s1“Hello World”;
```

```
std :: string s2s1,5; //s15s2
```

• //*fillchar*

```
std :: string s5'a'; // saaaaa
```

• //

```
std :: string s1“Hello World”;
```

```
std :: string s2s1.begins1.begin+ 5; //s15s2
```

`std::stringstring iostream`//◦

---

`const char *nullptr`◦

```
std::string oops(nullptr);
std::cout << oops << "\n";
```

```
index >= size() atstd::out_of_range°  
operator[] index > size() index == size()
```

## C++ 11

1. **const** ;
2. **const**CharT() °.

## C++ 11

1. CharT() °.
2. °.

---

## C++ 14 "foo" "foo"s s const char\* "foo" std::string "foo" °

```
std::string_literalsstd::literalss °
```

## Examples

```
std::string::substr° °  
° (0, str.length()) ]
```

```
std::string str = "Hello foo, bar and world!";  
std::string newstr = str.substr(11); // "bar and world!"
```

```
°
```

```
std::string str = "Hello foo, bar and world!";  
std::string newstr = str.substr(11); // "bar and world!"
```

```
substr
```

```
std::string str = "Hello foo, bar and world!";  
std::string newstr = str.substr(11); // "bar and world!"
```

---

```
std::string replacestd::string °
```

```
replace
```

```
//Define string  
std::string str = "Hello foo, bar and world!";  
std::string alternate = "Hello foobar";  
  
//1)  
str.replace(6, 3, "bar"); // "Hello bar, bar and world!"
```

```
//2)
str.replace(str.begin() + 6, str.end(), "nobody!"); //"Hello nobody!"

//3)
str.replace(19, 5, alternate, 6, 6); //"Hello foo, bar and foobar!"
```

## C++ 14

```
//Define string
std::string str = "Hello foo, bar and world!";
std::string alternate = "Hello foobar";

//1)
str.replace(6, 3, "bar"); //"Hello bar, bar and world!"

//2)
str.replace(str.begin() + 6, str.end(), "nobody!"); //"Hello nobody!"

//3)
str.replace(19, 5, alternate, 6, 6); //"Hello foo, bar and foobar!"
```

```
//Define string
std::string str = "Hello foo, bar and world!";
std::string alternate = "Hello foobar";

//1)
str.replace(6, 3, "bar"); //"Hello bar, bar and world!"

//2)
str.replace(str.begin() + 6, str.end(), "nobody!"); //"Hello nobody!"

//3)
str.replace(19, 5, alternate, 6, 6); //"Hello foo, bar and foobar!"
```

## C++ 11

```
//Define string
std::string str = "Hello foo, bar and world!";
std::string alternate = "Hello foobar";

//1)
str.replace(6, 3, "bar"); //"Hello bar, bar and world!"

//2)
str.replace(str.begin() + 6, str.end(), "nobody!"); //"Hello nobody!"

//3)
str.replace(19, 5, alternate, 6, 6); //"Hello foo, bar and foobar!"
```

---

replacewithstr

```
//Define string
std::string str = "Hello foo, bar and world!";
std::string alternate = "Hello foobar";

//1)
```

```
str.replace(6, 3, "bar"); //Hello bar, bar and world!"  
  
//2)  
str.replace(str.begin() + 6, str.end(), "nobody!"); //Hello nobody!"  
  
//3)  
str.replace(19, 5, alternate, 6, 6); //Hello foo, bar and foobar!"
```

replacewithstr

```
//Define string  
std::string str = "Hello foo, bar and world!";  
std::string alternate = "Hello foobar";  
  
//1)  
str.replace(6, 3, "bar"); //Hello bar, bar and world!"  
  
//2)  
str.replace(str.begin() + 6, str.end(), "nobody!"); //Hello nobody!"  
  
//3)  
str.replace(19, 5, alternate, 6, 6); //Hello foo, bar and foobar!"
```

++=std::string +

```
std::string hello = "Hello";  
std::string world = "world";  
std::string helloworld = hello + world; // "Helloworld"
```

+=

```
std::string hello = "Hello";  
std::string world = "world";  
std::string helloworld = hello + world; // "Helloworld"
```

C

```
std::string hello = "Hello";  
std::string world = "world";  
std::string helloworld = hello + world; // "Helloworld"
```

push\_back() char

```
std::string hello = "Hello";  
std::string world = "world";  
std::string helloworld = hello + world; // "Helloworld"
```

append() +=

```
std::string hello = "Hello";  
std::string world = "world";  
std::string helloworld = hello + world; // "Helloworld"
```

```
std::string  
  
std::string str("Hello world!");
```

[]n

n.

```
std::string::operator[].
```

```
std::string str("Hello world!");
```

n

n.

```
std::string::at std::out_of_range
```

```
std::string str("Hello world!");
```

C++ 11

◦

```
std::string str("Hello world!");
```

```
std::string str("Hello world!");
```

## 1. str::strtok C++3

- std::strtok strings strings [strtok\\_s](#)
- std::strtok [Visual Studio](#)
- std::strtok std::string const string **S** const char\* **S** std::strtok std::string

```
std::strtok .
```

```
// String to tokenize  
std::string str{ "The quick brown fox" };  
// Vector to store tokens  
vector<std::string> tokens;  
  
for (auto i = strtok(&str[0], " "); i != NULL; i = strtok(NULL, " "))  
    tokens.push_back(i);
```

## 2. std::istream\_iterator< std::string> std::strtok const vector<string>

```

// String to tokenize
std::string str{ "The quick brown fox" };
// Vector to store tokens
vector<std::string> tokens;

for (auto i = strtok(&str[0], " "); i != NULL; i = strtok(NULL, " "))
    tokens.push_back(i);

```

3. std::regex\_token\_iterator<std::regex::

## C++ 11

```

// String to tokenize
std::string str{ "The quick brown fox" };
// Vector to store tokens
vector<std::string> tokens;

for (auto i = strtok(&str[0], " "); i != NULL; i = strtok(NULL, " "))
    tokens.push_back(i);

```

`regex_token_iterator` ◦

## constchar \*

const char\* std::string::c\_str() ◦ std::string::const◦

## C++ 17

data() char\* std::string◦

## C++ 11

char\* &s[0] ◦ C++ 11 null◦ &s[0] s &s.front() s◦

## C++ 11

```

std::string str("This is a string.");
const char* cstr = str.c_str(); // cstr points to: "This is a string.\0"
const char* data = str.data(); // data points to: "This is a string.\0"

```

```

std::string str("This is a string.");
const char* cstr = str.c_str(); // cstr points to: "This is a string.\0"
const char* data = str.data(); // data points to: "This is a string.\0"

```

`std::string::find` ◦ ◦ `std::string::npos`

```

std::string str = "Curiosity killed the cat";
auto it = str.find("cat");

if (it != std::string::npos)
    std::cout << "Found at position: " << it << '\n';
else
    std::cout << "Not found!\n";

```

```
std::string str = "Curiosity killed the cat";
auto it = str.find("cat");

if (it != std::string::npos)
    std::cout << "Found at position: " << it << '\n';
else
    std::cout << "Not found!\n";
```

◦

```
std::string str = "Curiosity killed the cat";
auto it = str.find("cat");

if (it != std::string::npos)
    std::cout << "Found at position: " << it << '\n';
else
    std::cout << "Not found!\n";
```

## 6

◦ 6'g' ◦

/

&lt;algorithm&gt; &lt;locale&gt;&lt;utility&gt; ◦

## C++ 11

◦ ◦ std::basic\_string std::string std::wstring std::vector std::list ◦

```
template <typename Sequence, // any basic_string, vector, list etc.
          typename Pred>      // a predicate on the element (character) type
Sequence& trim(Sequence& seq, Pred pred) {
    return trim_start(trim_end(seq, pred), pred);
}
```

```
template <typename Sequence, // any basic_string, vector, list etc.
          typename Pred>      // a predicate on the element (character) type
Sequence& trim(Sequence& seq, Pred pred) {
    return trim_start(trim_end(seq, pred), pred);
}
```

```
template <typename Sequence, // any basic_string, vector, list etc.
          typename Pred>      // a predicate on the element (character) type
Sequence& trim(Sequence& seq, Pred pred) {
    return trim_start(trim_end(seq, pred), pred);
}
```

std::string std::isspace()

```
template <typename Sequence, // any basic_string, vector, list etc.
          typename Pred> // a predicate on the element (character) type
Sequence& trim(Sequence& seq, Pred pred) {
    return trim_start(trim_end(seq, pred), pred);
}
```

```
std::wstring std::iswspace() {
```

```
template <typename Sequence, // any basic_string, vector, list etc.
          typename Pred> // a predicate on the element (character) type
Sequence& trim(Sequence& seq, Pred pred) {
    return trim_start(trim_end(seq, pred), pred);
}
```

```
== != < <= >>=std::string S
```

```
std::string str1 = "Foo";
std::string str2 = "Bar";

assert(!(str1 < str2));
assert(str1 > str2);
assert(!(str1 <= str2));
assert(str1 >= str2);
assert(!(str1 == str2));
assert(str1 != str2);
```

```
std::string::compare() {
```

- operator ==

```
str1.length() == str2.length() true false.
```

- operator !=

```
str1.length() != str2.length() true false.
```

- operator <operator >

```
.
```

- operator <=operator >=

```
.
```

◦ str1str2 nm str1[i]str2[i]  $i = 0, 1, 2 \dots maxnm$ .  $i \in \{m\}$ .

---

```
<
```

```
std::string str1 = "Foo";
std::string str2 = "Bar";

assert(!(str1 < str2));
```

```
assert(str > str2);
assert(!(str1 <= str2)));
assert(str1 >= str2);
assert(!(str1 == str2));
assert(str1 != str2);
```

1. 'B' == 'B' - o
2. 'a' == 'a' - o
3. 'r' == 'r' - o
4. str2str1。 str2 < str1。

## std :: wstring

C++ std::basic\_string。 std::string std::wstring

- std::stringchar
- std::wstringwchar\_t

wstring\_convert

```
#include <string>
#include <codecvt>
#include <locale>

std::string input_str = "this is a -string-, which is a sequence based on the -char- type.";
std::wstring input_wstr = L"this is a -wide- string, which is based on the -wchar_t- type.";

// conversion
std::wstring str_turned_to_wstr =
std::wstring_convert<std::codecvt_utf8<wchar_t>>().from_bytes(input_str);

std::string wstr_turned_to_str =
std::wstring_convert<std::codecvt_utf8<wchar_t>>().to_bytes(input_wstr);
```

/

```
#include <string>
#include <codecvt>
#include <locale>

std::string input_str = "this is a -string-, which is a sequence based on the -char- type.";
std::wstring input_wstr = L"this is a -wide- string, which is based on the -wchar_t- type.";

// conversion
std::wstring str_turned_to_wstr =
std::wstring_convert<std::codecvt_utf8<wchar_t>>().from_bytes(input_str);

std::string wstr_turned_to_str =
std::wstring_convert<std::codecvt_utf8<wchar_t>>().to_bytes(input_wstr);
```

```
#include <string>
#include <codecvt>
#include <locale>
```

```

std::string input_str = "this is a -string-, which is a sequence based on the -char- type.";
std::wstring input_wstr = L"this is a -wide- string, which is based on the -wchar_t- type.";

// conversion
std::wstring str_turned_to_wstr =
std::wstring_convert<std::codecvt_utf8<wchar_t>>().from_bytes(input_str);

std::string wstr_turned_to_str =
std::wstring_convert<std::codecvt_utf8<wchar_t>>().to_bytes(input_wstr);

std::wstring_convert<std::codecvt_utf8<wchar_t>>().from_bytes("Hello World!")。

```

char wchar\_t。 wchar\_t 2 Windows UTF-16 Windows 2000 UCS-2 UTF-32。 Linux。 char16\_t char32\_t C  
++ 11 UTF16 UTF32”。

## std :: string\_view

C ++ 17

C ++ 17 std::string\_view const char 。 。 C ++ 17

```

void foo(std::string const& s);           // pre-C++17, single argument, could incur
                                           // allocation if caller's data was not in a string
                                           // (e.g. string literal or vector<char> )

void foo(const char* s, size_t len); // pre-C++17, two arguments, have to pass them
                                     // both everywhere

void foo(const char* s);               // pre-C++17, single argument, but need to call
                                     // strlen()

template <class StringT>
void foo(StringT const& s);           // pre-C++17, caller can pass arbitrary char data
                                           // provider, but now foo() has to live in a header

```

```

void foo(std::string const& s);           // pre-C++17, single argument, could incur
                                           // allocation if caller's data was not in a string
                                           // (e.g. string literal or vector<char> )

void foo(const char* s, size_t len); // pre-C++17, two arguments, have to pass them
                                     // both everywhere

void foo(const char* s);               // pre-C++17, single argument, but need to call
                                     // strlen()

template <class StringT>
void foo(StringT const& s);           // pre-C++17, caller can pass arbitrary char data
                                           // provider, but now foo() has to live in a header

```

std::string\_view 。

string\_view 。

```

std::string

void foo(std::string const& s);           // pre-C++17, single argument, could incur
                                            // allocation if caller's data was not in a string
                                            // (e.g. string literal or vector<char> )

void foo(const char* s, size_t len); // pre-C++17, two arguments, have to pass them
                                            // both everywhere

void foo(const char* s);           // pre-C++17, single argument, but need to call
                                            // strlen()

template <class StringT>
void foo(StringT const& s);           // pre-C++17, caller can pass arbitrary char data
                                            // provider, but now foo() has to live in a header

```

## C++ 11

```
std::string
```

```

std::string str = "Hello World!";
for (auto c : str)
    std::cout << c;

```

“” for

```

std::string str = "Hello World!";
for (auto c : str)
    std::cout << c;

```

/

```
std::string°
```

"123abc"123 °

---

std::atof\*C

```

std::string ten = "10";

double num1 = std::atof(ten.c_str());
int num2 = std::atoi(ten.c_str());
long num3 = std::atol(ten.c_str());

```

## C++ 11

```

std::string ten = "10";

double num1 = std::atof(ten.c_str());
int num2 = std::atoi(ten.c_str());
long num3 = std::atol(ten.c_str());

```

0 ° 0“0”°

```
std::stoi(std::string s)
```

## C++ 11

```
std::string ten = "10";  
  
double num1 = std::atof(ten.c_str());  
int num2 = std::atoi(ten.c_str());  
long num3 = std::atol(ten.c_str());
```

```
std::atof("10") == 10.0
```

```
std::string ten = "10";  
  
double num1 = std::atof(ten.c_str());  
int num2 = std::atoi(ten.c_str());  
long num3 = std::atol(ten.c_str());
```

## C++ 11<codecvt><locale>

```
#include <iostream>  
#include <codecvt>  
#include <locale>  
#include <string>  
using namespace std;  
  
int main() {  
    // converts between wstring and utf8 string  
    wstring_convert<codecvt_utf8_utf16<wchar_t>> wchar_to_utf8;  
    // converts between u16string and utf8 string  
    wstring_convert<codecvt_utf8_utf16<char16_t>, char16_t> utf16_to_utf8;  
  
    wstring wstr = L"foobar";  
    string utf8str = wchar_to_utf8.to_bytes(wstr);  
    wstring wstr2 = wchar_to_utf8.from_bytes(utf8str);  
  
    wcout << wstr << endl;  
    cout << utf8str << endl;  
    wcout << wstr2 << endl;  
  
    u16string u16str = u"foobar";  
    string utf8str2 = utf16_to_utf8.to_bytes(u16str);  
    u16string u16str2 = utf16_to_utf8.from_bytes(utf8str2);  
  
    return 0;  
}
```

## Visual Studio 2015 char16\_twstring\_convert

```
#include <iostream>  
#include <codecvt>  
#include <locale>  
#include <string>  
using namespace std;  
  
int main() {
```

```

// converts between wstring and utf8 string
wstring_convert<codecvt_utf8_utf16<wchar_t>> wchar_to_utf8;
// converts between u16string and utf8 string
wstring_convert<codecvt_utf8_utf16<char16_t>, char16_t> utf16_to_utf8;

wstring wstr = L"foobar";
string utf8str = wchar_to_utf8.to_bytes(wstr);
wstring wstr2 = wchar_to_utf8.from_bytes(utf8str);

wcout << wstr << endl;
cout << utf8str << endl;
wcout << wstr2 << endl;

u16string u16str = u"foobar";
string utf8str2 = utf16_to_utf8.to_bytes(u16str);
u16string u16str2 = utf16_to_utf8.from_bytes(utf8str2);

return 0;
}

```

## C++ 14

### C++ 14 `std::mismatch`

```

std::string prefix = "foo";
std::string string = "foobar";

bool isPrefix = std::mismatch(prefix.begin(), prefix.end(),
    string.begin(), string.end()).first == prefix.end();

```

### C++ 14 `mismatch()`。

## C++ 14

```

std::mismatch()

std::string prefix = "foo";
std::string string = "foobar";

bool isPrefix = std::mismatch(prefix.begin(), prefix.end(),
    string.begin(), string.end()).first == prefix.end();

```

## C++ 17

```

std::string_view

std::string prefix = "foo";
std::string string = "foobar";

bool isPrefix = std::mismatch(prefix.begin(), prefix.end(),
    string.begin(), string.end()).first == prefix.end();

```

## std :: string

```
std::ostringstream<< std::ostringstreamstd::ostringstream std::string std::string
```

```
int
```

```
#include <sstream>

int main()
{
    int val = 4;
    std::ostringstream str;
    str << val;
    std::string converted = str.str();
    return 0;
}
```

```
#include <sstream>

int main()
{
    int val = 4;
    std::ostringstream str;
    str << val;
    std::string converted = str.str();
    return 0;
}
```

◦

```
#include <sstream>

int main()
{
    int val = 4;
    std::ostringstream str;
    str << val;
    std::string converted = str.str();
    return 0;
}
```

## C++ 11

C++ 11 `std::to_string` `std::to_wstring` ◦

```
#include <sstream>

int main()
{
    int val = 4;
    std::ostringstream str;
    str << val;
    std::string converted = str.str();
    return 0;
}
```

**std :: string** <https://riptutorial.com/zh-TW/cplusplus/topic/488/std---string>

# 104: std :: iomanip

## Examples

std ::

```
int val = 10;
// val will be printed to the extreme left end of the output console:
std::cout << val << std::endl;
// val will be printed in an output field of length 10 starting from right end of the field:
std::cout << std::setw(10) << val << std::endl;
```

```
int val = 10;
// val will be printed to the extreme left end of the output console:
std::cout << val << std::endl;
// val will be printed in an output field of length 10 starting from right end of the field:
std::cout << std::setw(10) << val << std::endl;
```

◦

◦ std::setw std :: iomanip.

std::setw

```
int val = 10;
// val will be printed to the extreme left end of the output console:
std::cout << val << std::endl;
// val will be printed in an output field of length 10 starting from right end of the field:
std::cout << std::setw(10) << val << std::endl;
```

n

## std :: setprecision

out << setprecision(n) in >> setprecision(n) outin n. ◦

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <limits>
int main()
{
    const long double pi = std::acos(-1.L);
    std::cout << "default precision (6): " << pi << '\n'
           << "std::precision(10):      " << std::setprecision(10) << pi << '\n'
           << "max precision:         "
           << std::setprecision(std::numeric_limits<long double>::digits10 + 1)
           << pi << '\n';
}
//Output
//default precision (6): 3.14159
```

```
//std::precision(10):      3.141592654
//max precision:          3.141592653589793239
```

## std :: setfill

```
out << setfill(c) c °

std::ostream::fill°

#include <iostream>
#include <iomanip>
int main()
{
    std::cout << "default fill: " << std::setw(10) << 42 << '\n'
        << "setfill('*'): " << std::setfill('*')
        << std::setw(10) << 42 << '\n';
}
//output::
//default fill:          42
//setfill('*'): *****42
```

## std :: setiosflags

```
out << setiosflags(mask) in >> setiosflags(mask) °

std::ios_base::fmtflags

• dec - I/O
• oct - I/O
• hex - I/O
• basefield - dec|oct|hex|0
• left -
• right -
• internal -
• adjustfield - left|right|internal °
• scientific -
• fixed -
• floatfield - scientific|fixed|(scientific|fixed)|0 °
• boolalpha - bool
• showbase - I/O
• showpoint -
• showpos - +
• skipws -
• unitbuf
• uppercase -
```

```
#include <iostream>
#include <string>
#include<iomanip>
int main()
```

```

{
    int l_iTemp = 47;
    std::cout<< std::resetiosflags(std::ios_base::basefield);
    std::cout<<std::setiosflags( std::ios_base::oct)<<l_iTemp<<std::endl;
    //output: 57
    std::cout<< std::resetiosflags(std::ios_base::basefield);
    std::cout<<std::setiosflags( std::ios_base::hex)<<l_iTemp<<std::endl;
    //output: 2f
    std::cout<<std::setiosflags( std::ios_base::uppercase)<<l_iTemp<<std::endl;
    //output 2F
    std::cout<<std::setfill('0')<<std::setw(12);
    std::cout<<std::resetiosflags(std::ios_base::uppercase);
    std::cout<<std::setiosflags( std::ios_base::right)<<l_iTemp<<std::endl;
    //output: 00000000002f

    std::cout<<std::resetiosflags(std::ios_base::basefield|std::ios_base::adjustfield);
    std::cout<<std::setfill('.')<<std::setw(10);
    std::cout<<std::setiosflags( std::ios_base::left)<<l_iTemp<<std::endl;
    //output: 47.......

    std::cout<<std::resetiosflags(std::ios_base::adjustfield)<<std::setfill('#');
    std::cout<<std::setiosflags(std::ios_base::internal|std::ios_base::showpos);
    std::cout<<std::setw(10)<<l_iTemp<<std::endl;
    //output +#####47

    double l_dTemp = -1.2;
    double pi = 3.14159265359;
    std::cout<<pi<<" "<<l_dTemp<<std::endl;
    //output +3.14159 -1.2
    std::cout<<std::setiosflags(std::ios_base::showpoint)<<l_dTemp<<std::endl;
    //output -1.20000
    std::cout<<setiosflags(std::ios_base::scientific)<<pi<<std::endl;
    //output: +3.141593e+00
    std::cout<<std::resetiosflags(std::ios_base::floatfield);
    std::cout<<setiosflags(std::ios_base::fixed)<<pi<<std::endl;
    //output: +3.141593
    bool b = true;
    std::cout<<std::setiosflags(std::ios_base::unitbuf|std::ios_base::boolalpha)<<b;
    //output: true
    return 0;
}

```

**std ::iomanip** <https://riptutorial.com/zh-TW/cplusplus/topic/6936/std---iomanip>

# 105: std ::

std::any

## Examples

```
std::any an_object{ std::string("hello world") };
if (an_object.has_value()) {
    std::cout << std::any_cast<std::string>(an_object) << '\n';
}

try {
    std::any_cast<int>(an_object);
} catch(std::bad_any_cast&) {
    std::cout << "Wrong type\n";
}

std::any_cast<std::string&>(an_object) = "42";
std::cout << std::any_cast<std::string>(an_object) << '\n';
```

```
std::any an_object{ std::string("hello world") };
if (an_object.has_value()) {
    std::cout << std::any_cast<std::string>(an_object) << '\n';
}

try {
    std::any_cast<int>(an_object);
} catch(std::bad_any_cast&) {
    std::cout << "Wrong type\n";
}

std::any_cast<std::string&>(an_object) = "42";
std::cout << std::any_cast<std::string>(an_object) << '\n';
```

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/7894/std--->

# 106: std ::Modifiers

std::forward\_list。 。 C。 std::list。

◦ `erase_after`。 std :: forward\_listContainerSize==AllocatorAwareContainerSequenceContainer。

## Examples

```
#include <forward_list>
#include <string>
#include <iostream>

template<typename T>
std::ostream& operator<<(std::ostream& s, const std::forward_list<T>& v) {
    s.put('[');
    char comma[3] = {'\0', ' ', '\0'};
    for (const auto& e : v) {
        s << comma << e;
        comma[0] = ',';
    }
    return s << ']';
}

int main()
{
    // c++11 initializer list syntax:
    std::forward_list<std::string> words1 {"the", "frogburt", "is", "also", "cursed"};
    std::cout << "words1: " << words1 << '\n';

    // words2 == words1
    std::forward_list<std::string> words2(words1.begin(), words1.end());
    std::cout << "words2: " << words2 << '\n';

    // words3 == words1
    std::forward_list<std::string> words3(words1);
    std::cout << "words3: " << words3 << '\n';

    // words4 is {"Mo", "Mo", "Mo", "Mo", "Mo"}
    std::forward_list<std::string> words4(5, "Mo");
    std::cout << "words4: " << words4 << '\n';
}
```

```
#include <forward_list>
#include <string>
#include <iostream>

template<typename T>
std::ostream& operator<<(std::ostream& s, const std::forward_list<T>& v) {
    s.put('[');
    char comma[3] = {'\0', ' ', '\0'};
    for (const auto& e : v) {
        s << comma << e;
        comma[0] = ',';
    }
    return s << ']';
}
```

```

}

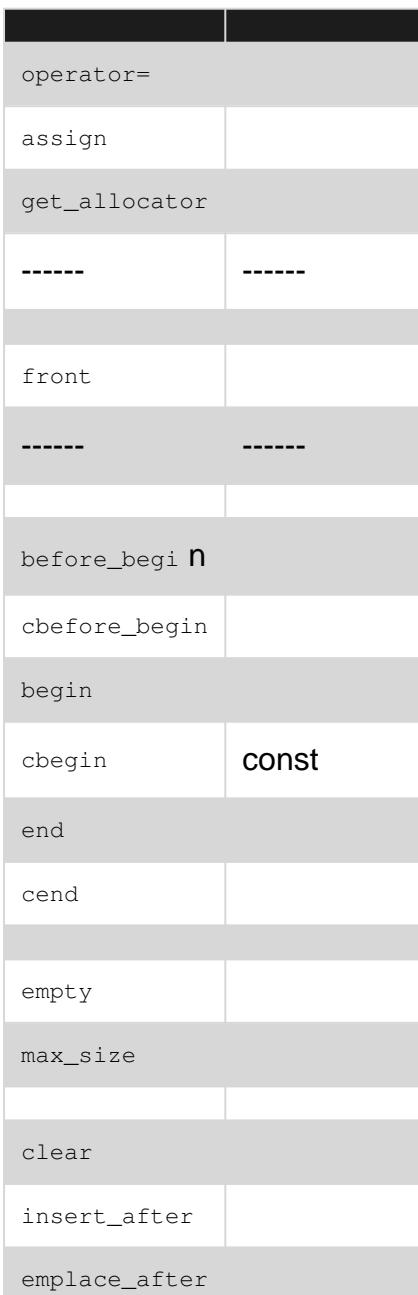
int main()
{
    // c++11 initializer list syntax:
    std::forward_list<std::string> words1 {"the", "frogurt", "is", "also", "cursed"};
    std::cout << "words1: " << words1 << '\n';

    // words2 == words1
    std::forward_list<std::string> words2(words1.begin(), words1.end());
    std::cout << "words2: " << words2 << '\n';

    // words3 == words1
    std::forward_list<std::string> words3(words1);
    std::cout << "words3: " << words3 << '\n';

    // words4 is {"Mo", "Mo", "Mo", "Mo", "Mo"}
    std::forward_list<std::string> words4(5, "Mo");
    std::cout << "words4: " << words4 << '\n';
}

```



erase_after	
push_front	
emplace_front	
pop_front	
resize	
swap	
merge	
splice_after	<b>forward_list</b>
remove	
remove_if	
reverse	
unique	
sort	

std ::Modifiers <https://riptutorial.com/zh-TW/cplusplus/topic/9703/std---modifiers>

# 107: std ::

## Examples

```
std::atomic<
```

```
std::memory_order<
```

**std :: atomic** TriviallyCopyable type T. std::atomicTriviallyCopyable type T. std::atomic<

**std :: atomic**

1. **bool** **typedef** std::atomic<T>

TypeDef	
std::atomic_bool	std::atomic<bool>

2 **typedef**

TypeDef	
std::atomic_char	std::atomic<char>
std::atomic_char	std::atomic<char>
std::atomic_schar	std::atomic<signed char>
std::atomic_uchar	std::atomic<unsigned char>
std::atomic_short	std::atomic<short>
std::atomic_ushort	std::atomic<unsigned short>
std::atomic_int	std::atomic<int>
std::atomic_uint	std::atomic<unsigned int>
std::atomic_long	std::atomic<long>
std::atomic_ulong	std::atomic<unsigned long>
std::atomic_llong	std::atomic<long long>
std::atomic_ullong	std::atomic<unsigned long long>
std::atomic_char16_t	std::atomic<char16_t>
std::atomic_char32_t	std::atomic<char32_t>
std::atomic_wchar_t	std::atomic<wchar_t>
std::atomic_int8_t	std::atomic<std::int8_t>

## Typedef

std::atomic_uint8_t	std::atomic<std::uint8_t>
std::atomic_int16_t	std::atomic<std::int16_t>
std::atomic_uint16_t	std::atomic<std::uint16_t>
std::atomic_int32_t	std::atomic<std::int32_t>
std::atomic_uint32_t	std::atomic<std::uint32_t>
std::atomic_int64_t	std::atomic<std::int64_t>
std::atomic_uint64_t	std::atomic<std::uint64_t>
std::atomic_int_least8_t	std::atomic<std::int_least8_t>
std::atomic_uint_least8_t	std::atomic<std::uint_least8_t>
std::atomic_int_least16_t	std::atomic<std::int_least16_t>
std::atomic_uint_least16_t	std::atomic<std::uint_least16_t>
std::atomic_int_least32_t	std::atomic<std::int_least32_t>
std::atomic_uint_least32_t	std::atomic<std::uint_least32_t>
std::atomic_int_least64_t	std::atomic<std::int_least64_t>
std::atomic_uint_least64_t	std::atomic<std::uint_least64_t>
std::atomic_int_fast8_t	std::atomic<std::int_fast8_t>
std::atomic_uint_fast8_t	std::atomic<std::uint_fast8_t>
std::atomic_int_fast16_t	std::atomic<std::int_fast16_t>
std::atomic_uint_fast16_t	std::atomic<std::uint_fast16_t>
std::atomic_int_fast32_t	std::atomic<std::int_fast32_t>
std::atomic_uint_fast32_t	std::atomic<std::uint_fast32_t>
std::atomic_int_fast64_t	std::atomic<std::int_fast64_t>
std::atomic_uint_fast64_t	std::atomic<std::uint_fast64_t>
std::atomic_intptr_t	std::atomic<std::intptr_t>
std::atomic_uintptr_t	std::atomic<std::uintptr_t>
std::atomic_size_t	std::atomic<std::size_t>
std::atomic_ptrdiff_t	std::atomic<std::ptrdiff_t>
std::atomic_intmax_t	std::atomic<std::intmax_t>
std::atomic_uintmax_t	std::atomic<std::uintmax_t>

std :: atomic\_int

```
#include <iostream>           // std::cout
#include <atomic>             // std::atomic, std::memory_order_relaxed
#include <thread>              // std::thread

std::atomic_int foo (0);

void set_foo(int x) {
    foo.store(x, std::memory_order_relaxed);      // set value atomically
}

void print_foo() {
    int x;
    do {
        x = foo.load(std::memory_order_relaxed);   // get value atomically
    } while (x==0);
    std::cout << "foo: " << x << '\n';
}

int main ()
{
    std::thread first (print_foo);
    std::thread second (set_foo,10);
    first.join();
    //second.join();
    return 0;
}
//output: foo: 10
```

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/7475/std--->

# 108: std ::

## Examples

OptionalsMaybe。 C ++ 17 std::optional。 std::optional<int>int。

Optionals。

std::optionalpair<bool, T>。

VS

nullptr。 - optional。

## vs Sentinel

◦ 0-1 nullptr。 00◦

VS std::pair<bool, T>

bool。

◦ optional<T>◦

C ++ 17 nullptr。 int。 std::optional。

struct Person。 。 Personpetstd::optional。

```
#include <iostream>
#include <optional>
#include <string>

struct Animal {
    std::string name;
};

struct Person {
    std::string name;
    std::optional<Animal> pet;
};

int main() {
    Person person;
    person.name = "John";

    if (person.pet) {
        std::cout << person.name << "'s pet's name is " <<
            person.pet->name << std::endl;
    }
    else {
```

```

        std::cout << person.name << " is alone." << std::endl;
    }
}

```

## C++ 17

- ◦
  - AppDelegate \*App::get\_delegate() nullptr ◦
  - ◦
- ◦
  - unsigned shortest\_path\_distance(Vertex a, Vertex b) unsigned  
shortest\_path\_distance(Vertex a, Vertex b)◦
- bool◦
  - std::pair<int, bool> parse(const std::string &str) **undefined** intboolfalse ◦

**JohnFluffyFurball**◦ Person::pet\_with\_name() **John**◦ **JohnWhiskers**std::nullopt◦

```

#include <iostream>
#include <optional>
#include <string>
#include <vector>

struct Animal {
    std::string name;
};

struct Person {
    std::string name;
    std::vector<Animal> pets;

    std::optional<Animal> pet_with_name(const std::string &name) {
        for (const Animal &pet : pets) {
            if (pet.name == name) {
                return pet;
            }
        }
        return std::nullopt;
    }
};

int main() {
    Person john;
    john.name = "John";

    Animal fluffy;
    fluffy.name = "Fluffy";
    john.pets.push_back(fluffy);

    Animal furball;
    furball.name = "Furball";
    john.pets.push_back(furball);

    std::optional<Animal> whiskers = john.pet_with_name("Whiskers");
    if (whiskers) {
        std::cout << "John has a pet named Whiskers." << std::endl;
    }
}

```

```
    else {
        std::cout << "Whiskers must not belong to John." << std::endl;
    }
}
```

```
std::optional<float> divide(float a, float b) {
    if (b!=0.f) return a/b;
    return {};
}
```

a/b。

```
std::optional<float> divide(float a, float b) {
    if (b!=0.f) return a/b;
    return {};
}
```

find( some\_range, 7 )some\_range7。 find\_if

◦

```
std::optional<float> divide(float a, float b) {
    if (b!=0.f) return a/b;
    return {};
}
```

```
std::optional<float> divide(float a, float b) {
    if (b!=0.f) return a/b;
    return {};
}
```

/◦

## value\_or

```
void print_name( std::ostream& os, std::optional<std::string> const& name ) {
    std::cout "Name is: " << name.value_or("<name missing>") << '\n';
}
```

value\_or optional◦

maybe-null◦ “”◦

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/2423/std--->

# 109: std ::

- std::map std::multimap <map>。
- std::map std::multimap。 std::multimap。
- std::map std::multimap std::map std::multimap。
- 。 search() insert() erase()  $\Theta \log n$ 。 std::unordered\_map。
- size() empty()  $\Theta 1$ 。

## Examples

```
std::map(key, value)。
```

```
std::map
```

```
std::map < std::string, int > ranking { std::make_pair("stackoverflow", 2),  
                                         std::make_pair("docs-beta", 1) };
```

```
std::map
```

```
std::map < std::string, int > ranking { std::make_pair("stackoverflow", 2),  
                                         std::make_pair("docs-beta", 1) };
```

overflow2.

```
std::map
```

```
std::map < std::string, int > ranking { std::make_pair("stackoverflow", 2),  
                                         std::make_pair("docs-beta", 1) };
```

```
operator[]。 const std::map。 find() at()。
```

## C++ 11

```
at() std::map
```

```
std::map < std::string, int > ranking { std::make_pair("stackoverflow", 2),  
                                         std::make_pair("docs-beta", 1) };
```

```
at() std::out_of_range。
```

```
std::map std::multimap
```

## C++ 11

```
std::map < std::string, int > ranking { std::make_pair("stackoverflow", 2),
                                         std::make_pair("docs-beta", 1) };
```

## std :: mapstd :: multimap

```
std::mapstd::multimapo {key, value}std::make_pair(key, value)o std::mapo
```

```
std::multimap < int, std::string > mmp { std::make_pair(2, "stackoverflow"),
                                         std::make_pair(1, "docs-beta"),
                                         std::make_pair(2, "stackexchange") };
```

// 1 docs-beta  
// 2 stackoverflow  
// 2 stackexchange

```
std::map < int, std::string > mp { std::make_pair(2, "stackoverflow"),
                                         std::make_pair(1, "docs-beta"),
                                         std::make_pair(2, "stackexchange") };
```

// 1 docs-beta  
// 2 stackoverflow

◦

```
std::multimap < int, std::string > mmp { std::make_pair(2, "stackoverflow"),
                                         std::make_pair(1, "docs-beta"),
                                         std::make_pair(2, "stackexchange") };
```

// 1 docs-beta  
// 2 stackoverflow  
// 2 stackexchange

```
std::map < int, std::string > mp { std::make_pair(2, "stackoverflow"),
                                         std::make_pair(1, "docs-beta"),
                                         std::make_pair(2, "stackexchange") };
```

// 1 docs-beta  
// 2 stackoverflow

```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
```

mmp.clear(); //empty multimap

```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
```

mmp.clear(); //empty multimap

```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
```

mmp.clear(); //empty multimap

pred

```
std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
```

mmp.clear(); //empty multimap

std::map map◦

```
std::map< std::string, size_t > fruits_count;
```

- insert() std::map◦ pair

```
std::map< std::string, size_t > fruits_count;
```

insert() boolpair

- booltrue ◦
- key◦ boolfalse ◦

```
std::map< std::string, size_t > fruits_count;
```

- std::map

```
std::map< std::string, size_t > fruits_count;
```

◦ std::map::operator[]◦

- insert()◦ insertvoid

```
std::map< std::string, size_t > fruits_count;
```

- insert() value\_type

```
std::map< std::string, size_t > fruits_count;
```

Olog n std::map◦

C ++ 11

make\_pair() emplace() pair

```
std::map< std::string, size_t > fruits_count;
```

emplace\_hint() hint◦ hint◦ emplace()

```
std::map< std::string, size_t > fruits_count;
```

## std :: map std :: multimap

std::map std::multimap

```

std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };

//Range based loop - since C++11
for(const auto &x: mmp)
    std::cout<< x.first << ":"<< x.second << std::endl;

//Forward iterator for loop: it would loop through first element to last element
//it will be a std::map< int, int >::iterator
for (auto it = mmp.begin(); it != mmp.end(); ++it)
    std::cout<< it->first << ":"<< it->second << std::endl; //Do something with iterator

//Backward iterator for loop: it would loop through last element to first element
//it will be a std::map< int, int >::reverse_iterator
for (auto it = mmp.rbegin(); it != mmp.rend(); ++it)
    std::cout<< it->first << ":"<< it->second << std::endl; //Do something with iterator

```

std::mapstd::multimapauto**SO**。

## std :: mapstd :: multimap

std::mapstd::multimap

- `find()`。 `end()`。

```

std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
auto it = mmp.find(6);
if(it!=mmp.end())
    std::cout << it->first << ", " << it->second << std::endl; //prints: 6, 5
else
    std::cout << "Value does not exist!" << std::endl;

it = mmp.find(66);
if(it!=mmp.end())
    std::cout << it->first << ", " << it->second << std::endl;
else
    std::cout << "Value does not exist!" << std::endl; // This line would be executed.

```

- `std::mapstd::multimapcount()`。 `std::mapcount()`**01**。 `std::multimap count()`**1**。

```

std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
auto it = mmp.find(6);
if(it!=mmp.end())
    std::cout << it->first << ", " << it->second << std::endl; //prints: 6, 5
else
    std::cout << "Value does not exist!" << std::endl;

it = mmp.find(66);
if(it!=mmp.end())
    std::cout << it->first << ", " << it->second << std::endl;
else
    std::cout << "Value does not exist!" << std::endl; // This line would be executed.

```

findmultimaps。

- `std::multimap:: equal_range()``std::pair:: end()`。

```

std::multimap< int , int > mmp{ {1, 2}, {3, 4}, {6, 5}, {8, 9}, {3, 4}, {6, 7} };
auto it = mmp.find(6);
if(it!=mmp.end())
    std::cout << it->first << ", " << it->second << std::endl; //prints: 6, 5
else
    std::cout << "Value does not exist!" << std::endl;

it = mmp.find(66);
if(it!=mmp.end())
    std::cout << it->first << ", " << it->second << std::endl;
else
    std::cout << "Value does not exist!" << std::endl; // This line would be executed.

```

std::mapempty() truefalse map. size() std::map

```

std::map<std::string , int> rank {{"facebook.com", 1} , {"google.com", 2}, {"youtube.com", 3}};
if(!rank.empty()){
    std::cout << "Number of elements in the rank map: " << rank.size() << std::endl;
}
else{
    std::cout << "The rank map is empty" << std::endl;
}

```

◦

```

#include <string>
#include <map>
std::map<std::string, size_t> fruits_count;

```

std::string size\_t ◦

◦ ◦

- multimap

- ```
#include <string>
#include <map>
std::map<std::string, size_t> fruits_count;
```

StrLessfalse ◦

**Multimap** ◦

```

#include <string>
#include <map>
std::map<std::string, size_t> fruits_count;

```

◦ ◦ ◦

```

#include <string>
#include <map>
std::map<std::string, size_t> fruits_count;

```

- unordered\_map◦

## std :: map

copyable assignable ◦ ◦ std::less<KeyType> <◦

```
struct CmpMyType
{
    bool operator() ( MyType const& lhs, MyType const& rhs ) const
    {
        // ...
    }
};
```

C ++“compare” ◦ compare(x, x) X false ◦ CmpMyType()(a, b) true CmpMyType()(b, a) false false ◦

◦

fx yfyxx y◦ ◦

C ++<◦

```
struct CmpMyType
{
    bool operator() ( MyType const& lhs, MyType const& rhs ) const
    {
        // ...
    }
};
```

/◦

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/681/std--->

# 110: std ::

## Examples

Pair `std::make_pair`

`firstsecond` .

```
#include <iostream>
#include <utility>

int main()
{
    std::pair<int,int> p = std::make_pair(1,2); //Creating the pair
    std::cout << p.first << " " << p.second << std::endl; //Accessing the elements

    //We can also create a pair and assign the elements later
    std::pair<int,int> p1;
    p1.first = 3;
    p1.second = 4;
    std::cout << p1.first << " " << p1.second << std::endl;

    //We can also create a pair using a constructor
    std::pair<int,int> p2 = std::pair<int,int>(5, 6);
    std::cout << p2.first << " " << p2.second << std::endl;

    return 0;
}
```

`lhsrhs`

- `operator==lhsrhs` `lhs.first == rhs.first` `lhs.second == rhs.second` true false

```
std::pair<int, int> p1 = std::make_pair(1, 2);
std::pair<int, int> p2 = std::make_pair(2, 2);

if (p1 == p2)
    std::cout << "equals";
else
    std::cout << "not equal"//statement will show this, because they are not identical
```

- `operator!=lhsrhs` `true` `lhs.first != rhs.first` OR `lhs.second != rhs.second` false
- `operator<lhs.first<rhs.first` true . `rhs.first<lhs.first` false . `lhs.second<rhs.second` true  
false .
- `operator<= return !(rhs<lhs)`
- `operator>rhs<lhs`

- operator>= **returns** !(lhs<rhs)

- operator<◦

```
std::pair<int, int> p1 = std::make_pair(1, 2);
std::pair<int, int> p2 = std::make_pair(2, 2);

if (p1 == p2)
    std::cout << "equals";
else
    std::cout << "not equal"//statement will show this, because they are not identical
```

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/4834/std--->

# 111: std ::

```
◦ ◦  
; C-arraysstd::array  
  
std::vector#include <vector><vector>  
  
std::vector◦ std::vector<std::vector<int> >◦
```

## Examples

### std :: vector

```
std::vector
```

#### C ++ 11

```
std::vector<int> v{ 1, 2, 3 }; // v becomes {1, 2, 3}  
  
// Different from std::vector<int> v(3, 6)  
std::vector<int> v{ 3, 6 }; // v becomes {3, 6}
```

```
std::vector<int> v{ 1, 2, 3 }; // v becomes {1, 2, 3}  
  
// Different from std::vector<int> v(3, 6)  
std::vector<int> v{ 3, 6 }; // v becomes {3, 6}
```

v2

```
std::vector<int> v{ 1, 2, 3 }; // v becomes {1, 2, 3}  
  
// Different from std::vector<int> v(3, 6)  
std::vector<int> v{ 3, 6 }; // v becomes {3, 6}
```

#### C ++ 11

v2

```
std::vector<int> v{ 1, 2, 3 }; // v becomes {1, 2, 3}  
  
// Different from std::vector<int> v(3, 6)  
std::vector<int> v{ 3, 6 }; // v becomes {3, 6}
```

v

```
std::vector<int> v{ 1, 2, 3 }; // v becomes {1, 2, 3}  
  
// Different from std::vector<int> v(3, 6)  
std::vector<int> v{ 3, 6 }; // v becomes {3, 6}
```

## C++ 11

### Iterator move-construction `std::make_move_iterator v`

```
std::vector<int> v{ 1, 2, 3 }; // v becomes {1, 2, 3}

// Different from std::vector<int> v(3, 6)
std::vector<int> v{ 3, 6 }; // v becomes {3, 6}
```

### `assign()` `std::vector`

```
std::vector<int> v{ 1, 2, 3 }; // v becomes {1, 2, 3}

// Different from std::vector<int> v(3, 6)
std::vector<int> v{ 3, 6 }; // v becomes {3, 6}
```

/

```
struct Point {
    double x, y;
    Point(double x, double y) : x(x), y(y) {}
};

std::vector<Point> v;
Point p(10.0, 2.0);
v.push_back(p); // p is copied into the vector.
```

## C++ 11

```
struct Point {
    double x, y;
    Point(double x, double y) : x(x), y(y) {}
};

std::vector<Point> v;
Point p(10.0, 2.0);
v.push_back(p); // p is copied into the vector.
```

`std::vector push_front()` `std::list``std::deque` `push_back()`

---

```
struct Point {
    double x, y;
    Point(double x, double y) : x(x), y(y) {}
};

std::vector<Point> v;
Point p(10.0, 2.0);
v.push_back(p); // p is copied into the vector.
```

## C++ 11

```
struct Point {
    double x, y;
    Point(double x, double y) : x(x), y(y) {}
};

std::vector<Point> v;
Point p(10.0, 2.0);
```

```
v.push_back(p); // p is copied into the vector.
```

```
struct Point {  
    double x, y;  
    Point(double x, double y) : x(x), y(y) {}  
};  
std::vector<Point> v;  
Point p(10.0, 2.0);  
v.push_back(p); // p is copied into the vector.
```

```
struct Point {  
    double x, y;  
    Point(double x, double y) : x(x), y(y) {}  
};  
std::vector<Point> v;  
Point p(10.0, 2.0);  
v.push_back(p); // p is copied into the vector.
```

`reserve()`

```
struct Point {  
    double x, y;  
    Point(double x, double y) : x(x), y(y) {}  
};  
std::vector<Point> v;  
Point p(10.0, 2.0);  
v.push_back(p); // p is copied into the vector.
```

`resize()` 200°

## std :: vector

`std::vector` ° v

```
std::vector<int> v;
```

## C ++ 11

```
std::vector<int> v;
```

## C ++ 11

```
std::vector<int> v;
```

```
std::vector<int> v;
```

## C ++ 14

```
std::vector<int> v;
```

```
std::vector<int> v;
```

; ◦ begin() end() ◦

## C++ 14

```
std::vector<int> v;
```

# const

C++ 11 `cbegin()` `cend()` `const` ◦ `const`

## C++ 11

```
std::vector<int> v;
```

## C++ 17

### as\_const

```
std::vector<int> v;
```

## C++

## C++ 14

```
std::vector<int> v;
```

std::vector C++ ◦ std::vector ◦ CC++ std::vector ◦

std::vector

•  
•

operator [] at () ◦

std::vector<vector<bool>> const ◦

[] at () [] at () ◦ index < 0 index >= size[] at () std::out\_of\_range ◦

C++ 11 C++ 11 ◦

## C++ 11

```
std::vector<int> v{ 1, 2, 3 };

std::vector<int> v{ 1, 2, 3 };

at() [] . . .

std::vector<int> v{ 1, 2, 3 };

operator[] iCPU.

front() back() [ ]

std::vector<int> v{ 1, 2, 3 };

front() back() . front() back() empty() . 'empty'

std::vector<int> v{ 1, 2, 3 };

110. 'empty' .
```

## C++ 11

data() std::vector C.

```
std::vector<int> v{ 1, 2, 3 };
```

## C++ 11

C++ 11 front() data()

```
std::vector<int> v{ 1, 2, 3 };
```

operator& . pre-C++ 11 std::addressof .

“std::vector” Iterators.

## C++ 11

```
std::vector<int> v{ 1, 2, 3 };
```

std::vector<T> T\* S . .

. vector / vector vector .

## C++ 11

```
std::vector<int> v{ 1, 2, 3 };
```

## **std :: vectorC**

```
std::vector<CC>。
```

C ++ 11

```
std::vector<int> v{ 1, 2, 3 };
int* p = v.data();
```

## C ++.data ()。

C ++ 11

```
std::vector<int> v{ 1, 2, 3 };
int* p = v.data();
```

```
v[0]v.front()。
```

push back resize .

```
std::vector<int> v{ 1, 2, 3 };
int* p = v.data();
```

1

```
std::vector< />
```

1

- vector capacity/

```
vector<int> v(5); // Vector has a size of 5; capacity is unknown.  
int *p1 = &v[0];  
v.push_back(2); // p1 may have been invalidated, since the capacity was unknown.  
  
v.reserve(20); // Capacity is now at least 20.  
int *p2 = &v[0];  
v.push_back(4); // p2 is *not* invalidated, since the size of `v` is now 7.  
v.insert(v.end(), 30, 9); // Inserts 30 elements at the end. The size exceeds the  
// requested capacity of 20, so `p2` is (probably) invalidated.  
int *p3 = &v[0];  
v.reserve(v.capacity() + 20); // Capacity exceeded, thus `p3` is invalid.
```

C++ 11

```
vector<int> v(5); // Vector has a size of 5; capacity is unknown.  
int *p1 = &v[0];  
v.push_back(2); // p1 may have been invalidated, since the capacity was unknown.
```

```

v.reserve(20);      // Capacity is now at least 20.
int *p2 = &v[0];
v.push_back(4);    // p2 is *not* invalidated, since the size of `v` is now 7.
v.insert(v.end(), 30, 9); // Inserts 30 elements at the end. The size exceeds the
                         // requested capacity of 20, so `p2` is (probably) invalidated.
int *p3 = &v[0];
v.reserve(v.capacity() + 20); // Capacity exceeded, thus `p3` is invalid.

```

- /° end

```

vector<int> v(5); // Vector has a size of 5; capacity is unknown.
int *p1 = &v[0];
v.push_back(2);   // p1 may have been invalidated, since the capacity was unknown.

v.reserve(20);    // Capacity is now at least 20.
int *p2 = &v[0];
v.push_back(4);    // p2 is *not* invalidated, since the size of `v` is now 7.
v.insert(v.end(), 30, 9); // Inserts 30 elements at the end. The size exceeds the
                         // requested capacity of 20, so `p2` is (probably) invalidated.
int *p3 = &v[0];
v.reserve(v.capacity() + 20); // Capacity exceeded, thus `p3` is invalid.

```

- /° end

```

vector<int> v(5); // Vector has a size of 5; capacity is unknown.
int *p1 = &v[0];
v.push_back(2);   // p1 may have been invalidated, since the capacity was unknown.

v.reserve(20);    // Capacity is now at least 20.
int *p2 = &v[0];
v.push_back(4);    // p2 is *not* invalidated, since the size of `v` is now 7.
v.insert(v.end(), 30, 9); // Inserts 30 elements at the end. The size exceeds the
                         // requested capacity of 20, so `p2` is (probably) invalidated.
int *p3 = &v[0];
v.reserve(v.capacity() + 20); // Capacity exceeded, thus `p3` is invalid.

```

- operator= **copymove** **clear()** **vector**/°

---

```

std::vector<int> v{ 1, 2, 3 };
v.pop_back();                                // v becomes {1, 2}

```

---

```

std::vector<int> v{ 1, 2, 3 };
v.pop_back();                                // v becomes {1, 2}

```

---

```

std::vector<int> v{ 1, 2, 3 };
v.pop_back();                                // v becomes {1, 2}

```

---

vector **std :: list** °

---

```
std::vector<int> v{ 1, 2, 3 };
v.pop_back(); // v becomes {1, 2}
```

◦ ◦

`erase - o std::remove` `erase o std :: list`

```
std::vector<int> v{ 1, 2, 3 };
v.pop_back(); // v becomes {1, 2}
```

## lambda

C++ 11

```
std::vector<int> v{ 1, 2, 3 };
v.pop_back(); // v becomes {1, 2}
```

```
std::vector<int> v{ 1, 2, 3 };
v.pop_back(); // v becomes {1, 2}
```

`it o remove_if o`

```
std::vector<int> v{ 1, 2, 3 };
v.pop_back(); // v becomes {1, 2}
```

- `itbase o`
- `vector::erase(iterator) o`
- `reverse_iterator::reverse_iterator(iterator) o`

`it = rev_itr(v.erase(it.base())) it v; it o`

`v.clear() capacity () o`

```
std::vector<int> v{ 1, 2, 3 };
v.pop_back(); // v becomes {1, 2}
```

## C++ 11

`shrink_to_fit()`

```
std::vector<int> v{ 1, 2, 3 };
v.pop_back(); // v becomes {1, 2}
```

`shrink_to_fit。`

## `std :: vector`

```
<algorithm>std::findstd::vector<
std::findoperator==。 ◦
std::findstd::vector::end conststd::vector::cend ◦
```

## C++ 11

```
static const int arr[] = {5, 4, 3, 2, 1};
std::vector<int> v (arr, arr + sizeof(arr) / sizeof(arr[0]) );

std::vector<int>::iterator it = std::find(v.begin(), v.end(), 4);
std::vector<int>::difference_type index = std::distance(v.begin(), it);
// `it` points to the second element of the vector, `index` is 1

std::vector<int>::iterator missing = std::find(v.begin(), v.end(), 10);
std::vector<int>::difference_type index_missing = std::distance(v.begin(), missing);
// `missing` is v.end(), `index_missing` is 5 (ie. size of the vector)
```

## C++ 11

```
static const int arr[] = {5, 4, 3, 2, 1};
std::vector<int> v (arr, arr + sizeof(arr) / sizeof(arr[0]) );

std::vector<int>::iterator it = std::find(v.begin(), v.end(), 4);
std::vector<int>::difference_type index = std::distance(v.begin(), it);
// `it` points to the second element of the vector, `index` is 1

std::vector<int>::iterator missing = std::find(v.begin(), v.end(), 10);
std::vector<int>::difference_type index_missing = std::distance(v.begin(), missing);
// `missing` is v.end(), `index_missing` is 5 (ie. size of the vector)
```

`binary_search◦`

---

`std::find_if◦ std::find std::find_if◦ bool`

## C++ 11

```
static const int arr[] = {5, 4, 3, 2, 1};
std::vector<int> v (arr, arr + sizeof(arr) / sizeof(arr[0]) );

std::vector<int>::iterator it = std::find(v.begin(), v.end(), 4);
std::vector<int>::difference_type index = std::distance(v.begin(), it);
```

```
// `it` points to the second element of the vector, `index` is 1

std::vector<int>::iterator missing = std::find(v.begin(), v.end(), 10);
std::vector<int>::difference_type index_missing = std::distance(v.begin(), missing);
// `missing` is v.end(), `index_missing` is 5 (ie. size of the vector)
```

## C++ 11

```
static const int arr[] = {5, 4, 3, 2, 1};
std::vector<int> v (arr, arr + sizeof(arr) / sizeof(arr[0]) );

std::vector<int>::iterator it = std::find(v.begin(), v.end(), 4);
std::vector<int>::difference_type index = std::distance(v.begin(), it);
// `it` points to the second element of the vector, `index` is 1

std::vector<int>::iterator missing = std::find(v.begin(), v.end(), 10);
std::vector<int>::difference_type index_missing = std::distance(v.begin(), missing);
// `missing` is v.end(), `index_missing` is 5 (ie. size of the vector)
```

## std :: vector

`std::begin``std::end``std::vector`

## C++ 11

```
int values[5] = { 1, 2, 3, 4, 5 }; // source array

std::vector<int> v(std::begin(values), std::end(values)); // copy array to new vector

for(auto &x: v)
    std::cout << x << " ";
std::cout << std::endl;
```

1 2 3 4 5

```
int values[5] = { 1, 2, 3, 4, 5 }; // source array

std::vector<int> v(std::begin(values), std::end(values)); // copy array to new vector

for(auto &x: v)
    std::cout << x << " ";
std::cout << std::endl;
```

## C++ 11 initializer\_list <>

```
int values[5] = { 1, 2, 3, 4, 5 }; // source array

std::vector<int> v(std::begin(values), std::end(values)); // copy array to new vector

for(auto &x: v)
    std::cout << x << " ";
std::cout << std::endl;
```

## 23.3.7 `vector<bool>::bool_` C++`vector<vector<bool>`

- `vector<bool>::operator[]` C API。
- `at()` operator `[>] bool = bool::operator<> std::vector<bool>`

## C++ 11

```
std::vector<bool> v = {true, false};
for (auto &b: v) { } // error
```

`bool& operator [>] vector<bool>`

```
std::vector<bool> v = {true, false};
for (auto &b: v) { } // error
```

`std::vector<bool>::at() 18 8 11.`

`vector<bool>::at() [XXXXXXXXX]`

`std::vector<char> 8`

## C++ 11

```
std::vector<bool> v = {true, false};
for (auto &b: v) { } // error
```

```
std::vector<bool> v = {true, false};
for (auto &b: v) { } // error
```

`std::vector<bool> 8`

## C++ 11

```
std::vector<bool> v = {true, false};
for (auto &b: v) { } // error
```

```
std::vector<bool> v = {true, false};
for (auto &b: v) { } // error
```

`std::vector<bool> 88 std::vector<bool> 1. . vector<bool> CAPI API.`

### 1. `size()` Convenience `empty() true`

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

### 2. 0

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
```

```
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

3. NN push\_back() insert() resize()。
4. Npop\_back() N pop\_back() erase() clear()。

## 5. VectorRAM

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

int

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

◦ /◦ ◦

1. capacity()。

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

2. reserve( N )N

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

3. shrink\_to\_fit()。

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

Vector 21.5 - ◦ ◦

```
vector<int> v = { 1, 2, 3 }; // size is 3
const vector<int>::size_type size = v.size();
cout << size << endl; // prints 3
cout << boolalpha << v.empty() << endl; // prints false
```

```
std::vector insert()

std::vector<int> a = {0, 1, 2, 3, 4};
std::vector<int> b = {5, 6, 7, 8, 9};

a.insert(a.end(), b.begin(), b.end());
```

insert()。

## C++11

```
std::begin() std::end()

std::vector<int> a = {0, 1, 2, 3, 4};
std::vector<int> b = {5, 6, 7, 8, 9};

a.insert(a.end(), b.begin(), b.end());
```

b o o

```
std::vector<int> a = {0, 1, 2, 3, 4};
std::vector<int> b = {5, 6, 7, 8, 9};

a.insert(a.end(), b.begin(), b.end());
```

std::vector o

```
// Initialize a vector with 100 elements
std::vector<int> v(100);

// The vector's capacity is always at least as large as its size
auto const old_capacity = v.capacity();
// old_capacity >= 100

// Remove half of the elements
v.erase(v.begin() + 50, v.end()); // Reduces the size from 100 to 50 (v.size() == 50),
                                // but not the capacity (v.capacity() == old_capacity)
```

o o o

```
// Initialize a vector with 100 elements
std::vector<int> v(100);

// The vector's capacity is always at least as large as its size
auto const old_capacity = v.capacity();
// old_capacity >= 100

// Remove half of the elements
v.erase(v.begin() + 50, v.end()); // Reduces the size from 100 to 50 (v.size() == 50),
                                // but not the capacity (v.capacity() == old_capacity)
```

## C++11

### C++11 shrink\_to\_fit()

```
// Initialize a vector with 100 elements
std::vector<int> v(100);

// The vector's capacity is always at least as large as its size
auto const old_capacity = v.capacity();
// old_capacity >= 100

// Remove half of the elements
v.erase(v.begin() + 50, v.end()); // Reduces the size from 100 to 50 (v.size() == 50),
// but not the capacity (v.capacity() == old_capacity)
```

shrink\_to\_fit()。

<algorithm>

< .

std::sort()

```
std::vector<int> v;
// add some code here to fill v with some elements
std::sort(v.begin(), v.end());
```

std::lower\_bound()。 std::find() std::find()。

```
std::vector<int> v;
// add some code here to fill v with some elements
std::sort(v.begin(), v.end());
```

std::lower\_bound()。

```
std::vector<int> v;
// add some code here to fill v with some elements
std::sort(v.begin(), v.end());
```

push\_back() std::sort()。 。

std::lower\_bound()。 std::upper\_bound()

```
std::vector<int> v;
// add some code here to fill v with some elements
std::sort(v.begin(), v.end());
```

std::equal\_range()

```
std::vector<int> v;
// add some code here to fill v with some elements
std::sort(v.begin(), v.end());
```

std::binary\_search()

```
std::vector<int> v;
```

```
// add some code here to fill v with some elements
std::sort(v.begin(), v.end());
```

## C++ 11

### C++ 11。。

```
#include <vector>
#include <iostream>

// If the compiler is unable to perform named return value optimization (NRVO)
// and elide the move altogether, it is required to move from v into the return value.
std::vector<int> fillVector(int a, int b) {
    std::vector<int> v;
    v.reserve(b-a+1);
    for (int i = a; i <= b; i++) {
        v.push_back(i);
    }
    return v; // implicit move
}

int main() { // declare and fill vector
    std::vector<int> vec = fillVector(1, 10);

    // print vector
    for (auto value : vec)
        std::cout << value << " "; // this will print "1 2 3 4 5 6 7 8 9 10 "

    std::cout << std::endl;

    return 0;
}
```

## C++ 11

### C++ 11copy elision。.

```
#include <vector>
#include <iostream>

// If the compiler is unable to perform named return value optimization (NRVO)
// and elide the move altogether, it is required to move from v into the return value.
std::vector<int> fillVector(int a, int b) {
    std::vector<int> v;
    v.reserve(b-a+1);
    for (int i = a; i <= b; i++) {
        v.push_back(i);
    }
    return v; // implicit move
}

int main() { // declare and fill vector
    std::vector<int> vec = fillVector(1, 10);

    // print vector
    for (auto value : vec)
        std::cout << value << " "; // this will print "1 2 3 4 5 6 7 8 9 10 "
```

```
    std::cout << std::endl;

    return 0;
}
```

```
std::max_element std::min_element 。 <algorithm> 。 。 v.end() 。
```

```
std::vector<int> v = {5, 2, 8, 10, 9};  
int maxElementIndex = std::max_element(v.begin(), v.end()) - v.begin();  
int maxElement = *std::max_element(v.begin(), v.end());  
  
int minElementIndex = std::min_element(v.begin(), v.end()) - v.begin();  
int minElement = *std::min_element(v.begin(), v.end());  
  
std::cout << "maxElementIndex:" << maxElementIndex << ", maxElement:" << maxElement << '\n';  
std::cout << "minElementIndex:" << minElementIndex << ", minElement:" << minElement << '\n';
```

maxElementIndex3maxElement10  
minElementIndex1minElement2

C++ 11

## std::minmax\_element<algorithm>

```
std::vector<int> v = {5, 2, 8, 10, 9};
int maxElementIndex = std::max_element(v.begin(), v.end()) - v.begin();
int maxElement = *std::max_element(v.begin(), v.end());

int minElementIndex = std::min_element(v.begin(), v.end()) - v.begin();
int minElement = *std::min_element(v.begin(), v.end());

std::cout << "maxElementIndex:" << maxElementIndex << ", maxElement:" << maxElement << '\n';
std::cout << "minElementIndex:" << minElementIndex << ", minElement:" << minElement << '\n';
```

2  
10

2D.

340

```
std::vector<std::vector<int> > matrix(3, std::vector<int>(4));
```

C ++ 11

7

```
std::vector<std::vector<int>> matrix(3, std::vector<int>(4));
```

2D

```
std::vector<std::vector<int> > matrix(3, std::vector<int>(4));
```

◦

```
std::vector<std::vector<int> > matrix(3, std::vector<int>(4));
```

## C++ 11

```
std::vector<std::vector<int> > matrix(3, std::vector<int>(4));
```

◦

◦ ◦

**std ::** <https://riptutorial.com/zh-TW/cplusplus/topic/511/std--->

# 112: std ::

```
Variant union. .
.
std::tuple std::optional .
.
std::getstd::get_if std::visit . if constexpr visitvisitvisit.
```

## Examples

## **std :: variant**

```
intstring.o

std::variant< int, std::string > var;

std::variant< int, std::string > var;

std::visit

std::variant< int, std::string > var;
```

lambda.

```
std::variant< int, std::string > var;  
  
◦ get_if  
  
std::variant< int, std::string > var;  
  
nullptr ◦  
  
◦ ◦  
  
◦ union ◦
```

```
template<class F>
struct pseudo_method {
    F f;
    // enable C++17 class type deduction:
```

```

pseudo_method( F&& fin ):f(std::move(fin)) {}

// Koenig lookup operator->*, as this is a pseudo-method it is appropriate:
template<class Variant> // maybe add SFINAE test that LHS is actually a variant.
friend decltype(auto) operator->*( Variant&& var, pseudo_method const& method ) {
    // var->*method returns a lambda that perfect forwards a function call,
    // behaving like a method pointer basically:
    return [&](auto&&...args)->decltype(auto) {
        // use visit to get the type of the variant:
        return std::visit(
            [&](auto&& self)->decltype(auto) {
                // decltype(x)(x) is perfect forwarding in a lambda:
                return method.f( decltype(self)(self), decltype(args)(args)... );
            },
            std::forward<Var>(var)
        );
    };
}
};

operator->*Variant .

```

```

template<class F>
struct pseudo_method {
    F f;
    // enable C++17 class type deduction:
    pseudo_method( F&& fin ):f(std::move(fin)) {}

    // Koenig lookup operator->*, as this is a pseudo-method it is appropriate:
    template<class Variant> // maybe add SFINAE test that LHS is actually a variant.
    friend decltype(auto) operator->*( Variant&& var, pseudo_method const& method ) {
        // var->*method returns a lambda that perfect forwards a function call,
        // behaving like a method pointer basically:
        return [&](auto&&...args)->decltype(auto) {
            // use visit to get the type of the variant:
            return std::visit(
                [&](auto&& self)->decltype(auto) {
                    // decltype(x)(x) is perfect forwarding in a lambda:
                    return method.f( decltype(self)(self), decltype(args)(args)... );
                },
                std::forward<Var>(var)
            );
        };
    };
}
};

print

```

```

template<class F>
struct pseudo_method {
    F f;
    // enable C++17 class type deduction:
    pseudo_method( F&& fin ):f(std::move(fin)) {}

    // Koenig lookup operator->*, as this is a pseudo-method it is appropriate:
    template<class Variant> // maybe add SFINAE test that LHS is actually a variant.
    friend decltype(auto) operator->*( Variant&& var, pseudo_method const& method ) {
        // var->*method returns a lambda that perfect forwards a function call,
        // behaving like a method pointer basically:

```

```

        return [&] (auto&&...args)->decltype(auto) {
            // use visit to get the type of the variant:
            return std::visit(
                [&] (auto&& self)->decltype(auto) {
                    // decltype(x)(x) is perfect forwarding in a lambda:
                    return method.f( decltype(self)(self), decltype(args)(args)... );
                },
                std::forward<Var>(var)
            );
        };
    };
}
;

```

◦

```

template<class F>
struct pseudo_method {
    F f;
    // enable C++17 class type deduction:
    pseudo_method( F&& fin ):f(std::move(fin)) {}

    // Koenig lookup operator->*, as this is a pseudo-method it is appropriate:
    template<class Variant> // maybe add SFINAE test that LHS is actually a variant.
    friend decltype(auto) operator->*( Variant&& var, pseudo_method const& method ) {
        // var->*method returns a lambda that perfect forwards a function call,
        // behaving like a method pointer basically:
        return [&] (auto&&...args)->decltype(auto) {
            // use visit to get the type of the variant:
            return std::visit(
                [&] (auto&& self)->decltype(auto) {
                    // decltype(x)(x) is perfect forwarding in a lambda:
                    return method.f( decltype(self)(self), decltype(args)(args)... );
                },
                std::forward<Var>(var)
            );
        };
    };
}
;

```

A::print(std::cout) ◦ B{}var B::print(std::cout) ◦

## C

```

template<class F>
struct pseudo_method {
    F f;
    // enable C++17 class type deduction:
    pseudo_method( F&& fin ):f(std::move(fin)) {}

    // Koenig lookup operator->*, as this is a pseudo-method it is appropriate:
    template<class Variant> // maybe add SFINAE test that LHS is actually a variant.
    friend decltype(auto) operator->*( Variant&& var, pseudo_method const& method ) {
        // var->*method returns a lambda that perfect forwards a function call,
        // behaving like a method pointer basically:
        return [&] (auto&&...args)->decltype(auto) {
            // use visit to get the type of the variant:
            return std::visit(
                [&] (auto&& self)->decltype(auto) {

```

```

        // decltype(x)(x) is perfect forwarding in a lambda:
        return method.f( decltype(self)(self), decltype(args)(args)... );
    },
    std::forward<Var>(var)
);
};

}
;

```

```

template<class F>
struct pseudo_method {
    F f;
    // enable C++17 class type deduction:
    pseudo_method( F&& fin ):f(std::move(fin)) {}

    // Koenig lookup operator->*, as this is a pseudo-method it is appropriate:
    template<class Variant> // maybe add SFINAE test that LHS is actually a variant.
    friend decltype(auto) operator->*( Variant&& var, pseudo_method const& method ) {
        // var->*method returns a lambda that perfect forwards a function call,
        // behaving like a method pointer basically:
        return [&](auto&&...args)->decltype(auto) {
            // use visit to get the type of the variant:
            return std::visit(
                [&](auto&& self)->decltype(auto) {
                    // decltype(x)(x) is perfect forwarding in a lambda:
                    return method.f( decltype(self)(self), decltype(args)(args)... );
                },
                std::forward<Var>(var)
            );
        };
    }
};

```

C.print(std::cout)。

print printf constexpr。

boost::variantstd::variant。

## **`std :: variant`**

◦

```

struct A {};
struct B { B()=default; B(B const&)=default; B(int){}; };
struct C { C()=delete; C(int) {}; C(C const&)=default; };
struct D { D( std::initializer_list<int> ) {}; D(D const&)=default; D()=default; };

std::variant<A,B> var_ab0; // contains a A()
std::variant<A,B> var_ab1 = 7; // contains a B(7)
std::variant<A,B> var_ab2 = var_ab1; // contains a B(7)
std::variant<A,B,C> var_abc0{ std::in_place_type<C>, 7 }; // contains a C(7)
std::variant<C> var_c0; // illegal, no default ctor for C
std::variant<A,D> var_ad0( std::in_place_type<D>, {1,3,3,4} ); // contains D{1,3,3,4}
std::variant<A,D> var_ad1( std::in_place_index<0> ); // contains A{}
std::variant<A,D> var_ad2( std::in_place_index<1>, {1,3,3,4} ); // contains D{1,3,3,4}

```

**std ::** <https://riptutorial.com/zh-TW/cplusplus/topic/5239/std--->

# 113: std ::

```
class T  
std::size_t N
```

```
std::array#include <array><array>o
```

## Examples

### std :: array

```
std::array<T, N> T NT
```

```
Tstd::array
```

```
// 1) Using aggregate-initialization  
std::array<int, 3> a{ 0, 1, 2 };  
// or equivalently  
std::array<int, 3> a = { 0, 1, 2 };  
  
// 2) Using the copy constructor  
std::array<int, 3> a{ 0, 1, 2 };  
std::array<int, 3> a2(a);  
// or equivalently  
std::array<int, 3> a2 = a;  
  
// 3) Using the move constructor  
std::array<int, 3> a = std::array<int, 3>{ 0, 1, 2 };
```

---

```
std::array<T, N> T NT
```

```
Tstd::array
```

```
// 1) Using aggregate-initialization  
std::array<int, 3> a{ 0, 1, 2 };  
// or equivalently  
std::array<int, 3> a = { 0, 1, 2 };  
  
// 2) Using the copy constructor  
std::array<int, 3> a{ 0, 1, 2 };  
std::array<int, 3> a2(a);  
// or equivalently  
std::array<int, 3> a2 = a;  
  
// 3) Using the move constructor  
std::array<int, 3> a = std::array<int, 3>{ 0, 1, 2 };
```

**at (pos)**

```
pos. posstd::out_of_range.
```

## O1.

```
#include <array>

int main()
{
    std::array<int, 3> arr;

    // write values
    arr.at(0) = 2;
    arr.at(1) = 4;
    arr.at(2) = 6;

    // read values
    int a = arr.at(0); // a is now 2
    int b = arr.at(1); // b is now 4
    int c = arr.at(2); // c is now 6

    return 0;
}
```

## 2 operator[pos]

```
pos. pos. at(pos).
```

## O1.

```
#include <array>

int main()
{
    std::array<int, 3> arr;

    // write values
    arr.at(0) = 2;
    arr.at(1) = 4;
    arr.at(2) = 6;

    // read values
    int a = arr.at(0); // a is now 2
    int b = arr.at(1); // b is now 4
    int c = arr.at(2); // c is now 6

    return 0;
}
```

## 3 std::get<pos>

```
pos. pos.
```

## O1.

```
#include <array>
```

```

int main()
{
    std::array<int, 3> arr;

    // write values
    arr.at(0) = 2;
    arr.at(1) = 4;
    arr.at(2) = 6;

    // read values
    int a = arr.at(0); // a is now 2
    int b = arr.at(1); // b is now 4
    int c = arr.at(2); // c is now 6

    return 0;
}

```

## 4 front()

- front() ◦

O1◦

Cc.front()\*c.begin() ◦

```

#include <array>

int main()
{
    std::array<int, 3> arr;

    // write values
    arr.at(0) = 2;
    arr.at(1) = 4;
    arr.at(2) = 6;

    // read values
    int a = arr.at(0); // a is now 2
    int b = arr.at(1); // b is now 4
    int c = arr.at(2); // c is now 6

    return 0;
}

```

## 5 back()

- back() ◦

O1◦

```

#include <array>

int main()
{
    std::array<int, 3> arr;

```

```

// write values
arr.at(0) = 2;
arr.at(1) = 4;
arr.at(2) = 6;

// read values
int a = arr.at(0); // a is now 2
int b = arr.at(1); // b is now 4
int c = arr.at(2); // c is now 6

return 0;
}

```

## 6 data()

◦ range [data(); data() + size())data()

O1.

```

#include <array>

int main()
{
    std::array<int, 3> arr;

    // write values
    arr.at(0) = 2;
    arr.at(1) = 4;
    arr.at(2) = 6;

    // read values
    int a = arr.at(0); // a is now 2
    int b = arr.at(1); // b is now 4
    int c = arr.at(2); // c is now 6

    return 0;
}

```

C std::arraysize()size()

```

int main() {
    std::array<int, 3> arr = { 1, 2, 3 };
    cout << arr.size() << endl;
}

```

std::array**STLfor**vector

```

int main() {
    std::array<int, 3> arr = { 1, 2, 3 };
    for (auto i : arr)
        cout << i << '\n';
}

```

fill()std::array

```
int main() {  
  
    std::array<int, 3> arr = { 1, 2, 3 };  
    // change all elements of the array to 100  
    arr.fill(100);  
  
}
```

std :: <https://riptutorial.com/zh-TW/cplusplus/topic/2712/std--->

## Examples

C++.

- T&& T T& ◦
- ◦
- std::move(obj) ◦ ◦ ◦
- std::move◦ std::move(obj) objauto obj2 = std::move(obj) ◦
- 

```
class A {
public:
    int a;
    int b;

    A(const A &other) {
        this->a = other.a;
        this->b = other.b;
    }
};
```

AA.

A(const A &) = default;◦

◦

```
class A {
public:
    int a;
    int b;

    A(const A &other) {
        this->a = other.a;
        this->b = other.b;
    }
};
```

zero◦ Wallet(Wallet&&) = default; nrOfDollarsPOD◦

“”◦ ◦

```
class A {
public:
```

```
int a;
int b;

A(const A &other) {
    this->a = other.a;
    this->b = other.b;
}

};
```

◦

◦ ◦

```
class A {
public:
    int a;
    int b;

A(const A &other) {
    this->a = other.a;
    this->b = other.b;
}

};
```

◦ ◦ ◦

operator ==

```
class A {
    int a;
    A& operator= (A&& other) {
        this->a = other.a;
        other.a = 0;
        return *this;
    }
};
```

◦ operator ==

```
class A {
    int a;
    A& operator= (A&& other) {
        this->a = other.a;
        other.a = 0;
        return *this;
    }
};
```

◦

## std :: moveOn<sup>2</sup>On

C ++ 11◦ oO'Ø◦

- `&& std::string&&std::string`
- `T( T&& )`
- `auto operator=(T&&) -> T&.`

`<utility>std::move. .`

---

**O n nO1. nÓN<sup>2</sup>On .**

## 2013919Dobbs Journal“” Andrew Koenig. . Collatz

```
// Based on an example by Andrew Koenig in his Dr. Dobbs Journal article
// "Containers That Never Change" September 19, 2013, available at
// <url: http://www.drdobbs.com/cpp/containters-that-never-change/240161543>

// Includes here, e.g. <vector>

namespace my {
    template< class Item >
    using Vector_ = /* E.g. std::vector<Item> */;

    auto concat( Vector_<int> const& v, int const x )
        -> Vector_<int>
    {
        auto result{ v };
        result.push_back( x );
        return result;
    }

    auto collatz_aux( int const n, Vector_<int> const& result )
        -> Vector_<int>
    {
        if( n == 1 )
        {
            return result;
        }
        auto const new_result = concat( result, n );
        if( n % 2 == 0 )
        {
            return collatz_aux( n/2, new_result );
        }
        else
        {
            return collatz_aux( 3*n + 1, new_result );
        }
    }

    auto collatz( int const n )
        -> Vector_<int>
    {
        assert( n != 0 );
        return collatz_aux( n, Vector_<int>() );
    }
} // namespace my

#include <iostream>
using namespace std;
```

```

auto main() -> int
{
    for( int const x : my::collatz( 42 ) )
    {
        cout << x << ' ';
    }
    cout << '\n';
}

```

```

// Based on an example by Andrew Koenig in his Dr. Dobbs Journal article
// "Containers That Never Change" September 19, 2013, available at
// <url: http://www.drdobbs.com/cpp/containers-that-never-change/240161543>

// Includes here, e.g. <vector>

namespace my {
    template< class Item >
    using Vector_ = /* E.g. std::vector<Item> */;

    auto concat( Vector_<int> const& v, int const x )
        -> Vector_<int>
    {
        auto result{ v };
        result.push_back( x );
        return result;
    }

    auto collatz_aux( int const n, Vector_<int> const& result )
        -> Vector_<int>
    {
        if( n == 1 )
        {
            return result;
        }
        auto const new_result = concat( result, n );
        if( n % 2 == 0 )
        {
            return collatz_aux( n/2, new_result );
        }
        else
        {
            return collatz_aux( 3*n + 1, new_result );
        }
    }

    auto collatz( int const n )
        -> Vector_<int>
    {
        assert( n != 0 );
        return collatz_aux( n, Vector_<int>() );
    }
} // namespace my

#include <iostream>
using namespace std;
auto main() -> int
{
    for( int const x : my::collatz( 42 ) )
    {
        cout << x << ' ';
    }
}
```

```

    }
    cout << '\n';
}

```

$O(n)$

g++ Visual C++ `collatz(42) collatz(42) 8Collatz368 * collatz(42) = 28.`

- const ◦ ◦ std::move

```

// Based on an example by Andrew Koenig in his Dr. Dobbs Journal article
// "Containers That Never Change" September 19, 2013, available at
// <url: http://www.drdobbs.com/cpp/containers-that-never-change/240161543>

// Includes here, e.g. <vector>

namespace my {
    template< class Item >
    using Vector_ = /* E.g. std::vector<Item> */;

    auto concat( Vector_<int> const& v, int const x )
        -> Vector_<int>
    {
        auto result{ v };
        result.push_back( x );
        return result;
    }

    auto collatz_aux( int const n, Vector_<int> const& result )
        -> Vector_<int>
    {
        if( n == 1 )
        {
            return result;
        }
        auto const new_result = concat( result, n );
        if( n % 2 == 0 )
        {
            return collatz_aux( n/2, new_result );
        }
        else
        {
            return collatz_aux( 3*n + 1, new_result );
        }
    }

    auto collatz( int const n )
        -> Vector_<int>
    {
        assert( n != 0 );
        return collatz_aux( n, Vector_<int>() );
    }
} // namespace my

#include <iostream>
using namespace std;
auto main() -> int
{
    for( int const x : my::collatz( 42 ) )

```

```

    {
        cout << x << ' ';
    }
    cout << '\n';
}

```

## g ++Visual C ++0。

*OnCollatz*  $O(n^2 \rightarrow O(n))$

---

◦ C ++ 14C ++◦ struct**move**struct result;◦

C ++const◦

---

```

// Based on an example by Andrew Koenig in his Dr. Dobbs Journal article
// "Containers That Never Change" September 19, 2013, available at
// <url: http://www.drdobbs.com/cpp/containters-that-never-change/240161543>

// Includes here, e.g. <vector>

namespace my {
    template< class Item >
    using Vector_ = /* E.g. std::vector<Item> */;

    auto concat( Vector_<int> const& v, int const x )
        -> Vector_<int>
    {
        auto result{ v };
        result.push_back( x );
        return result;
    }

    auto collatz_aux( int const n, Vector_<int> const& result )
        -> Vector_<int>
    {
        if( n == 1 )
        {
            return result;
        }
        auto const new_result = concat( result, n );
        if( n % 2 == 0 )
        {
            return collatz_aux( n/2, new_result );
        }
        else
        {
            return collatz_aux( 3*n + 1, new_result );
        }
    }

    auto collatz( int const n )
        -> Vector_<int>
    {
        assert( n != 0 );
        return collatz_aux( n, Vector_<int>() );
    }
}

```

```

} // namespace my

#include <iostream>
using namespace std;
auto main() -> int
{
    for( int const x : my::collatz( 42 ) )
    {
        cout << x << ' ';
    }
    cout << '\n';
}

```

```

void print(const std::vector<int>& vec) {
    for (auto&& val : vec) {
        std::cout << val << ", ";
    }
    std::cout << std::endl;
}

int main() {
    // initialize vec1 with 1, 2, 3, 4 and vec2 as an empty vector
    std::vector<int> vec1{1, 2, 3, 4};
    std::vector<int> vec2;

    // The following line will print 1, 2, 3, 4
    print(vec1);

    // The following line will print a new line
    print(vec2);

    // The vector vec2 is assigned with move assingment.
    // This will "steal" the value of vec1 without copying it.
    vec2 = std::move(vec1);

    // Here the vec1 object is in an indeterminate state, but still valid.
    // The object vec1 is not destroyed,
    // but there's is no guarantees about what it contains.

    // The following line will print 1, 2, 3, 4
    print(vec2);
}

```

```

void consumingFunction(std::vector<int> vec) {
    // Some operations
}

int main() {
    // initialize vec with 1, 2, 3, 4
    std::vector<int> vec{1, 2, 3, 4};

    // Send the vector by move
    consumingFunction(std::move(vec));

    // Here the vec object is in an indeterminate state.
    // Since the object is not destroyed, we can assign it a new content.
    // We will, in this case, assign an empty value to the vector,
    // making it effectively empty
    vec = {};
}

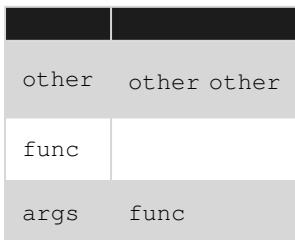
```

```
// Since the vector has gained a determinate value, we can use it normally.  
vec.push_back(42);  
  
// Send the vector by move again.  
consumingFunction(std::move(vec));  
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2129/>

# 115:

- 
- &&
- && funcArgs && ... args



- std::thread
- std::threadjoin

## Examples

◦  
- ◦

```
int n;
std::thread thread{ calculateSomething, std::ref(n) };

//Doing some other stuff

//We need 'n' now!
//Wait for the thread to finish - if it is not already done
thread.join();

//Now 'n' has the result of the calculation done in the seperate thread
std::cout << n << '\n';
```

---

detach

```
int n;
std::thread thread{ calculateSomething, std::ref(n) };

//Doing some other stuff

//We need 'n' now!
//Wait for the thread to finish - if it is not already done
thread.join();

//Now 'n' has the result of the calculation done in the seperate thread
std::cout << n << '\n';
```

const std::thread<sup>o</sup> std::reference\_wrapper

```
void foo(int& b)
{
    b = 10;
}

int a = 1;
std::thread thread{ foo, std::ref(a) }; // 'a' is now really passed as reference

thread.join();
std::cout << a << '\n'; // Outputs 10
```

```
void foo(int& b)
{
    b = 10;
}

int a = 1;
std::thread thread{ foo, std::ref(a) }; // 'a' is now really passed as reference

thread.join();
std::cout << a << '\n'; // Outputs 10
```

## **std :: thread**

## C++`std::thread`

- - 
  - Functor
  - Lambda

```
#include <iostream>
#include <thread>

void foo(int a)
{
    std::cout << a << '\n';
}

int main()
{
    // Create and execute the thread
    std::thread thread(foo, 10); // foo is the function to execute, 10 is the
                                // argument to pass to it

    // Keep going; the thread is executed separately

    // Wait for the thread to finish; we stay here until it is done
    thread.join();

    return 0;
}
```

```
#include <iostream>
#include <thread>

void foo(int a)
{
    std::cout << a << '\n';
}

int main()
{
    // Create and execute the thread
    std::thread thread(foo, 10); // foo is the function to execute, 10 is the
                                // argument to pass to it

    // Keep going; the thread is executed separately

    // Wait for the thread to finish; we stay here until it is done
    thread.join();

    return 0;
}
```

---

## Functor

```
#include <iostream>
#include <thread>

void foo(int a)
{
    std::cout << a << '\n';
}

int main()
{
    // Create and execute the thread
    std::thread thread(foo, 10); // foo is the function to execute, 10 is the
                                // argument to pass to it

    // Keep going; the thread is executed separately

    // Wait for the thread to finish; we stay here until it is done
    thread.join();

    return 0;
}
```

---

## Lambda

```
#include <iostream>
#include <thread>

void foo(int a)
{
    std::cout << a << '\n';
```

```

}

int main()
{
    // Create and execute the thread
    std::thread thread(foo, 10); // foo is the function to execute, 10 is the
                                // argument to pass to it

    // Keep going; the thread is executed separately

    // Wait for the thread to finish; we stay here until it is done
    thread.join();

    return 0;
}

```

std::this\_thread::

|             |           |
|-------------|-----------|
| get_id      | <b>id</b> |
| sleep_for   |           |
| sleep_until |           |
| yield       |           |

std::this\_thread::get\_id**id**

```

void foo()
{
    //Print this threads id
    std::cout << std::this_thread::get_id() << '\n';
}

std::thread thread{ foo };
thread.join(); //'threads' id has now been printed, should be something like 12556

foo(); //The id of the main thread is printed, should be something like 2420

```

std::this\_thread::sleep\_for**3**

```

void foo()
{
    //Print this threads id
    std::cout << std::this_thread::get_id() << '\n';
}

std::thread thread{ foo };
thread.join(); //'threads' id has now been printed, should be something like 12556

foo(); //The id of the main thread is printed, should be something like 2420

```

std::this\_thread::sleep\_until**3**

```

void foo()
{
    //Print this threads id
    std::cout << std::this_thread::get_id() << '\n';
}

std::thread thread{ foo };
thread.join(); //'threads' id has now been printed, should be something like 12556

foo(); //The id of the main thread is printed, should be something like 2420

```

```

std::this_thread::yield

void foo()
{
    //Print this threads id
    std::cout << std::this_thread::get_id() << '\n';
}

std::thread thread{ foo };
thread.join(); //'threads' id has now been printed, should be something like 12556

foo(); //The id of the main thread is printed, should be something like 2420

```

## std :: asyncstd :: thread

std::async std::thread

```

#include <future>
#include <iostream>

unsigned int square(unsigned int i){
    return i*i;
}

int main() {
    auto f = std::async(std::launch::async, square, 8);
    std::cout << "square currently running\n"; //do something while square is running
    std::cout << "result is " << f.get() << '\n'; //getting the result from square
}

```

- std::asyncstd::future . future

```

#include <future>
#include <iostream>

unsigned int square(unsigned int i){
    return i*i;
}

int main() {
    auto f = std::async(std::launch::async, square, 8);
    std::cout << "square currently running\n"; //do something while square is running
    std::cout << "result is " << f.get() << '\n'; //getting the result from square
}

```

- std::async(std::async(square, 5)); . . . std::launch::async
- .

std::thread join() detach() . std::terminate . **RAII**

```
class thread_joiner
{
public:
    thread_joiner(std::thread t)
        : t_(std::move(t))
    { }

    ~thread_joiner()
    {
        if(t_.joinable()) {
            t_.join();
        }
    }

private:
    std::thread t_;
}
```

```
class thread_joiner
{
public:
    thread_joiner(std::thread t)
        : t_(std::move(t))
    { }

    ~thread_joiner()
    {
        if(t_.joinable()) {
            t_.join();
        }
    }

private:
    std::thread t_;
}
```

,t() join()

.

joinable std::terminate .

```
#include <thread>

void foo()
{
```

```

        std::this_thread::sleep_for(std::chrono::seconds(3));
    }
    //create 100 thread objects that do nothing
    std::thread executors[100];

    // Some code

    // I want to create some threads now

    for (int i = 0; i < 100; i++)
    {
        // If this object doesn't have a thread assigned
        if (!executors[i].joinable())
            executors[i] = std::thread(foo);
    }
}

```

- std::mutex ◦ ◦ std::lock\_guard

```

std::mutex m;
void worker() {
    std::lock_guard<std::mutex> guard(m); // Acquires a lock on the mutex
    // Synchronized code here
} // the mutex is automatically released when guard goes out of scope

```

- std::lock\_guard ◦ std::unique\_lock

```

std::mutex m;
void worker() {
    std::lock_guard<std::mutex> guard(m); // Acquires a lock on the mutex
    // Synchronized code here
} // the mutex is automatically released when guard goes out of scope

```

- ◦

- std::unique\_lock<std::mutex> std::condition\_variable ◦ ◦

- std::thread std::condition\_variable std::mutex◦

```

#include <condition_variable>
#include <cstddef>
#include <iostream>
#include <mutex>
#include <queue>
#include <random>
#include <thread>

int main()
{
    std::condition_variable cond;
    std::mutex mtx;
    std::queue<int> intq;
    bool stopped = false;

    std::thread producer{ [&] () {
    {

```

```

// Prepare a random number generator.
// Our producer will simply push random numbers to intq.
//
std::default_random_engine gen{};
std::uniform_int_distribution<int> dist{};

std::size_t count = 4006;
while(count--)
{
    // Always lock before changing
    // state guarded by a mutex and
    // condition_variable (a.k.a. "condvar").
    std::lock_guard<std::mutex> L{mtx};

    // Push a random int into the queue
    intq.push(dist(gen));

    // Tell the consumer it has an int
    cond.notify_one();
}

// All done.
// Acquire the lock, set the stopped flag,
// then inform the consumer.
std::lock_guard<std::mutex> L{mtx};

std::cout << "Producer is done!" << std::endl;

stopped = true;
cond.notify_one();
};

std::thread consumer{ [&] ()
{
    do{
        std::unique_lock<std::mutex> L{mtx};
        cond.wait(L, [&] ())
        {
            // Acquire the lock only if
            // we've stopped or the queue
            // isn't empty
            return stopped || ! intq.empty();
        });
        // We own the mutex here; pop the queue
        // until it empties out.

        while( ! intq.empty())
        {
            const auto val = intq.front();
            intq.pop();

            std::cout << "Consumer popped: " << val << std::endl;
        }

        if(stopped){
            // producer has signaled a stop
            std::cout << "Consumer is done!" << std::endl;
            break;
        }
    }
}};


```

```

        }while(true);
    }};

consumer.join();
producer.join();

std::cout << "Example Completed!" << std::endl;

return 0;
}

```

## C++ 11.

## C++ 14

```

struct tasks {
    // the mutex, condition variable and deque form a single
    // thread-safe triggered queue of tasks:
    std::mutex m;
    std::condition_variable v;
    // note that a packaged_task<void> can store a packaged_task<R>:
    std::deque<std::packaged_task<void()>> work;

    // this holds futures representing the worker threads being done:
    std::vector<std::future<void>> finished;

    // queue( lambda ) will enqueue the lambda into the tasks for the threads
    // to use. A future of the type the lambda returns is given to let you get
    // the result out.
    template<class F, class R=std::result_of_t<F&()>>
    std::future<R> queue(F&& f) {
        // wrap the function object into a packaged task, splitting
        // execution from the return value:
        std::packaged_task<R()> p(std::forward<F>(f));

        auto r=p.get_future(); // get the return value before we hand off the task
        {
            std::unique_lock<std::mutex> l(m);
            work.emplace_back(std::move(p)); // store the task<R()> as a task<void()>
        }
        v.notify_one(); // wake a thread to work on the task
    }

    return r; // return the future result of the task
}

// start N threads in the thread pool.
void start(std::size_t N=1){
    for (std::size_t i = 0; i < N; ++i)
    {
        // each thread is a std::async running this->thread_task():
        finished.push_back(
            std::async(
                std::launch::async,
                [this]{ thread_task(); }
            )
        );
    }
}

// abort() cancels all non-started tasks, and tells every working thread
// stop running, and waits for them to finish up.

```

```

void abort() {
    cancel_pending();
    finish();
}
// cancel_pending() merely cancels all non-started tasks:
void cancel_pending() {
    std::unique_lock<std::mutex> l(m);
    work.clear();
}
// finish enqueues a "stop the thread" message for every thread, then waits for them:
void finish() {
{
    std::unique_lock<std::mutex> l(m);
    for(auto&&unused:finished) {
        work.push_back({});
    }
}
v.notify_all();
finished.clear();
}
~tasks() {
    finish();
}
private:
// the work that a worker thread does:
void thread_task() {
    while(true) {
        // pop a task off the queue:
        std::packaged_task<void()> f;
        {
            // usual thread-safe queue code:
            std::unique_lock<std::mutex> l(m);
            if (work.empty()){
                v.wait(l,[&]{return !work.empty();});
            }
            f = std::move(work.front());
            work.pop_front();
        }
        // if the task is invalid, it means we are asked to abort:
        if (!f.valid()) return;
        // otherwise, run the task:
        f();
    }
}
};


```

tasks.queue( []{ return "hello world"s; } )std::future<std::string> **task**hello world。

tasks.start(10) **10**◦

packaged\_task<void()>std::function◦ packaged\_task<void()>◦

◦

**C++11**

**C++11**result\_of\_t<blah>typename result\_of<blah>::type◦

◦

thread\_local ◦ thread\_local◦

- ◦
- ◦ extern ◦
- ◦
- ◦
- ◦ ◦

```
void debug_counter() {  
    thread_local int count = 0;  
    Logger::log("This function has been called %d times by this thread", ++count);  
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/699/>

◦ ◦

## Examples

### std :: shared\_lock

shared\_lock◦

```
#include <unordered_map>
#include <mutex>
#include <shared_mutex>
#include <thread>
#include <string>
#include <iostream>

class PhoneBook {
public:
    string getPhoneNo( const std::string & name )
    {
        shared_lock<shared_timed_mutex> r(_protect);
        auto it = _phonebook.find( name );
        if ( it == _phonebook.end() )
            return (*it).second;
        return "";
    }
    void addPhoneNo ( const std::string & name, const std::string & phone )
    {
        unique_lock<shared_timed_mutex> w(_protect);
        _phonebook[name] = phone;
    }

    shared_timed_mutex _protect;
    unordered_map<string,string> _phonebook;
};
```

### std :: call\_oncestd :: once\_flag

std::call\_once◦ std::system\_error◦

std::once\_flag◦

```
#include <mutex>
#include <iostream>

std::once_flag flag;
void do_something()
{
    std::call_once(flag, [](){std::cout << "Happens once" << std::endl;});

    std::cout << "Happens every time" << std::endl;
}
```

```
o  
o o o  
  
class text_buffer  
{  
    // for readability/maintainability  
    using mutex_type = std::shared_timed_mutex;  
    using reading_lock = std::shared_lock<mutex_type>;  
    using updates_lock = std::unique_lock<mutex_type>;  
  
public:  
    // This returns a scoped lock that can be shared by multiple  
    // readers at the same time while excluding any writers  
    [[nodiscard]]  
    reading_lock lock_for_reading() const { return reading_lock(mtx); }  
  
    // This returns a scoped lock that is exclusive to one  
    // writer preventing any readers  
    [[nodiscard]]  
    updates_lock lock_for_updates() { return updates_lock(mtx); }  
  
    char* data() { return buf; }  
    char const* data() const { return buf; }  
  
    char* begin() { return buf; }  
    char const* begin() const { return buf; }  
  
    char* end() { return buf + sizeof(buf); }  
    char const* end() const { return buf + sizeof(buf); }  
  
    std::size_t size() const { return sizeof(buf); }  
  
private:  
    char buf[1024];  
    mutable mutex_type mtx; // mutable allows const objects to be locked  
};
```

```
o  
  
class text_buffer  
{  
    // for readability/maintainability  
    using mutex_type = std::shared_timed_mutex;  
    using reading_lock = std::shared_lock<mutex_type>;  
    using updates_lock = std::unique_lock<mutex_type>;  
  
public:  
    // This returns a scoped lock that can be shared by multiple  
    // readers at the same time while excluding any writers  
    [[nodiscard]]  
    reading_lock lock_for_reading() const { return reading_lock(mtx); }  
  
    // This returns a scoped lock that is exclusive to one  
    // writer preventing any readers  
    [[nodiscard]]  
    updates_lock lock_for_updates() { return updates_lock(mtx); }
```

```

    char* data() { return buf; }
    char const* data() const { return buf; }

    char* begin() { return buf; }
    char const* begin() const { return buf; }

    char* end() { return buf + sizeof(buf); }
    char const* end() const { return buf + sizeof(buf); }

    std::size_t size() const { return sizeof(buf); }

private:
    char buf[1024];
    mutable mutex_type mtx; // mutable allows const objects to be locked
};


```

◦

```

class text_buffer
{
    // for readability/maintainability
    using mutex_type = std::shared_timed_mutex;
    using reading_lock = std::shared_lock<mutex_type>;
    using updates_lock = std::unique_lock<mutex_type>;

public:
    // This returns a scoped lock that can be shared by multiple
    // readers at the same time while excluding any writers
    [[nodiscard]]
    reading_lock lock_for_reading() const { return reading_lock(mtx); }

    // This returns a scoped lock that is exclusive to one
    // writer preventing any readers
    [[nodiscard]]
    updates_lock lock_for_updates() { return updates_lock(mtx); }

    char* data() { return buf; }
    char const* data() const { return buf; }

    char* begin() { return buf; }
    char const* begin() const { return buf; }

    char* end() { return buf + sizeof(buf); }
    char const* end() const { return buf + sizeof(buf); }

    std::size_t size() const { return sizeof(buf); }

private:
    char buf[1024];
    mutable mutex_type mtx; // mutable allows const objects to be locked
};


```

◦

```

class text_buffer
{
    // for readability/maintainability
    using mutex_type = std::shared_timed_mutex;


```

```

using reading_lock = std::shared_lock<mutex_type>;
using updates_lock = std::unique_lock<mutex_type>;

public:
    // This returns a scoped lock that can be shared by multiple
    // readers at the same time while excluding any writers
    [[nodiscard]]
    reading_lock lock_for_reading() const { return reading_lock(mtx); }

    // This returns a scoped lock that is exclusive to one
    // writer preventing any readers
    [[nodiscard]]
    updates_lock lock_for_updates() { return updates_lock(mtx); }

    char* data() { return buf; }
    char const* data() const { return buf; }

    char* begin() { return buf; }
    char const* begin() const { return buf; }

    char* end() { return buf + sizeof(buf); }
    char const* end() const { return buf + sizeof(buf); }

    std::size_t size() const { return sizeof(buf); }

private:
    char buf[1024];
    mutable mutex_type mtx; // mutable allows const objects to be locked
};

```

## [std :: deferred :: lockstd :: lock](#)

## [std :: condition\\_variable\\_anystd :: cv\\_status](#)

std::condition\_variable std::condition\_variable\_any**BasicLockable**。

std::cv\_status

- [std :: cv\\_status :: no\\_timeout](#)
- [std :: cv\\_status :: no\\_timeout](#)

<https://riptutorial.com/zh-TW/cplusplus/topic/9794/>

117:

C ++. . .

1

- GCCGNU g ++
  - clangLLVM clang ++ C
  - MSVCMicrosoft Visual C ++Visual Studio visual-c ++
  - C ++ BuilderEmbarcadero C ++ Builder RAD Studio c ++ builder

C ++。

Q

## Examples

GCC

main.cpp -Og **GCC**。

```
g++ -o app -Wall main.cpp -O0
```

**-O -O1 -O2 -O3 -Os -Ofast**

```
g++ -o app -Wall main.cpp -O0
```

-O-O-O-O1。

Q° -02-03

```
g++ -o app -Wall main.cpp -O0
```

```
g++ -o app -Wall main.cpp -O0
```

\app\exe Windows /app\ Linux Mac OS

GCCWindows- and Unix- output

mit einem Wallmain aus. 80

file 0

```
g++ -o app -Wall main.cpp -O0
```

[gcc.gnu.org](http://gcc.gnu.org) ° -Og - - - -Ofast °

-Wall ° -Wextra-Wall-Wextra °

C ++-std= ISO C ++° GCC 6.1.0 std= flag<sub>c++98 / c++03 c++11 c++14c++17 / c++1z</sub> ° °

```
g++ -o app -Wall main.cpp -O0
```

GCC-std= flag° gnu++XX XXc++ °

° 6.1.0GCC-std=gnu++03 ;GCC 6.1.0-std=gnu++14 °

GCC -pthreadGCC C ++ 11C ++std::threadstd::wait\_for ° °

—  
-l

```
g++ -o app -Wall main.cpp -O0
```

-L

```
g++ -o app -Wall main.cpp -O0
```

```
g++ -o app -Wall main.cpp -O0
```

--start-group--end-group

```
g++ -o app -Wall main.cpp -O0
```

## Visual C ++

GCCClangVisual StudioIDEVisual C ++ °

Visual Studio° Visual Studio Command Prompt / Developer Command Prompt / x86 Native Tools Command Prompt / x64 Native Tools Command Prompt Visual StudioVC\Program Files (x86)\Microsoft Visual studio x\VC x201010.0201514.0 VCVARSALL °

GCCVisual Studio cl.exe link.exe ° cl.exelink.execllink ° cllinkcllink ° Visual Studio link °

cllink °

[Windows shell“%cd% ° ° clC:\src> %cd%C:\src\ ° ]

main.cpp

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".

cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

## “niam.cpp”

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".

cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".

cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".

cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

/o/Feo-param link/OUT:o-param.exe.dll “” o-paramo /o/Fe**Visual Studio**o /o**GCCClang**o

/oI/Fe/o

---

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".

cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

---

**VCVARSALL**o link;/**MACHINE**o

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".
```

```
cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

/o/Fe◦

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".
```

```
cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

---

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".
```

```
cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

cllink◦

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".
```

```
cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

---

```
cl main.cpp
// Generates object file "main.obj".
// Performs linking with "main.obj".
// Generates executable "main.exe".
```

```
cl /Od main.cpp
// Same as above.
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

---

\* nix/GCC / Clang cl linkVisual Studio-c /c◦ Windows\* nix◦ g++clang++cl◦

```
cl main.cpp
```

```
// Generates object file "main.obj".  
// Performs linking with "main.obj".  
// Generates executable "main.exe".  
  
cl /Od main.cpp  
// Same as above.  
// "/Od" is the "Optimisation: disabled" option, and is the default when no /O is specified.
```

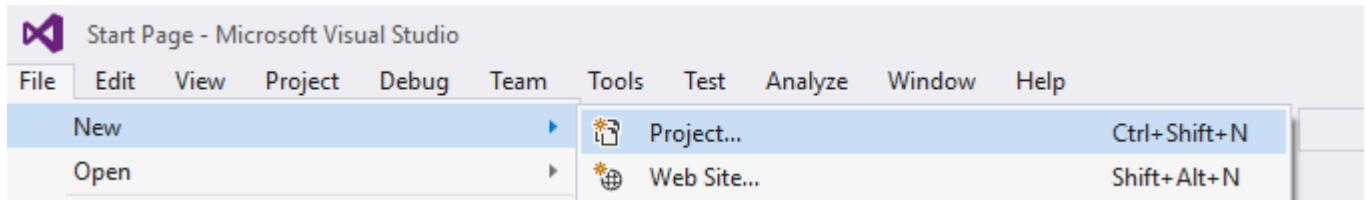
g++clang++/o .

MSVC。Visual C ++ 2015 Update 3/stdo /std:c++14/std:c++latest /std:c++17o

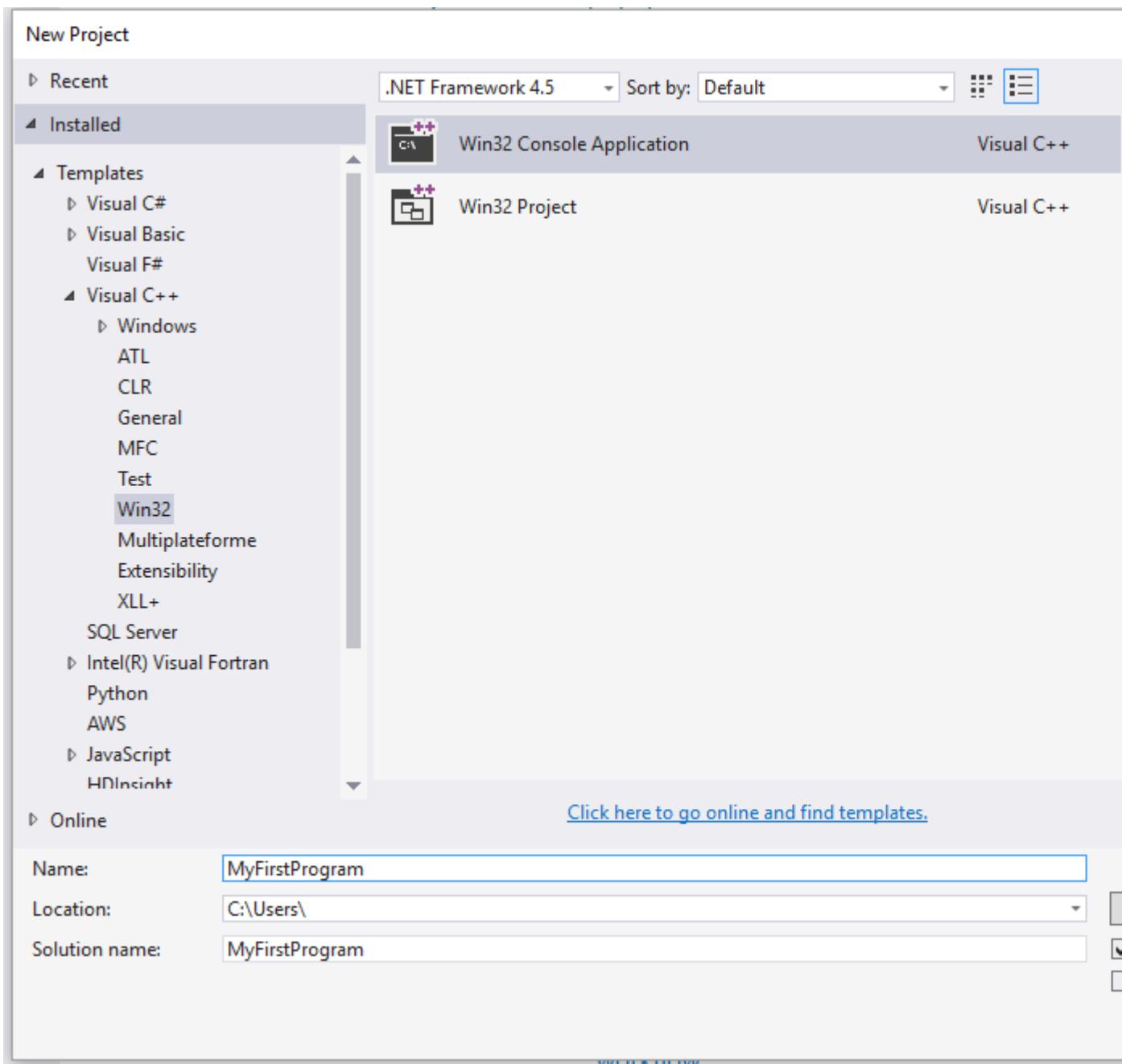
。

## Visual Studio - Hello World

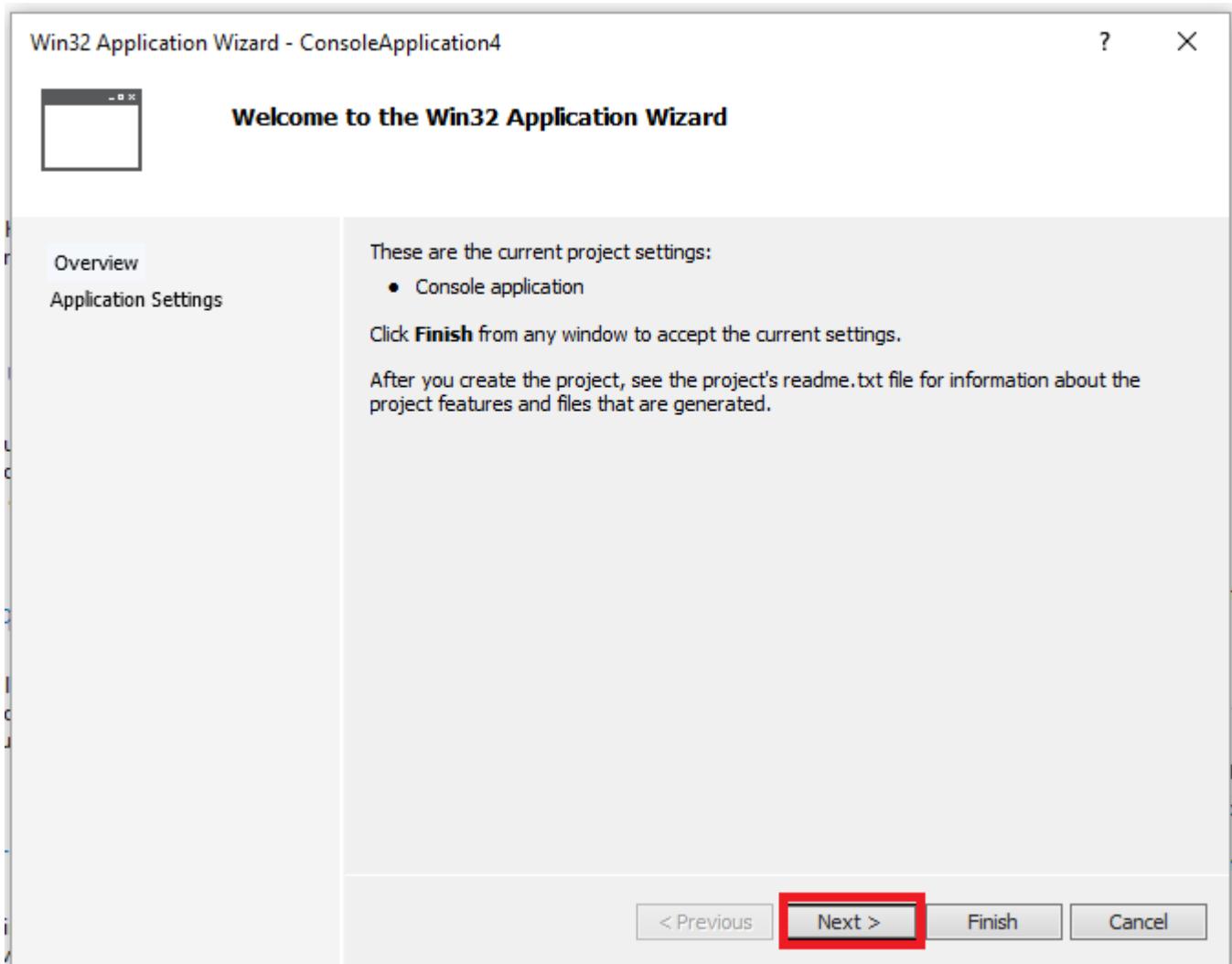
1. [Visual Studio Community 2015](#)
2. Visual Studio
3. - > - >



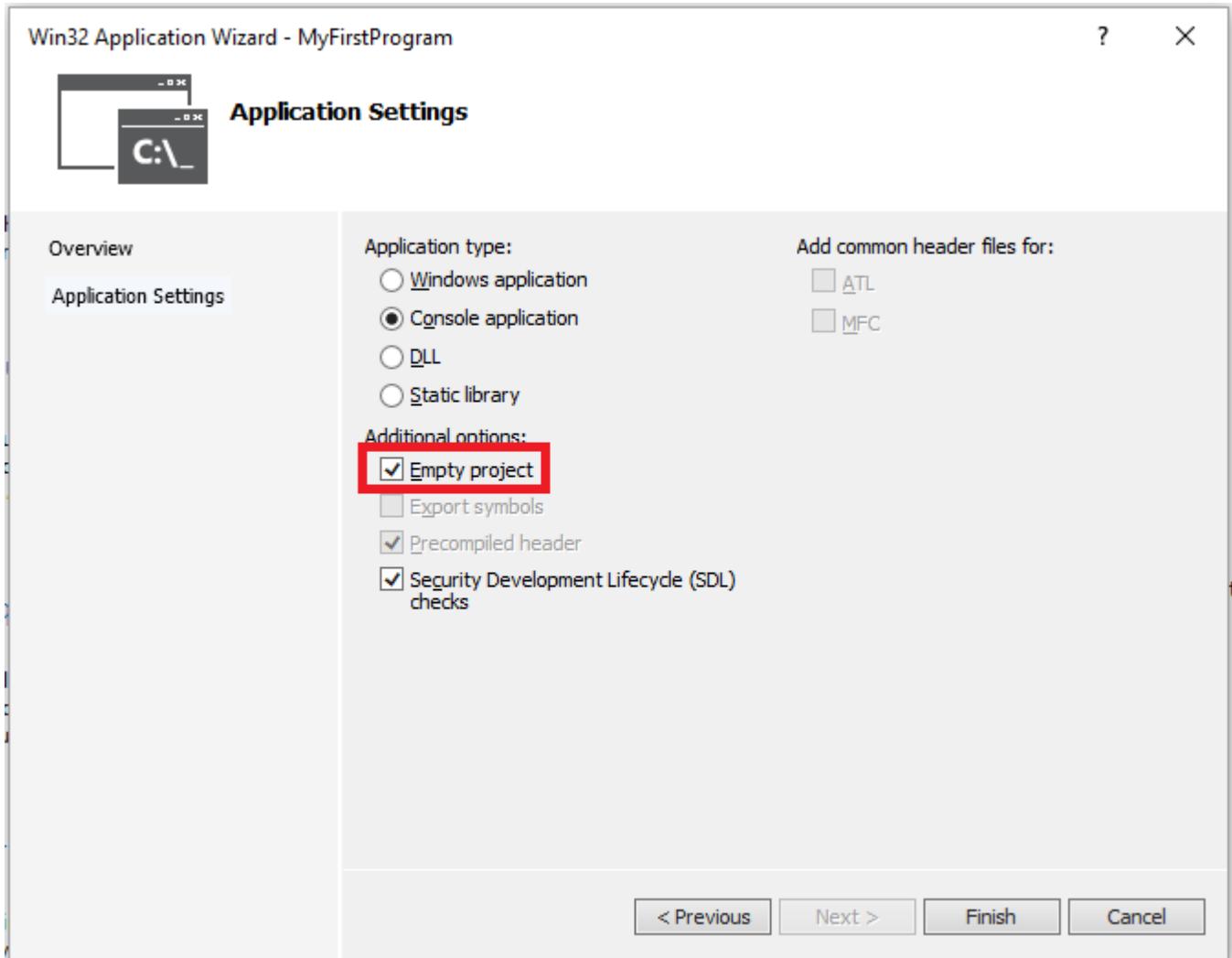
4. - > Visual C ++ - > Win32MyFirstProgram。



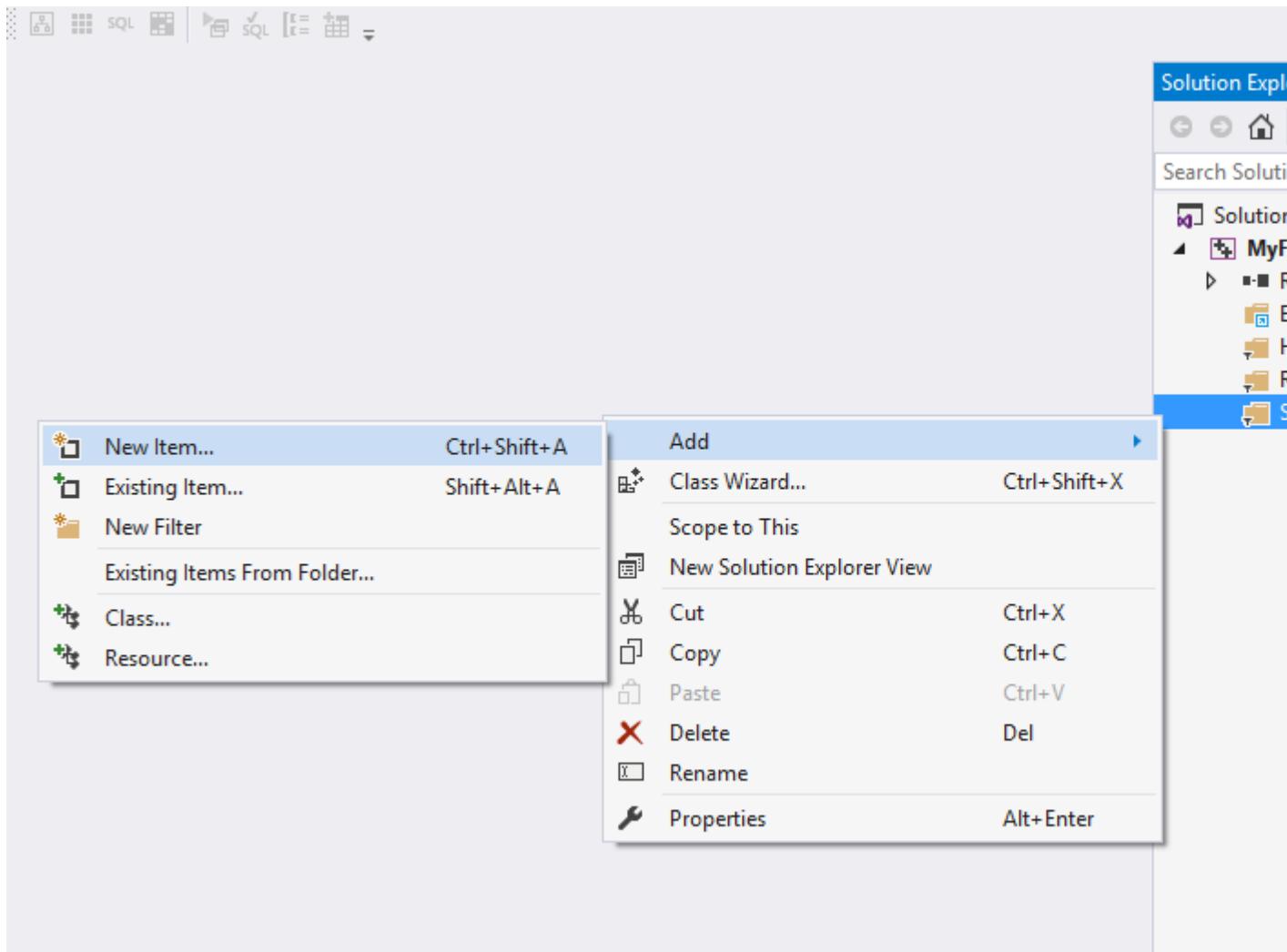
- 5.
6. “”。



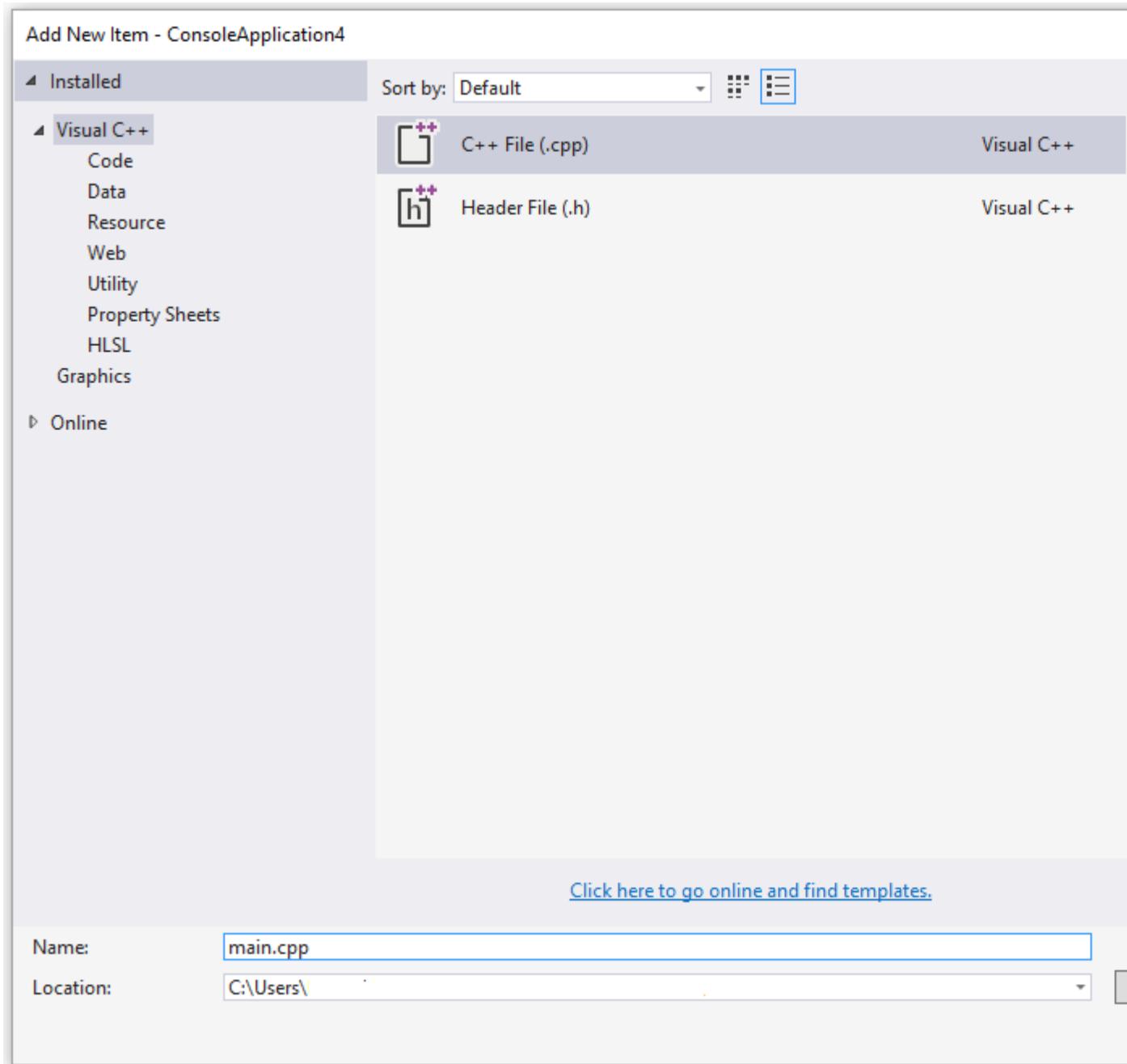
7. Empty project **Finish**



8. ->->



9. C ++ File main.cpp Add



## 10main.cpp

```
#include <iostream>

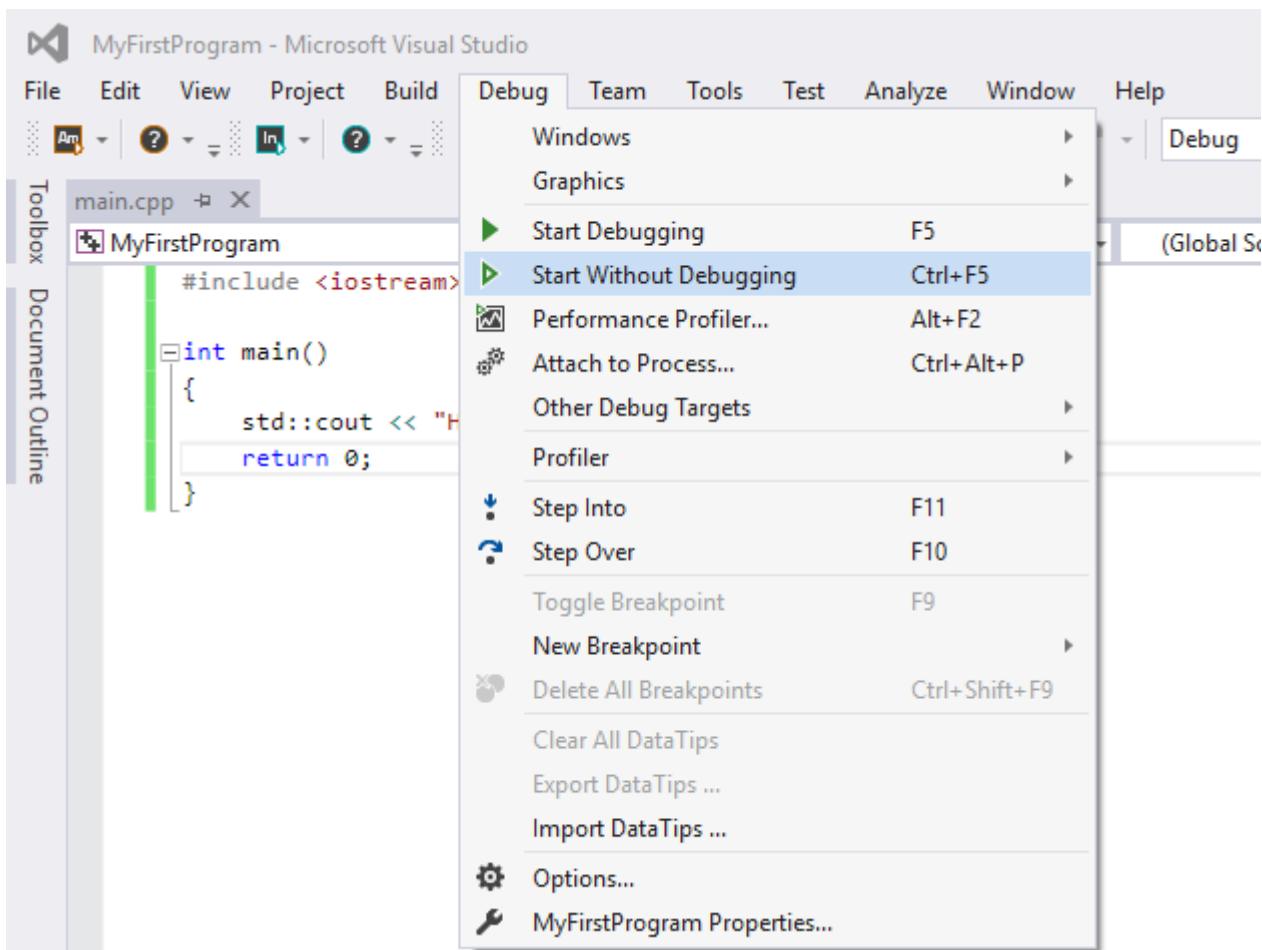
int main()
{
    std::cout << "Hello World!\n";
    return 0;
}
```

The screenshot shows the Microsoft Visual Studio interface. The title bar reads "MyFirstProgram - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Print. The status bar shows "Debug x86". On the left, the Toolbox and Document Outline are visible. The main area shows the code editor with the file "main.cpp" open. The code is:

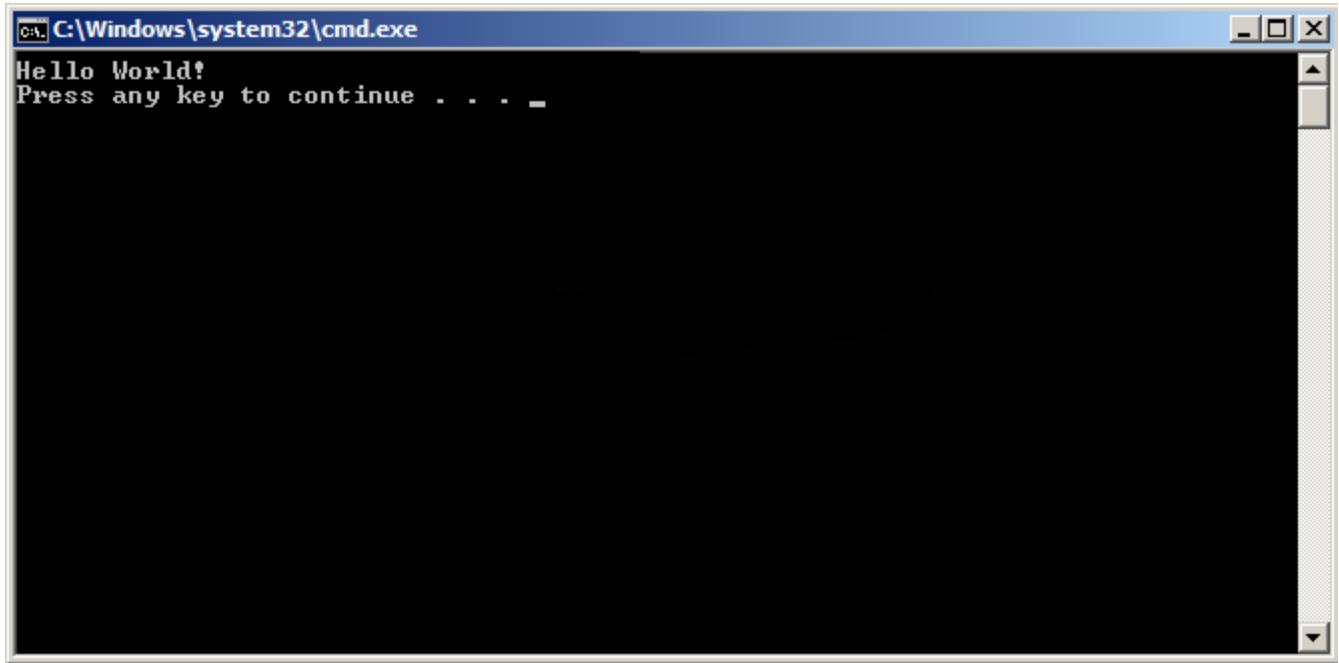
```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
    return 0;
}
```

11. Debug -> Start Without Debugging **ctrl + F5**



12. .



## Clang

ClangGCC clang++ g++ GCC。 -std=version gnu11。

WindowsMSVC cl.exe clang-cl.exe 。 clangMSVC。

visual studioPlatform toolset clang-cl。

clang。 clang-cl MSVC。

clangclang-clLinuxIDEMacXCode。 。

CMakeCCCXX

```
mkdir build
cd build
CC=clang CXX=clang++ cmake ..
cmake --build .
```

Cmake。

C++。

- Web。
- 。
- /。

。

- C++。
- 。

- C ++.
- C ++C ++C ++ 14C ++ 17.
- . . .
- .
  
- C ++. .
- . . . .
- .
  
- . .
  
- <http://codepad.org/> . .
- <http://coliru.stacked-crooked.com/> . GCCClang.
- <http://cpp.sh/> - C ++ 14. GUI.
- <https://gcc.godbolt.org/> - . . GCCClangMSVC CL icc ELLCCZapccARMARMv8ARM64 Atmel AVR MIPS MIPS64 MSP430 PowerPC x86x64 architectures. .
- <https://ideone.com/> - . GCCClang.
- <http://melpon.org/wandbox> - ClangGNU / GCC.
- <http://onlinegdb.com/> - IDEgccgdb.
- <http://rextester.com/> - CC ++ClangGCCVisual StudioBoost.
- [http://tutorialspoint.com/compile\\_cpp11\\_online.php](http://tutorialspoint.com/compile_cpp11_online.php) - GCCUNIX shell.
- <http://webcompiler.cloudapp.net/> - Visual Studio 2015 Microsoft RiSE4fun.

## C ++

C ++. CC ++. prog.cppC ++compile

```
g++ -Wall -ansi -o prog prog.cpp
```

4.

1. C ++C ++#include#define.

2. C ++C ++.

3. .

4.

.

#include#define. C ++.

C ++#include#define#if #ifdef#endif.

◦ ##.

. .

#if#error

◦

```
g++ -Wall -ansi -o prog prog.cpp
```

◦ C ++◦ ELFCOFFa.out...◦ ◦ ◦

◦ ◦ ◦

◦ ◦ ◦

◦

“”◦

-S

```
g++ -Wall -ansi -o prog prog.cpp
```

◦ UNIX.oMSDOS.OBJ◦ ◦ ◦

-C

```
g++ -Wall -ansi -o prog prog.cpp
```

◦ ◦

◦ ◦ ◦

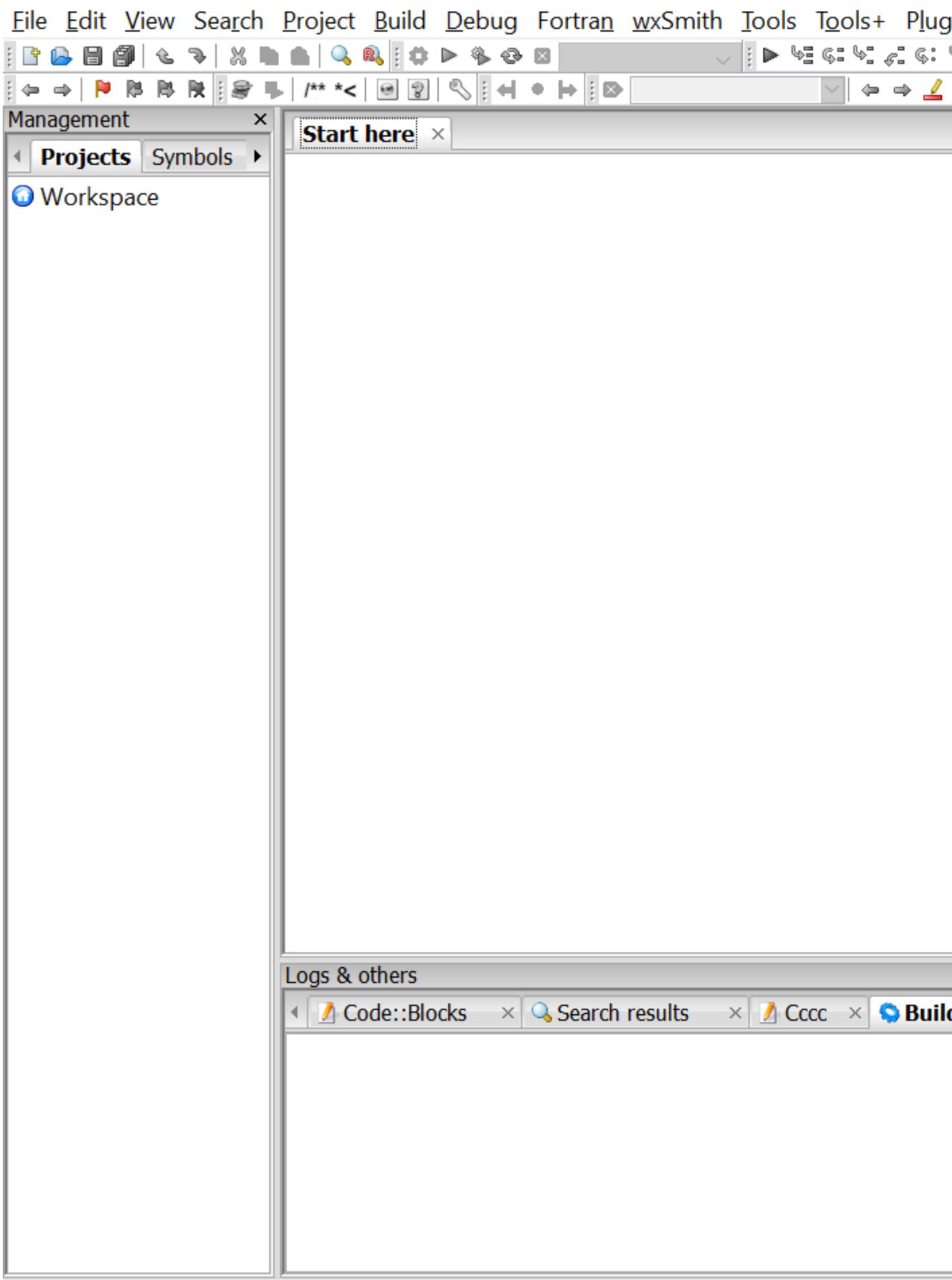
◦ ◦ ◦

## Code :: Blocks

1. Code :: Blocks◦ Windowsmingw◦

2. Code :: Blocks“”

# Start here - Code::Blocks 16.01



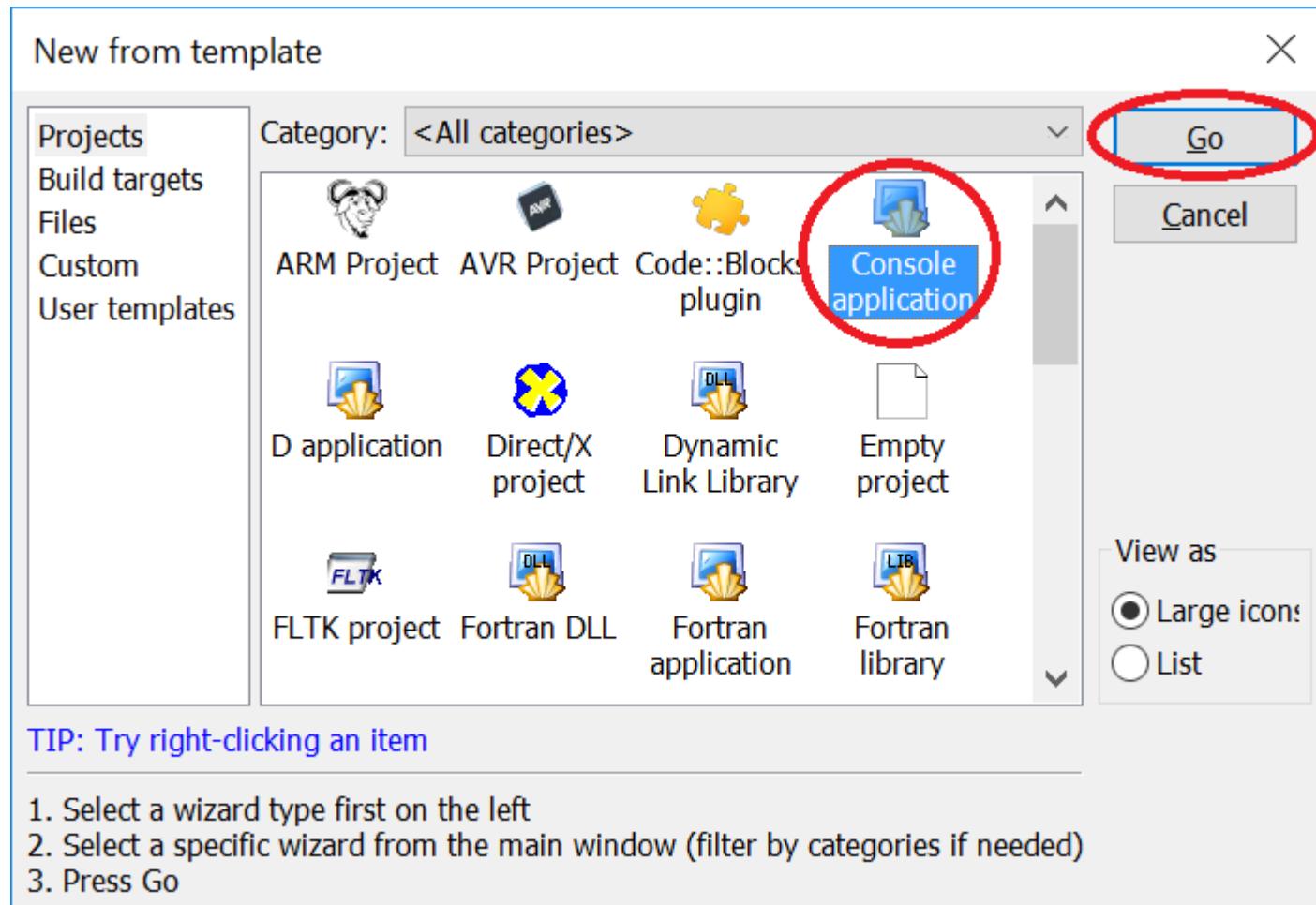
Start here

<https://riptutorial.com/zh-TW/home>



540

3. 66666



#### 4. “”“C ++”””””。

5. 。 “Hello world”。 //

# main.cpp [df] - Code::Blocks 16.01

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins



## Management

Projects    Symbols

Workspace  
df  
Sources  
main.cpp

## main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     cout << "Hello world!" << endl;
8     return 0;
9 }
10
```

## Logs & others

Code::Blocks    Search results    Cccccc    Build

C:\Users\glind\Desktop\df\main.cpp

<https://riptutorial.com/zh-TW/home>





“Hello world”

The screenshot shows a terminal window with a black background and white text. At the top left is a small icon of a terminal window. The text output is as follows:

```
Hello world!  
Process returned 0 (0x0) execution time : 0.047 s  
Press any key to continue.
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4708/>

# 118:

C。

- `extern string-literal { declaration-seq opt }`
- `extern string-literal`

```
extern "C" C++Cextern "C++" . . .
```

## Examples

Unix

CC。

```
volatile sig_atomic_t death_signal = 0;
extern "C" void cleanup(int signum) {
    death_signal = signum;
}
int main() {
    bind(...);
    listen(...);
    signal(SIGTERM, cleanup);
    while (int fd = accept(...)) {
        if (fd == -1 && errno == EINTR && death_signal) {
            printf("Caught signal %d; shutting down\n", death_signal);
            break;
        }
        // ...
    }
}
```

CC ++

ACC ++CC ++。 foo.h

```
typedef struct Foo {
    int bar;
} Foo;
Foo make_foo(int);
```

make\_foo

C ++#include <foo.h> make\_fooC。 make\_fooCC ++C ++make\_fooC ++make\_fooC。

extern "C"。

```
typedef struct Foo {
    int bar;
```

```
} Foo;  
Foo make_foo(int);
```

foo.h**CC ++**foo.h make\_fooextern "C"**unmangledC**。

<https://riptutorial.com/zh-TW/cplusplus/topic/9268/>

# 119:

- virtual void f;
- virtual void g= 0;
- // C ++ 11
  - virtual void h;
  - void i;
  - virtual void jfinal;
  - void kfinal;
- virtual .
- C ++ 11override .
- 。 [expr.delete]§5.3.5/ 3 。

## Examples

### C ++ 11override with virtual

overrideC ++ 11.

- 
- Basevirtual

run time

◦

override

```
#include <iostream>

struct X {
    virtual void f() { std::cout << "X::f()\n"; }
};

struct Y : X {
    // Y::f() will not override X::f() because it has a different signature,
    // but the compiler will accept the code (and silently ignore Y::f()).
    virtual void f(int a) { std::cout << a << "\n"; }
};
```

override

```
#include <iostream>

struct X {
    virtual void f() { std::cout << "X::f()\n"; }
```

```
};

struct Y : X {
    // Y::f() will not override X::f() because it has a different signature,
    // but the compiler will accept the code (and silently ignore Y::f()).
    virtual void f(int a) { std::cout << a << "\n"; }
};
```

override<sup>o</sup> override

```
#include <iostream>

struct X {
    virtual void f() { std::cout << "X::f()\n"; }
};

struct Y : X {
    // Y::f() will not override X::f() because it has a different signature,
    // but the compiler will accept the code (and silently ignore Y::f()).
    virtual void f(int a) { std::cout << a << "\n"; }
};
```

```
#include <iostream>

struct X {
    virtual void f() { std::cout << "X::f()\n"; }
};

struct Y : X {
    // Specifying virtual again here is optional
    // because it can be inferred from X::f().
    virtual void f() { std::cout << "Y::f()\n"; }
};

void call(X& a) {
    a.f();
}

int main() {
    X x;
    Y y;
    call(x); // outputs "X::f()"
    call(y); // outputs "Y::f()"
}
```

```
#include <iostream>

struct X {
    virtual void f() { std::cout << "X::f()\n"; }
};

struct Y : X {
    // Specifying virtual again here is optional
    // because it can be inferred from X::f().
    virtual void f() { std::cout << "Y::f()\n"; }
};
```

```

void call(X& a) {
    a.f();
}

int main() {
    X x;
    Y y;
    call(x); // outputs "X::f()"
    call(y); // outputs "Y::f()"
}

```

## C++ 11<sub>final</sub>

```

class Base {
public:
    virtual void foo() {
        std::cout << "Base::Foo\n";
    }
};

class Derived1 : public Base {
public:
    // Overriding Base::foo
    void foo() final {
        std::cout << "Derived1::Foo\n";
    }
};

class Derived2 : public Derived1 {
public:
    // Compilation error: cannot override final method
    virtual void foo() {
        std::cout << "Derived2::Foo\n";
    }
};

```

`final`virtual'`

`final"virtual"`

`overridefinal`

```

class Base {
public:
    virtual void foo() {
        std::cout << "Base::Foo\n";
    }
};

class Derived1 : public Base {
public:
    // Overriding Base::foo
    void foo() final {
        std::cout << "Derived1::Foo\n";
    }
};

```

```

class Derived2 : public Derived1 {
public:
    // Compilation error: cannot override final method
    virtual void foo() {
        std::cout << "Derived2::foo\n";
    }
};

```

◦

```

#include <iostream>
using namespace std;

class base {
public:
    base() { f("base constructor"); }
    ~base() { f("base destructor"); }

    virtual const char* v() { return "base::v()"; }

    void f(const char* caller) {
        cout << "When called from " << caller << ", " << v() << " gets called.\n";
    }
};

class derived : public base {
public:
    derived() { f("derived constructor"); }
    ~derived() { f("derived destructor"); }

    const char* v() override { return "derived::v()"; }

};

int main() {
    derived d;
}

```

**base :: v.**  
**derived :: v.**  
**derived :: v.**  
**basebase :: v.**

◦ **C ++ \*this**◦

```

#include <iostream>
using namespace std;

class base {
public:
    base() { f("base constructor"); }
    ~base() { f("base destructor"); }

    virtual const char* v() { return "base::v()"; }

    void f(const char* caller) {

```

```

        cout << "When called from " << caller << ", " << v() << " gets called.\n";
    }
};

class derived : public base {
public:
    derived() { f("derived constructor"); }
    ~derived() { f("derived destructor"); }

    const char* v() override { return "derived::v()"; }
};

int main() {
    derived d;
}

```

= 0virtual。 ;°

```

struct Abstract {
    virtual void f() = 0;
};

struct Concrete {
    void f() override {}
};

Abstract a; // Error.
Concrete c; // Good.

```

◦ ◦ ◦

```

struct Abstract {
    virtual void f() = 0;
};

struct Concrete {
    void f() override {}
};

Abstract a; // Error.
Concrete c; // Good.

```

● ◦ ◦ virtual◦ ◦

```

struct Abstract {
    virtual void f() = 0;
};

struct Concrete {
    void f() override {}
};

Abstract a; // Error.
Concrete c; // Good.

```

●

◦

```
struct Abstract {
    virtual void f() = 0;
};

struct Concrete {
    void f() override {}
};

Abstract a; // Error.
Concrete c; // Good.
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1752/>

# 120:

## Examples

---

---

**ET** . .

. . **ETET**.

◦

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

Vector+\*.

**ET** Vectorresult

1. vec\_1 vec\_2vec\_3◦
2. Vector\_tmp \_tmp = vec\_2\*vec\_3; \_tmp = vec\_2\*vec\_3; ◦
3. result = vec\_1 + \_tmp; result result = vec\_1 + \_tmp; ◦

**ET 2** Vector \_tmp Vector◦

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

Vector vec\_1, vec\_2, vec\_3result ◦ . ◦

---

**ET**

---

**ET**

1. **PAE** /◦ /◦ **PAE**◦ **PAE**◦
2. ◦ /◦

**ET**◦

```
Vector vec_1, vec_2, vec_3;
```

```
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

1. **plus\_expr**
2. **innerprod\_expr** .

## ET

- **ET** . **PAE** . .

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

- **result** **result** -**result** . **result****elem\_res****vec\_1 elem\_1 vec\_2 elem\_2 vec\_3 elem\_3**

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

Vector .

---

.

---

## vec.hhstd :: vector .

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

## File expr.hhvector plusvector inner product

.

1. 1. **CRT** .
2. **2PAE const** .
3. **3PAE binary\_operation** **vector\_plusvector\_innerprod** . **PAEP****PAE** . .
4. **4vector\_plusvector\_innerprod** . **PAE** **soperator** **+\*PAE** .

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

main.ccsrc

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

GCC 5.3-03 -std=c++14

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

- **ET** > 3x<sub>0</sub>
  - Vector **ETsector**
  - Boost::demangle**ET**

- **ET** ° °
  - **ET<sub>auto</sub>** ° **PAE<sub>auto</sub>** °

```
Vector vec_1, vec_2, vec_3;  
  
// Initializing vec_1, vec_2 and vec_3.  
  
Vector result = vec_1 + vec_2*vec_3;
```

for for.

ET

- **boost :: protoET**。
  - **EigenET**。

• `ArrayMatrix`。 Bjarne Stroustrup“C ++”。

## ◦ Matrix

```
template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

    for (size_t y = 0; y != lhs.rows(); ++y) {
        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}
```

## Matrix

```
template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

    for (size_t y = 0; y != lhs.rows(); ++y) {
```

```

        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}

```

`operator+()。`

`”。`

`Matrix d = a + b + c = ((a + b) + c)operator+(operator+(a, b), c) = operator+(). 2. Matrix。`

```

template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

    for (size_t y = 0; y != lhs.rows(); ++y) {
        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}

```

`MatrixMatrix::add2() Matrix::AddMultiply()。`

`Matrix d = a + b + c = a + b + c d。`

`operator+() Matrix”。`

**2**

```

template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

```

```

Matrix() : values(COL * ROW) {}

static size_t cols() { return COL; }
static size_t rows() { return ROW; }

const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

    for (size_t y = 0; y != lhs.rows(); ++y) {
        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}

```

operator+()

```

template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

    for (size_t y = 0; y != lhs.rows(); ++y) {
        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}

```

```

operator+() 2MatrixMatrix""。 。 。

MatrixSum<>

template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

    for (size_t y = 0; y != lhs.rows(); ++y) {
        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}

```

d = a + b

```

template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

```

```

        for (size_t y = 0; y != lhs.rows(); ++y) {
            for (size_t x = 0; x != lhs.cols(); ++x) {
                result(x, y) = lhs(x, y) + rhs(x, y);
            }
        }
        return result;
    }
}

```

abd °

◦ a + b + c

```

template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

```

```

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

```

```

    for (size_t y = 0; y != lhs.rows(); ++y) {
        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}

```

```

template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:

```

```

        std::vector<T> values;
    };

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

    for (size_t y = 0; y != lhs.rows(); ++y) {
        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}

```

Matrix<sup>o</sup> Matrix::operator=() “”

```

template <typename T, std::size_t COL, std::size_t ROW>
class Matrix {
public:
    using value_type = T;

    Matrix() : values(COL * ROW) {}

    static size_t cols() { return COL; }
    static size_t rows() { return ROW; }

    const T& operator()(size_t x, size_t y) const { return values[y * COL + x]; }
    T& operator()(size_t x, size_t y) { return values[y * COL + x]; }

private:
    std::vector<T> values;
};

template <typename T, std::size_t COL, std::size_t ROW>
Matrix<T, COL, ROW>
operator+(const Matrix<T, COL, ROW>& lhs, const Matrix<T, COL, ROW>& rhs)
{
    Matrix<T, COL, ROW> result;

    for (size_t y = 0; y != lhs.rows(); ++y) {
        for (size_t x = 0; x != lhs.cols(); ++x) {
            result(x, y) = lhs(x, y) + rhs(x, y);
        }
    }
    return result;
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/3404/>

# 121: Elision

## Examples

◦ ◦

```
std::string get_string()
{
    return std::string("I am a string.");
}
```

std::string /◦ ◦

C ++/◦ ◦

1. /◦ //◦

2. elision/◦

C ++ 11

```
std::string get_string()
{
    return std::string("I am a string.");
}
```

func “” my\_type◦ “”

“”◦ ;◦

/◦

elision elision◦ ◦ ◦

C ++ 17

elision◦ elision◦ //◦

```
std::mutex a_mutex;
std::lock_guard<std::mutex> get_lock()
{
    return std::lock_guard<std::mutex>(a_mutex);
}
```

a\_mutex◦

std::lock\_guard◦ C ++/◦

C ++ 17◦

C ++ 17/. 。

pre-C ++ 17/. C ++ 17.

C ++ 17 prvalue。 。 prvalue/。 C ++ 17.

prvalueC ++ 17。 get\_lock /

```
std::mutex a_mutex;
std::lock_guard<std::mutex> get_lock()
{
    return std::lock_guard<std::mutex>(a_mutex);
}
```

get\_lock prvalue。 ;the\_lock 。 /。

“”C ++ 17 。 ;/C ++/。

prvalue。

```
std::mutex a_mutex;
std::lock_guard<std::mutex> get_lock()
{
    return std::lock_guard<std::mutex>(a_mutex);
}
```

C ++ 17/. /。 lock\_guard。 。 ABI。

```
std::mutex a_mutex;
std::lock_guard<std::mutex> get_lock()
{
    return std::lock_guard<std::mutex>(a_mutex);
}
```

prvalue prvalueprvalue

```
std::string func()
{
    return std::string("foo");
}
```

。

prvalue prvalue。

```
void func(std::string str) { ... }

func(std::string("foo"));
```

string str。 elisionstr+。

```
explicit func("foo") stringconst char*string explicit explicit /.
```

- return
- 
- 

/

```
std::string func()  
{  
    std::string str("foo");  
    //Do stuff  
    return str;  
}
```

## elision

```
std::string func()  
{  
    std::string str("foo");  
    //Do stuff  
    return str;  
}
```

Elid<sub>ret</sub>。

## elision。

```
std::string func()  
{  
    std::string str("foo");  
    //Do stuff  
    return str;  
}
```

## elision

prvalue prvalue。

```
std::string str = std::string("foo");
```

std::string str("foo");。

```
std::string str = std::string("foo");
```

copy elision std::string 2。 1。

Elision <https://riptutorial.com/zh-TW/cplusplus/topic/2489/elision>

# 122:

- MyClassconst MyClassother;
- MyClassMyClassother;
- MyClassvolatile const MyClassother;
- MyClassvolatile MyClassother;
- 
- MyClassoperator =const MyClassrhs;
- MyClassoperator =MyClassrhs;
- MyClassoperator =MyClass rhs;
- const MyClassoperator =const MyClassrhs;
- const MyClassoperator =MyClassrhs;
- const MyClassoperator =MyClass rhs;
- MyClass=const MyClassrhs;
- MyClass=MyClassrhs;
- MyClass=MyClass rhs;

RHS    . MyClass operator =MyClassrhs;

C ++

GeeksForGeeks

C ++

## Examples

o

```
// Assignment Operator
#include <iostream>
#include <string>

using std::cout;
using std::endl;

class Foo
{
public:
    Foo(int data)
    {
        this->data = data;
    }
    ~Foo() {};
    Foo& operator=(const Foo& rhs)
    {
        data = rhs.data;
    }
}
```

```

        return *this;
    }

    int data;
};

int main()
{
    Foo foo(2); //Foo(int data) called
    Foo foo2(42);
    foo = foo2; // Assignment Operator Called
    cout << foo.data << endl; //Prints 42
}

```

foo = foo2; operator= function. <http://cpp.sh/3qtbm>

## Assignment Constructor . . .

```

// Copy Constructor
#include <iostream>
#include <string>

using std::cout;
using std::endl;

class Foo
{
public:
    Foo(int data)
    {
        this->data = data;
    }
    ~Foo() {};
    Foo(const Foo& rhs)
    {
        data = rhs.data;
    }

    int data;
};

int main()
{
    Foo foo(2); //Foo(int data) called
    Foo foo2 = foo; // Copy Constructor called
    cout << foo2.data << endl;
}

```

foo2 = foo; main. foo2intfoo. <http://cpp.sh/5iu7>

. .

```

// Copy vs Assignment Constructor
#include <iostream>
#include <string>

using std::cout;

```

```

using std::endl;

class Foo
{
public:
    Foo(int data)
    {
        this->data = data;
    }
    ~Foo() {};
    Foo(const Foo& rhs)
    {
        data = rhs.data;
    }

    Foo& operator=(const Foo& rhs)
    {
        data = rhs.data;
        return *this;
    }

    int data;
};

int main()
{
    Foo foo(2); //Foo(int data) / Normal Constructor called
    Foo foo2 = foo; //Copy Constructor Called
    cout << foo2.data << endl;

    Foo foo3(42);
    foo3=foo; //Assignment Constructor Called
    cout << foo3.data << endl;
}

```

```

// Copy vs Assignment Constructor
#include <iostream>
#include <string>

using std::cout;
using std::endl;

class Foo
{
public:
    Foo(int data)
    {
        this->data = data;
    }
    ~Foo() {};
    Foo(const Foo& rhs)
    {
        data = rhs.data;
    }

    Foo& operator=(const Foo& rhs)
    {
        data = rhs.data;
        return *this;
    }
}

```

```
    int data;
};

int main()
{
    Foo foo(2); //Foo(int data) / Normal Constructor called
    Foo foo2 = foo; //Copy Constructor Called
    cout << foo2.data << endl;

    Foo foo3(42);
    foo3=foo; //Assignment Constructor Called
    cout << foo3.data << endl;
}
```

Foo foo2 = foo; Foo foo2 = foo; . . . **foo3** foo3=foo ;

<https://riptutorial.com/zh-TW/cplusplus/topic/7158/>

# 123:

## Examples

;const◦ constconst◦

```
const int x = 123;
x = 456;      // error
int& r = x; // error

struct S {
    void f();
    void g() const;
};

const S s;
s.f(); // error
s.g(); // OK
```

## decltype

### C++ 11

◦

- edecltype(e) e◦

```
int x = 42;
std::vector<decltype(x)> v(100, x); // v is a vector<int>
```

- edecltype(e)◦

```
int x = 42;
std::vector<decltype(x)> v(100, x); // v is a vector<int>
```

- decltype(e)e◦

- eT decltype(e) T&◦
- eT Xdecltype(e) T&&◦
- eT prvalue decltype(e) T

◦

```
int x = 42;
std::vector<decltype(x)> v(100, x); // v is a vector<int>
```

### C++ 14

decltype(auto) return decltype auto◦

```
int x = 42;
std::vector<decltype(x)> v(100, x); // v is a vector<int>
```

◦

- int signed signed int int◦
- char signed char char char ◦ signed char -127+127◦
- short long long long◦
- signed bool wchar\_t char16\_t char32\_t◦

```
signed char celsius_temperature;
std::cin >> celsius_temperature;
if (celsius_temperature < -35) {
    std::cout << "cold day, eh?\n";
}
```

◦

- int unsigned unsigned int◦
- unsigned char char char char ◦ 255◦
- unsigned short long long long ◦ bool wchar\_t char16\_t char32\_t◦

```
char invert_case_table[256] = { ..., 'a', 'b', 'c', ..., 'A', 'B', 'C', ... };
char invert_case(char c) {
    unsigned char index = c;
    return invert_case_table[index];
    // note: returning invert_case_table[c] directly does the
    // wrong thing on implementations where char is a signed type
}
```

:volatile◦ const volatile◦

memory\_mapped\_port volatile sizeof(int) 1◦ volatile sizeof(int)◦

```
extern volatile char memory_mapped_port;
void write_to_device(int x) {
    const char* p = reinterpret_cast<const char*>(&x);
    for (int i = 0; i < sizeof(int); i++) {
        memory_mapped_port = p[i];
    }
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7840/>

# 124:

CC ++。 C ++。 。

## Examples

RAlI。 。 RAlI。 。 std::auto\_ptr。

```
#include <memory>
#include <iostream>
using namespace std;

int main() {
{
    auto_ptr ap(new int(5)); // dynamic memory is the resource
    cout << *ap << endl; // prints 5
} // auto_ptr is destroyed, its resource is automatically freed
}
```

C ++ 11

```
std::auto_ptr

#include <memory>
#include <iostream>
using namespace std;

int main() {
{
    auto_ptr ap(new int(5)); // dynamic memory is the resource
    cout << *ap << endl; // prints 5
} // auto_ptr is destroyed, its resource is automatically freed
}
```

std::auto\_ptr。 auto\_ptrs。 。 std::shared\_ptr

```
#include <memory>
#include <iostream>
using namespace std;

int main() {
{
    auto_ptr ap(new int(5)); // dynamic memory is the resource
    cout << *ap << endl; // prints 5
} // auto_ptr is destroyed, its resource is automatically freed
}
```

。 。 。 1。 3.

|   |   |   |
|---|---|---|
|   | 1 | 2 |
| 1 | 1 |   |

|   | 1   | 2   |
|---|-----|-----|
| 2 |     | 1   |
| 3 | 112 |     |
| 4 |     | 112 |
| 5 | 2   |     |
| 6 |     | 2   |

23.21。。

mutman of **mut** ual **ex** clusion。 “”。“”。。。

C ++ 11

std::mutex C ++ 11。

```
#include <thread>
#include <mutex>
#include <iostream>
using namespace std;

void add_1(int& i, const mutex& m) { // function to be run in thread
    m.lock();
    i += 1;
    m.unlock();
}

int main() {
    int var = 1;
    mutex m;

    cout << var << endl; // prints 1

    thread t1(add_1, var, m); // create thread with arguments
    thread t2(add_1, var, m); // create another thread
    t1.join(); t2.join(); // wait for both threads to finish

    cout << var << endl; // prints 3
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/8336/>

# 125:

## Examples

1. .

```
class foo {  
    int x;  
public:  
    int get_x();  
    void set_x(int new_x);  
};
```

2. . . .

```
class foo {  
    int x;  
public:  
    int get_x();  
    void set_x(int new_x);  
};
```

3. .

```
class foo {  
    int x;  
public:  
    int get_x();  
    void set_x(int new_x);  
};
```

4. class class C ++.

```
class foo {  
    int x;  
public:  
    int get_x();  
    void set_x(int new_x);  
};
```

5. 23°

```
class foo {  
    int x;  
public:  
    int get_x();  
    void set_x(int new_x);  
};
```

C ++ 11

6.

◦

```
class foo {
    int x;
public:
    int get_x();
    void set_x(int new_x);
};
```

class

- struct**base**structpublicprivate ◦
- struct;class◦

1. ◦

```
enum Direction {
    UP,
    LEFT,
    DOWN,
    RIGHT
};
Direction d = UP;
```

## C++ 11

C++ 11 enumclassstruct◦ : TT◦

```
enum Direction {
    UP,
    LEFT,
    DOWN,
    RIGHT
};
Direction d = UP;
```

enumenum◦

```
enum Direction {
    UP,
    LEFT,
    DOWN,
    RIGHT
};
Direction d = UP;
```

2. ◦ ◦ ◦

```
enum Direction {
    UP,
    LEFT,
    DOWN,
    RIGHT
```

```
};  
Direction d = UP;
```

## C++ 11

3. . .

unscoped . .

.

```
enum Direction {  
    UP,  
    LEFT,  
    DOWN,  
    RIGHT  
};  
Direction d = UP;
```

1. .

```
// Example is from POSIX  
union sigval {  
    int     sival_int;  
    void   *sival_ptr;  
};
```

2. . . .

```
// Example is from POSIX  
union sigval {  
    int     sival_int;  
    void   *sival_ptr;  
};
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7838/>

# 126:

mT'mo Tm ° C ++ - °

C ++C ++out-arguments out-argument ° ° Barbara LiskovLiskovLSP °

°

virtualvirtual °

## Examples

### 1.

```
// 1. Base example not using language support for covariance, dynamic type checking.

class Top
{
public:
    virtual Top* clone() const = 0;
    virtual ~Top() = default;           // Necessary for `delete` via Top*.
};

class D : public Top
{

public:
    Top* clone() const override
    { return new D( *this ); }
};

class DD : public D
{
private:
    int answer_ = 42;

public:
    int answer() const
    { return answer_; }

    Top* clone() const override
    { return new DD( *this ); }
};

#include <assert.h>
#include <iostream>
#include <typeinfo>
using namespace std;

int main()
{
    cout << boolalpha;

    DD* p1 = new DD();
    Top* p2 = p1->clone();
```

```

bool const correct_dynamic_type = typeid( *p2 ) == typeid( DD );
cout << correct_dynamic_type << endl; // "true"

assert( correct_dynamic_type ); // This is essentially dynamic type checking. :(
auto p2_most_derived = static_cast<DD*>( p2 );
cout << p2_most_derived->answer() << endl; // "42"
delete p2;
delete p1;
}

```

## 2.◦

```
// 2. Covariant result version of the base example, static type checking.
```

```

class Top
{
public:
    virtual Top* clone() const = 0;
    virtual ~Top() = default; // Necessary for `delete` via Top*.
};

class D : public Top
{
public:
    D* /* ← Covariant return */ clone() const override
    { return new D( *this ); }
};

class DD : public D
{
private:
    int answer_ = 42;

public:
    int answer() const
    { return answer_; }

    DD* /* ← Covariant return */ clone() const override
    { return new DD( *this ); }
};

#include <iostream>
using namespace std;

int main()
{
    DD* p1 = new DD();
    DD* p2 = p1->clone();
    // Correct dynamic type DD for *p2 is guaranteed by the static type checking.

    cout << p2->answer() << endl; // "42"
    delete p2;
    delete p1;
}

```

## 3.◦

```

// 3. Covariant smart pointer result (automated cleanup).

#include <memory>
using std::unique_ptr;

template< class Type >
auto up( Type* p ) { return unique_ptr<Type>( p ); }

class Top
{
private:
    virtual Top* virtual_clone() const = 0;

public:
    unique_ptr<Top> clone() const
    { return up( virtual_clone() ); }

    virtual ~Top() = default;           // Necessary for `delete` via Top*.
};

class D : public Top
{
private:
    D* /* ← Covariant return */ virtual_clone() const override
    { return new D( *this ); }

public:
    unique_ptr<D> /* ← Apparent covariant return */ clone() const
    { return up( virtual_clone() ); }
};

class DD : public D
{
private:
    int answer_ = 42;

    DD* /* ← Covariant return */ virtual_clone() const override
    { return new DD( *this ); }

public:
    int answer() const
    { return answer_; }

    unique_ptr<DD> /* ← Apparent covariant return */ clone() const
    { return up( virtual_clone() ); }
};

#include <iostream>
using namespace std;

int main()
{
    auto p1 = unique_ptr<DD>(new DD());
    auto p2 = p1->clone();
    // Correct dynamic type DD for *p2 is guaranteed by the static type checking.

    cout << p2->answer() << endl;           // "42"
    // Cleanup is automated via unique_ptr.
}

```

<https://riptutorial.com/zh-TW/cplusplus/topic/5411/>

# 127:

## Examples

switch。

```
// print the numbers to a file, one per line
for (const int num : num_list) {
    errno = 0;
    fprintf(file, "%d\n", num);
    if (errno == ENOSPC) {
        fprintf(stderr, "no space left on device; output will be truncated\n");
        break;
    }
}
```

◦

```
int sum = 0;
for (int i = 0; i < N; i++) {
    int x;
    std::cin >> x;
    if (x < 0) continue;
    sum += x;
    // equivalent to: if (x >= 0) sum += x;
}
```

do-while。

```
// Gets the next non-whitespace character from standard input
char read_char() {
    char c;
    do {
        c = getchar();
    } while (isspace(c));
    return c;
}
```

forC ++ 11for。

```
// print 10 asterisks
for (int i = 0; i < 10; i++) {
    putchar('*');
}
```

while。

```
int i = 0;
// print 10 asterisks
while (i < 10) {
    putchar('*');
```

```
i++;
}

std::vector<int> primes = {2, 3, 5, 7, 11, 13};

for(auto prime : primes) {
    std::cout << prime << std::endl;
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7841/>

# 128:

## Examples

C

```
// This creates an array with 5 values.  
const int array[] = { 1, 2, 3, 4, 5 };  
  
#ifdef BEFORE_CPP11  
  
// You can use `sizeof` to determine how many elements are in an array.  
const int* first = array;  
const int* afterLast = first + sizeof(array) / sizeof(array[0]);  
  
// Then you can iterate over the array by incrementing a pointer until  
// it reaches past the end of our array.  
for (const int* i = first; i < afterLast; ++i) {  
    std::cout << *i << std::endl;  
}  
  
#else  
  
// With C++11, you can let the STL compute the start and end iterators:  
for (auto i = std::begin(array); i != std::end(array); ++i) {  
    std::cout << *i << std::endl;  
}  
  
#endif
```

15

1  
2  
3  
4

```
// This creates an array with 5 values.  
const int array[] = { 1, 2, 3, 4, 5 };  
  
#ifdef BEFORE_CPP11  
  
// You can use `sizeof` to determine how many elements are in an array.  
const int* first = array;  
const int* afterLast = first + sizeof(array) / sizeof(array[0]);  
  
// Then you can iterate over the array by incrementing a pointer until  
// it reaches past the end of our array.  
for (const int* i = first; i < afterLast; ++i) {  
    std::cout << *i << std::endl;  
}  
  
#else
```

```

// With C++11, you can let the STL compute the start and end iterators:
for (auto i = std::begin(array); i != std::end(array); ++i) {
    std::cout << *i << std::endl;
}

#endif

```

## 5. C.

```

// This creates an array with 5 values.
const int array[] = { 1, 2, 3, 4, 5 };

#ifndef BEFORE_CPP11

// You can use `sizeof` to determine how many elements are in an array.
const int* first = array;
const int* afterLast = first + sizeof(array) / sizeof(array[0]);

// Then you can iterate over the array by incrementing a pointer until
// it reaches past the end of our array.
for (const int* i = first; i < afterLast; ++i) {
    std::cout << *i << std::endl;
}

#else

// With C++11, you can let the STL compute the start and end iterators:
for (auto i = std::begin(array); i != std::end(array); ++i) {
    std::cout << *i << std::endl;
}

#endif

```

## ◦◦◦ C sizeof◦◦◦

```

// This creates an array with 5 values.
const int array[] = { 1, 2, 3, 4, 5 };

#ifndef BEFORE_CPP11

// You can use `sizeof` to determine how many elements are in an array.
const int* first = array;
const int* afterLast = first + sizeof(array) / sizeof(array[0]);

// Then you can iterate over the array by incrementing a pointer until
// it reaches past the end of our array.
for (const int* i = first; i < afterLast; ++i) {
    std::cout << *i << std::endl;
}

#else

// With C++11, you can let the STL compute the start and end iterators:
for (auto i = std::begin(array); i != std::end(array); ++i) {
    std::cout << *i << std::endl;
}

#endif

```

- `iafterLast` `iarray`◦

```
// This creates an array with 5 values.
const int array[] = { 1, 2, 3, 4, 5 };

#ifndef BEFORE_CPP11

// You can use `sizeof` to determine how many elements are in an array.
const int* first = array;
const int* afterLast = first + sizeof(array) / sizeof(array[0]);

// Then you can iterate over the array by incrementing a pointer until
// it reaches past the end of our array.
for (const int* i = first; i < afterLast; ++i) {
    std::cout << *i << std::endl;
}

#else

// With C++11, you can let the STL compute the start and end iterators:
for (auto i = std::begin(array); i != std::end(array); ++i) {
    std::cout << *i << std::endl;
}

#endif
```

- `dereference``*i`◦
- 

- ◦

A B C

A B C

- ◦ A ◦ `erase(A)` ◦
- 

A B C

- `end()`

A B C

## C ++ `begin()``end()` ◦ `firstlast`

A B C

A B C

`begin()``end()`

A B C

A B C

◦

- insert
- erase
- 

—

◦ ◦

A B C

C ++◦ ◦

—

◦ ◦

A B C

**std :: distance** firstsecond ◦

◦ a = b + 3; b = a; ++b; ++b; ++b; 3b = a; ++b; ++b; ++b; ◦ ◦

—

C ++◦ ◦ ◦

- ◦ ◦
- **Forward Iterators**◦
- ◦
- ◦
- **C ++ 17**◦

◦ random\_shuffle ◦

—

◦

A B C

A B C

◦

```
reverse_iterator。 base()。
```

```
rbegin() rend()
```

```
std::vector<int> v{1, 2, 3, 4, 5};  
for (std::vector<int>::reverse_iterator it = v.rbegin(); it != v.rend(); ++it)  
{  
    cout << *it;  
} // prints 54321
```

```
base()。 base()
```

```
std::vector<int> v{1, 2, 3, 4, 5};  
for (std::vector<int>::reverse_iterator it = v.rbegin(); it != v.rend(); ++it)  
{  
    cout << *it;  
} // prints 54321
```

```
std::vector<int> v{1, 2, 3, 4, 5};  
for (std::vector<int>::reverse_iterator it = v.rbegin(); it != v.rend(); ++it)  
{  
    cout << *it;  
} // prints 54321
```

```
beginiterator。
```

```
enditeratorrend。
```

```
vectorconst_iterator const beginendconst_iterator。 const_iterator const cbeginend。
```

```
#include <vector>  
#include <iostream>  
  
int main() {  
    std::vector<int> v = { 1, 2, 3, 4, 5 }; //intialize vector using an initializer_list  
  
    for (std::vector<int>::iterator it = v.begin(); it != v.end(); ++it) {  
        std::cout << *it << " ";  
    }  
  
    return 0;  
}
```

```
1 2 3 4 5
```

## Map Iterator

◦

```
mapconst_iterator constconst_iterator。 iterator。
```

```
// Create a map and insert some values
```

```

std::map<char,int> mymap;
mymap['b'] = 100;
mymap['a'] = 200;
mymap['c'] = 300;

// Iterate over all tuples
for (std::map<char,int>::iterator it = mymap.begin(); it != mymap.end(); ++it)
    std::cout << it->first << " => " << it->second << '\n';

```

a => 200  
b => 100  
c => 300

```

// Data stream. Any number of various whitespace characters will be OK.
std::istringstream istr("1\t2      3 4");
std::vector<int> v;

// Constructing stream iterators and copying data from stream into vector.
std::copy(
    // Iterator which will read stream data as integers.
    std::istream_iterator<int>(istr),
    // Default constructor produces end-of-stream iterator.
    std::istream_iterator<int>(),
    std::back_inserter(v));

// Print vector contents.
std::copy(v.begin(), v.end(),
    // Will print values to standard output as integers delimited by " -- ".
    std::ostream_iterator<int>(std::cout, " -- "));

```

1 -- 2 -- 3 -- 4 --。

”。

## C++

```

template<class T>
struct generator_iterator {
    using difference_type=std::ptrdiff_t;
    using value_type=T;
    using pointer=T*;
    using reference=T;
    using iterator_category=std::input_iterator_tag;
    std::optional<T> state;
    std::function< std::optional<T>() > operation;
    // we store the current element in "state" if we have one:
    T operator*() const {
        return *state;
    }
    // to advance, we invoke our operation. If it returns a nullopt
    // we have reached the end:
    generator_iterator& operator++() {
        state = operation();
        return *this;
    }
    generator_iterator operator++(int) {
        auto r = *this;

```

```
++(*this);
return r;
}
// generator iterators are only equal if they are both in the "end" state:
friend bool operator==( generator_iterator const& lhs, generator_iterator const& rhs ) {
    if (!lhs.state && !rhs.state) return true;
    return false;
}
friend bool operator!=( generator_iterator const& lhs, generator_iterator const& rhs ) {
    return !(lhs==rhs);
}
// We implicitly construct from a std::function with the right signature:
generator_iterator( std::function< std::optional<T>() > f ):operation(std::move(f))
{
    if (operation)
        state = operation();
}
// default all special member functions:
generator_iterator( generator_iterator && ) =default;
generator_iterator( generator_iterator const& ) =default;
generator_iterator& operator=( generator_iterator && ) =default;
generator_iterator& operator=( generator_iterator const& ) =default;
generator_iterator() =default;
};


```

◦

◦

std::function◦

<https://riptutorial.com/zh-TW/cplusplus/topic/473/>

# 129:

- *function\_name* [ *function\_args* ][ *function\_attributes* ] [ *function\_qualifiers* ] -> *trailing-return-type* [ *requires\_clause* ]

◦  
◦ ->◦

## Examples

```
class ClassWithAReallyLongName {  
public:  
    class Iterator { /* ... */ };  
    Iterator end();  
};
```

end

```
class ClassWithAReallyLongName {  
public:  
    class Iterator { /* ... */ };  
    Iterator end();  
};
```

end

```
class ClassWithAReallyLongName {  
public:  
    class Iterator { /* ... */ };  
    Iterator end();  
};
```

“”◦

## Lambda

lambda ;lambdas◦ lambda◦

```
struct Base {};  
struct Derived1 : Base {};  
struct Derived2 : Base {};  
auto lambda = [](bool b) -> Base* { if (b) return new Derived1; else return new Derived2; };  
// ill-formed: auto lambda = Base* [](bool b) { ... };
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4142/>

# 130:

thisC ++。 this

```
    this.someMember();
```

->

```
    this.someMember();
```

this

'this'

<http://www.geeksforgeeks.org/this-pointer-in-c/>

[https://www.tutorialspoint.com/cplusplus/cpp\\_this\\_pointer.htm](https://www.tutorialspoint.com/cplusplus/cpp_this_pointer.htm)

## Examples

this ;◦ this ;◦

```
struct ThisPointer {
    int i;

    ThisPointer(int ii);

    virtual void func();

    int get_i() const;
    void set_i(int ii);
};

ThisPointer::ThisPointer(int ii) : i(ii) {}
// Compiler rewrites as:
ThisPointer::ThisPointer(int ii) : this->i(ii) {}

// Constructor is responsible for turning allocated memory into 'this'.
// As the constructor is responsible for creating the object, 'this' will not be "fully"
// valid until the instance is fully constructed.

/* virtual */ void ThisPointer::func() {
    if (some_external_condition) {
        set_i(182);
    } else {
        i = 218;
    }
}
// Compiler rewrites as:
/* virtual */ void ThisPointer::func(ThisPointer* this) {
    if (some_external_condition) {
        this->set_i(182);
    } else {
        this->i = 218;
    }
}
```

```

int ThisPointer::get_i() const { return i; }
// Compiler rewrites as:
int ThisPointer::get_i(const ThisPointer* this) { return this->i; }

void ThisPointer::set_i(int ii) { i = ii; }
// Compiler rewrites as:
void ThisPointer::set_i(ThisPointer* this, int ii) { this->i = ii; }

```

this;◦ thisV◦

---

- ◦ ctd\_goodctd\_badCtorThisBaseCtorThisBase() CtorThisCtorThis() CtorThisDerived ◦ ◦

```

struct ThisPointer {
    int i;

    ThisPointer(int ii);

    virtual void func();

    int get_i() const;
    void set_i(int ii);
};

ThisPointer::ThisPointer(int ii) : i(ii) {}
// Compiler rewrites as:
ThisPointer::ThisPointer(int ii) : this->i(ii) {}
// Constructor is responsible for turning allocated memory into 'this'.
// As the constructor is responsible for creating the object, 'this' will not be "fully"
// valid until the instance is fully constructed.

/* virtual */ void ThisPointer::func() {
    if (some_external_condition) {
        set_i(182);
    } else {
        i = 218;
    }
}
// Compiler rewrites as:
/* virtual */ void ThisPointer::func(ThisPointer* this) {
    if (some_external_condition) {
        this->set_i(182);
    } else {
        this->i = 218;
    }
}

int ThisPointer::get_i() const { return i; }
// Compiler rewrites as:
int ThisPointer::get_i(const ThisPointer* this) { return this->i; }

void ThisPointer::set_i(int ii) { i = ii; }
// Compiler rewrites as:
void ThisPointer::set_i(ThisPointer* this, int ii) { this->i = ii; }

```

- ctd\_good
  - CtorThisBaseCtorThis◦ si◦
  - ij(this->i)◦ ij◦

- jk(j)◦ jk◦
- ctd\_bad
  - kj(this->k)◦ kj◦
  - CtorThisDerivedCtorThisCtorThis◦ bk◦
  - ctd\_badCtorThis()CtorThis CtorThisDerived()CtorThisDerived**vtable**◦ virt\_func()
   
CtorThis::virt\_func() CtorThisDerived::virt\_func()◦

this◦

```
// Example for this pointer
#include <iostream>
#include <string>

using std::cout;
using std::endl;

class Class
{
public:
    Class();
    ~Class();
    int getPrivateNumber () const;
private:
    int private_number = 42;
};

Class::Class(){}
Class::~Class(){}

int Class::getPrivateNumber() const
{
    return this->private_number;
}

int main()
{
    Class class_example;
    cout << class_example.getPrivateNumber() << endl;
}
```

◦

.....

```
// Dog Class Example
#include <iostream>
#include <string>

using std::cout;
using std::endl;

/*
 * @class Dog
 *   @member name
 *     Dog's name
 *   @function bark
```

```

*      Dog Barks!
*  @function getName
*      To Get Private
*      Name Variable
*/
class Dog
{
public:
    Dog(std::string name);
    ~Dog();
    void bark() const;
    std::string getName() const;
private:
    std::string name;
};

Dog::Dog(std::string name)
{
    /*
     *  this->name is the
     *  name variable from
     *  the class dog . and
     *  name is from the
     *  parameter of the function
     */
    this->name = name;
}

Dog::~Dog() {}

void Dog::bark() const
{
    cout << "BARK" << endl;
}

std::string Dog::getName() const
{
    return this->name;
}

int main()
{
    Dog dog("Max");
    cout << dog.getName() << endl;
    dog.bark();
}

```

```

// Dog Class Example
#include <iostream>
#include <string>

using std::cout;
using std::endl;

/*
 * @class Dog
 *  @member name
 *      Dog's name
 *  @function bark

```

```

*      Dog Barks!
*      @function getName
*      To Get Private
*      Name Variable
*/
class Dog
{
public:
    Dog(std::string name);
    ~Dog();
    void bark() const;
    std::string getName() const;
private:
    std::string name;
};

Dog::Dog(std::string name)
{
    /*
     *  this->name is the
     *  name variable from
     *  the class dog . and
     *  name is from the
     *  parameter of the function
     */
    this->name = name;
}

Dog::~Dog() {}

void Dog::bark() const
{
    cout << "BARK" << endl;
}

std::string Dog::getName() const
{
    return this->name;
}

int main()
{
    Dog dog("Max");
    cout << dog.getName() << endl;
    dog.bark();
}

```

Dogthis-> name。

<http://cpp.sh/75r7>

## Pointer CV-Qualifiers

this**CV**◦ this; **CV**◦

```

struct ThisCVQ {
    void no_qualifier()           {} // "this" is: ThisCVQ*

```

```
void c_qualifier() const {} // "this" is: const ThisCVQ*
void v_qualifier() volatile {} // "this" is: volatile ThisCVQ*
void cv_qualifier() const volatile {} // "this" is: const volatile ThisCVQ*
};
```

this~~this~~ CV-。

```
struct ThisCVQ {
    void no_qualifier() {} // "this" is: ThisCVQ*
    void c_qualifier() const {} // "this" is: const ThisCVQ*
    void v_qualifier() volatile {} // "this" is: volatile ThisCVQ*
    void cv_qualifier() const volatile {} // "this" is: const volatile ThisCVQ*
};
```

---

this~~const~~ const volatile ° ~~mutable~~ ° const °

° ° °

° 10°

C++const°

```
struct ThisCVQ {
    void no_qualifier() {} // "this" is: ThisCVQ*
    void c_qualifier() const {} // "this" is: const ThisCVQ*
    void v_qualifier() volatile {} // "this" is: volatile ThisCVQ*
    void cv_qualifier() const volatile {} // "this" is: const volatile ThisCVQ*
};
```

const\_cast~~this~~ -CV- ~~mutable~~ ° const\_cast const~~mutable~~ ° °

~~constCV;CVconst UB~~

```
struct ThisCVQ {
    void no_qualifier() {} // "this" is: ThisCVQ*
    void c_qualifier() const {} // "this" is: const ThisCVQ*
    void v_qualifier() volatile {} // "this" is: volatile ThisCVQ*
    void cv_qualifier() const volatile {} // "this" is: const volatile ThisCVQ*
};
```

°

---

thisvolatile const volatile ° volatile °

---

CV~~this~~CV

- CV°
- constconstconst volatile°
- volatilevolatileconst volatile°

- const volatileconst volatile◦

const◦

```
struct ThisCVQ {
    void no_qualifier() {} // "this" is: ThisCVQ*
    void c_qualifier() const {} // "this" is: const ThisCVQ*
    void v_qualifier() volatile {} // "this" is: volatile ThisCVQ*
    void cv_qualifier() const volatile {} // "this" is: const volatile ThisCVQ*
};
```

## C++ 11

this cvref-qualifiers\*this◦ Ref-qualifiersnormalvalue\*thisthis◦

ref-qualifiers this◦ ref-qualifiers\*this◦

```
struct RefQualifiers {
    std::string s;

    RefQualifiers(const std::string& ss = "The nameless one.") : s(ss) {}

    // Normal version.
    void func() & { std::cout << "Accessed on normal instance " << s << std::endl; }
    // Rvalue version.
    void func() && { std::cout << "Accessed on temporary instance " << s << std::endl; }

    const std::string& still_a_pointer() & { return this->s; }
    const std::string& still_a_pointer() && { this->s = "Bob"; return this->s; }
};

// ...

RefQualifiers rf("Fred");
rf.func(); // Output: Accessed on normal instance Fred
RefQualifiers{}.func(); // Output: Accessed on temporary instance The nameless one
```

ref-qualifiers;◦ cv-qualifiersref-qualifiersconst◦

```
struct RefQualifiers {
    std::string s;

    RefQualifiers(const std::string& ss = "The nameless one.") : s(ss) {}

    // Normal version.
    void func() & { std::cout << "Accessed on normal instance " << s << std::endl; }
    // Rvalue version.
    void func() && { std::cout << "Accessed on temporary instance " << s << std::endl; }

    const std::string& still_a_pointer() & { return this->s; }
    const std::string& still_a_pointer() && { this->s = "Bob"; return this->s; }
};

// ...

RefQualifiers rf("Fred");
```

```
rf.func();           // Output: Accessed on normal instance Fred  
RefQualifiers{}.func(); // Output: Accessed on temporary instance The nameless one
```

<https://riptutorial.com/zh-TW/cplusplus/topic/7146/>

# 131:

◦ ◦

1. ::
2. [] () T(...) . -> ++ -- dynamic\_cast static\_cast reinterpret\_cast const\_cast typeid
3. ++ -- \* & + - ! ~ sizeof new delete delete[] ; C (T)... ; C ++ 11 sizeof... alignof noexcept
4. .\*->\*
5. \* /%
6. +-
7. <<>>
8. < > <= >=
9. ==!=
10. & AND
11. ^
12. |
13. &&
14. ||
15. ?:
16. = \*= /= %= += -= >>= <<= &= ^= |=
17. throw
18. ,

◦ ◦

◦

- ?:◦ ◦ a ? b , c : d◦
- ?throw a = b ? c : d a = (b ? c : d)throw a ? b : c throw (a ? b : c) ◦
- :a ? b : c = d a ? b : (c = d) ◦

## Examples

### C ++

◦

( )◦ ◦

```
// volume of a spherical shell = 4 pi R^3 - 4 pi r^3
double vol = 4.0*pi*R*R*R/3.0 - 4.0*pi*r*r*r/3.0;

//Addition:

int a = 2+4/2;           // equal to: 2+(4/2)           result: 4
int b = (3+3)/2;         // equal to: (3+3)/2          result: 3

//With Multiplication

int c = 3+4/2*6;         // equal to: 3+((4/2)*6)      result: 15
```

```

int d = 3*(3+6)/9;           // equal to: (3*(3+6))/9      result: 3

//Division and Modulo

int g = 3-3%1;              // equal to: 3 % 1 = 0   3 - 0 = 3
int h = 3-(3%1);            // equal to: 3 % 1 = 0   3 - 0 = 3
int i = 3-3/1%3;            // equal to: 3 / 1 = 3   3 % 3 = 0   3 - 0 = 3
int l = 3-(3/1)%3;          // equal to: 3 / 1 = 3   3 % 3 = 0   3 - 0 = 3
int m = 3-(3/(1%3));       // equal to: 1 % 3 = 1   3 / 1 = 3   3 - 3 = 0

```

## ANDOR

### C ++ORAND◦

```

// You can drive with a foreign license for up to 60 days
bool can_drive = has_domestic_license || has_foreign_license && num_days <= 60;

```

```

// You can drive with a foreign license for up to 60 days
bool can_drive = has_domestic_license || has_foreign_license && num_days <= 60;

```

◦ ◦

### &&||◦

### &&||◦◦

### C ++&&||◦◦

||true◦◦

```

#include <iostream>
#include <string>

using namespace std;

bool True(string id){
    cout << "True" << id << endl;
    return true;
}

bool False(string id){
    cout << "False" << id << endl;
    return false;
}

int main(){
    bool result;
    //let's evaluate 3 booleans with || and && to illustrate operator precedence
    //precedence does not mean that && will be evaluated first but rather where
    //parentheses would be added
    //example 1
    result =
        False("A") || False("B") && False("C");
        // eq. False("A") || (False("B") && False("C"))

```

```

//FalseA
//FalseB
//"Short-circuit evaluation skip of C"
//A is false so we have to evaluate the right of ||,
//B being false we do not have to evaluate C to know that the result is false

result =
    True("A") || False("B") && False("C");
    // eq. True("A") || (False("B") && False("C"))
cout << result << " :======" << endl;
//TrueA
//"Short-circuit evaluation skip of B"
//"Short-circuit evaluation skip of C"
//A is true so we do not have to evaluate
//      the right of || to know that the result is true
//If || had precedence over && the equivalent evaluation would be:
//(True("A") || False("B")) && False("C")
//What would print
//TrueA
//"Short-circuit evaluation skip of B"
//FalseC
//Because the parentheses are placed differently
//the parts that get evaluated are differently
//which makes that the end result in this case would be False because C is false
}

```

◦

## postfix

```

int a = 1;
++a;           // result: 2
a--;           // result: 1
int minusa=-a; // result: -1

bool b = true;
!b; // result: true

a=4;
int c = a++/2; // equal to: (a==4) 4 / 2   result: 2 ('a' incremented postfix)
cout << a << endl; // prints 5!
int d = ++a/2;    // equal to: (a+1) == 6 / 2 result: 3

int arr[4] = {1,2,3,4};

int *ptr1 = &arr[0]; // points to arr[0] which is 1
int *ptr2 = ptr1++; // ptr2 points to arr[0] which is still 1; ptr1 incremented
std::cout << *ptr1++ << std::endl; // prints 2

int e = arr[0]++; // receives the value of arr[0] before it is incremented
std::cout << e << std::endl; // prints 1
std::cout << *ptr2 << std::endl; // prints arr[0] which is now 2

```

<https://riptutorial.com/zh-TW/cplusplus/topic/3895/>

# 132:

C ++ -> ◦ <string>+◦ operator ◦

◦ ◦

- “”◦
- .\*◦
- ::◦
- ?:◦
- dynamic\_cast static\_cast reinterpret\_cast const\_cast typeid sizeof alignof noexcept
- #####◦

99.98

- && | | bool
- ,◦
- &

◦

&& | | C ++ 17 ,◦

## Examples

- +=
- -=
- \*\*=
- //=
- &&=
- ||=
- ^^=
- >>>>=
- <<<<=

◦

class / struct

```
//operator+ should be implemented in terms of operator+=
T operator+(T lhs, const T& rhs)
{
    lhs += rhs;
    return lhs;
}

T& operator+=(T& lhs, const T& rhs)
{
    //Perform addition
    return lhs;
}
```

```
class / struct

//operator+ should be implemented in terms of operator+=
T operator+(T lhs, const T& rhs)
{
    lhs += rhs;
    return lhs;
}

T& operator+=(T& lhs, const T& rhs)
{
    //Perform addition
    return lhs;
}
```

---

operator+**const**const**const**。

1. Object foobar = foo + bar;foo
2. const operator+operator+=

const&◦ ◦

---

operator+=◦

const◦ const&**const**◦

2

- ++foofoo++
- --foofoo--

++--◦

class / struct

```
//Prefix operator ++foo
T& operator++(T& lhs)
{
    //Perform addition
    return lhs;
}

//Postfix operator foo++ (int argument is used to separate pre- and postfix)
//Should be implemented in terms of ++foo (prefix operator)
T operator++(T& lhs, int)
{
    T t(lhs);
    ++lhs;
    return t;
}
```

class / struct

```

//Prefix operator ++foo
T& operator++(T& lhs)
{
    //Perform addition
    return lhs;
}

//Postfix operator foo++ (int argument is used to separate pre- and postfix)
//Should be implemented in terms of ++foo (prefix operator)
T operator++(T& lhs, int)
{
    T t(lhs);
    ++lhs;
    return t;
}

```

◦ const◦

◦ const◦

**“”constconst◦**

**for++postfix ++◦ for++“”◦ for++**

```

//Prefix operator ++foo
T& operator++(T& lhs)
{
    //Perform addition
    return lhs;
}

//Postfix operator foo++ (int argument is used to separate pre- and postfix)
//Should be implemented in terms of ++foo (prefix operator)
T operator++(T& lhs, int)
{
    T t(lhs);
    ++lhs;
    return t;
}

```

- ==!=
- ><
- >=<=

**2 ==<◦**

**class / struct**

```

//Only implement those 2
bool operator==(const T& lhs, const T& rhs) { /* Compare */ }
bool operator<(const T& lhs, const T& rhs) { /* Compare */ }

//Now you can define the rest
bool operator!=(const T& lhs, const T& rhs) { return !(lhs == rhs); }

```

```
bool operator>(const T& lhs, const T& rhs) { return rhs < lhs; }
bool operator<=(const T& lhs, const T& rhs) { return !(lhs > rhs); }
bool operator>=(const T& lhs, const T& rhs) { return !(lhs < rhs); }
```

class / struct

```
//Only implement those 2
bool operator==(const T& lhs, const T& rhs) { /* Compare */ }
bool operator<(const T& lhs, const T& rhs) { /* Compare */ }

//Now you can define the rest
bool operator!=(const T& lhs, const T& rhs) { return !(lhs == rhs); }
bool operator>(const T& lhs, const T& rhs) { return rhs < lhs; }
bool operator<=(const T& lhs, const T& rhs) { return !(lhs > rhs); }
bool operator>=(const T& lhs, const T& rhs) { return !(lhs < rhs); }
```

---

bool truefalse。

const&。 & const **reference**。

class / struct const const const。

◦

class / struct

```
operator T() const { /* return something */ }
```

const const。

```
operator T() const { /* return something */ }
```

**operator** []。

**99.982** const const const []。

const& **value** const。

[]

```
std::vector<int> v{ 1 };
v[0] = 2; //Changes value of 1 to 2
           //wouldn't be possible if not returned by reference
```

---

class / struct

```
std::vector<int> v{ 1 };
v[0] = 2; //Changes value of 1 to 2
           //wouldn't be possible if not returned by reference
```

[ ] [ ] ... °

```
std::vector<int> v{ 1 };
v[0] = 2; //Changes value of 1 to 2
//wouldn't be possible if not returned by reference
```

## operator ()

class / struct

```
//R -> Return type
//Types -> any different type
R operator() (Type name, Type2 name2, ...)
{
    //Do something
    //return something
}

//Use it like this (R is return type, a and b are variables)
R foo = object(a, b, ...);
```

```
//R -> Return type
//Types -> any different type
R operator() (Type name, Type2 name2, ...)
{
    //Do something
    //return something
}

//Use it like this (R is return type, a and b are variables)
R foo = object(a, b, ...);
```

°

class / struct " " ° class / struct °

= °

1. °

- ;
- °

2. ° ° °

3. ° classstruct °

4. \*this ° int b = (a = 6) + 4; //b == 10 °

```
//T is some type
T& operator=(const T& other)
{
    //Do something (like copying values)
    return *this;
}
```

```
otherconst&operator=const °  
  
class / struct=class / struct° °  
  
class / structclass / struct°
```

## NOT

NOT ~ °

```
class / struct
```

```
T operator~(T lhs)  
{  
    //Do operation  
    return lhs;  
}
```

```
class / struct
```

```
T operator~(T lhs)  
{  
    //Do operation  
    return lhs;  
}
```

---

```
operator~° const int a = ~a + 1; °
```

```
class / structthis °
```

## I/O

<<>> “”””

- std::ostream<< std::cout
- std::istream>> std::cin

```
class / struct“””
```

- std::ostream std::cout << a << b; ° std::ostream&
- lhs
- rhs° const& rhs° const Vector& °

```
//Overload std::ostream operator<< to allow output from Vector's  
std::ostream& operator<<(std::ostream& lhs, const Vector& rhs)  
{  
    lhs << "x: " << rhs.x << " y: " << rhs.y << " z: " << rhs.z << '\n';  
    return lhs;  
}  
  
Vector v = { 1, 2, 3};
```

```
//Now you can do
std::cout << v;
```

+ - \*/complex<T>.

◦

```
#include <type_traits>

namespace not_std{

using std::decay_t;

//-----
// complex< value_t >
//-----

template<typename value_t>
struct complex
{
    value_t x;
    value_t y;

    complex &operator += (const value_t &x)
    {
        this->x += x;
        return *this;
    }
    complex &operator += (const complex &other)
    {
        this->x += other.x;
        this->y += other.y;
        return *this;
    }

    complex &operator -= (const value_t &x)
    {
        this->x -= x;
        return *this;
    }
    complex &operator -= (const complex &other)
    {
        this->x -= other.x;
        this->y -= other.y;
        return *this;
    }

    complex &operator *= (const value_t &s)
    {
        this->x *= s;
        this->y *= s;
        return *this;
    }
    complex &operator *= (const complex &other)
    {
        (*this) = (*this) * other;
        return *this;
    }
}
```

```

complex &operator /= (const value_t &s)
{
    this->x /= s;
    this->y /= s;
    return *this;
}
complex &operator /= (const complex &other)
{
    (*this) = (*this) / other;
    return *this;
}

complex(const value_t &x, const value_t &y)
: x{x}
, y{y}
{}

template<typename other_value_t>
explicit complex(const complex<other_value_t> &other)
: x{static_cast<const value_t &>(other.x)}
, y{static_cast<const value_t &>(other.y)}
{}

complex &operator = (const complex &) = default;
complex &operator = (complex &&) = default;
complex(const complex &) = default;
complex(complex &&) = default;
complex() = default;
};

// Absolute value squared
template<typename value_t>
value_t absqr(const complex<value_t> &z)
{ return z.x*z.x + z.y*z.y; }

//-----
// operator - (negation)
//-----

template<typename value_t>
complex<value_t> operator - (const complex<value_t> &z)
{ return {-z.x, -z.y}; }

//-----
// operator +
//-----

template<typename left_t,typename right_t>
auto operator + (const complex<left_t> &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a.x + b.x)>>
{ return{a.x + b.x, a.y + b.y}; }

template<typename left_t,typename right_t>
auto operator + (const left_t &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a + b.x)>>
{ return{a + b.x, b.y}; }

template<typename left_t,typename right_t>
auto operator + (const complex<left_t> &a, const right_t &b)
-> complex<decay_t<decltype(a.x + b)>>

```

```

{ return{a.x + b, a.y}; }

//-----
// operator -
//-----

template<typename left_t,typename right_t>
auto operator - (const complex<left_t> &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a.x - b.x)>>
{ return{a.x - b.x, a.y - b.y}; }

template<typename left_t,typename right_t>
auto operator - (const left_t &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a - b.x)>>
{ return{a - b.x, - b.y}; }

template<typename left_t,typename right_t>
auto operator - (const complex<left_t> &a, const right_t &b)
-> complex<decay_t<decltype(a.x - b)>>
{ return{a.x - b, a.y}; }

//-----
// operator *
//-----

template<typename left_t, typename right_t>
auto operator * (const complex<left_t> &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a.x * b.x)>>
{
    return {
        a.x*b.x - a.y*b.y,
        a.x*b.y + a.y*b.x
    };
}

template<typename left_t, typename right_t>
auto operator * (const left_t &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a * b.x)>>
{ return {a * b.x, a * b.y}; }

template<typename left_t, typename right_t>
auto operator * (const complex<left_t> &a, const right_t &b)
-> complex<decay_t<decltype(a.x * b)>>
{ return {a.x * b, a.y * b}; }

//-----
// operator /
//-----

template<typename left_t, typename right_t>
auto operator / (const complex<left_t> &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a.x / b.x)>>
{
    const auto r = absqr(b);
    return {
        ( a.x*b.x + a.y*b.y ) / r,
        (-a.x*b.y + a.y*b.x) / r
    };
}

template<typename left_t, typename right_t>
```

```

auto operator / (const left_t &a, const complex<right_t> &b)
-> complex<decay_t<decltype(a / b.x)>>
{
    const auto s = a/absqr(b);
    return {
        b.x * s,
        -b.y * s
    };
}

template<typename left_t, typename right_t>
auto operator / (const complex<left_t> &a, const right_t &b)
-> complex<decay_t<decltype(a.x / b)>>
{ return {a.x / b, a.y / b}; }

}// namespace not_std

int main(int argc, char **argv)
{
    using namespace not_std;

    complex<float> fz{4.0f, 1.0f};

    // makes a complex<double>
    auto dz = fz * 1.0;

    // still a complex<double>
    auto idz = 1.0f/dz;

    // also a complex<double>
    auto one = dz * idz;

    // a complex<double> again
    auto one_again = fz * idz;

    // Operator tests, just to make sure everything compiles.

    complex<float> a{1.0f, -2.0f};
    complex<double> b{3.0, -4.0};

    // All of these are complex<double>
    auto c0 = a + b;
    auto c1 = a - b;
    auto c2 = a * b;
    auto c3 = a / b;

    // All of these are complex<float>
    auto d0 = a + 1;
    auto d1 = 1 + a;
    auto d2 = a - 1;
    auto d3 = 1 - a;
    auto d4 = a * 1;
    auto d5 = 1 * a;
    auto d6 = a / 1;
    auto d7 = 1 / a;

    // All of these are complex<double>
    auto e0 = b + 1;
    auto e1 = 1 + b;
    auto e2 = b - 1;

```

```

    auto e3 = 1 - b;
    auto e4 = b * 1;
    auto e5 = 1 * b;
    auto e6 = b / 1;
    auto e7 = 1 / b;

    return 0;
}

```

## C ++"C ++。

```

namespace named_operator {
    template<class D>struct make_operator{constexpr make_operator(){}};

    template<class T, char, class O> struct half_apply { T&& lhs; };

    template<class Lhs, class Op>
    half_apply<Lhs, '*', Op> operator*( Lhs&& lhs, make_operator<Op> ) {
        return {std::forward<Lhs>(lhs)};
    }

    template<class Lhs, class Op, class Rhs>
    auto operator*( half_apply<Lhs, '*', Op>&& lhs, Rhs&& rhs )
-> decltype( named_invoke( std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs) ) )
{
        return named_invoke( std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs) );
    }
}

```

◦

```

namespace named_operator {
    template<class D>struct make_operator{constexpr make_operator(){}};

    template<class T, char, class O> struct half_apply { T&& lhs; };

    template<class Lhs, class Op>
    half_apply<Lhs, '*', Op> operator*( Lhs&& lhs, make_operator<Op> ) {
        return {std::forward<Lhs>(lhs)};
    }

    template<class Lhs, class Op, class Rhs>
    auto operator*( half_apply<Lhs, '*', Op>&& lhs, Rhs&& rhs )
-> decltype( named_invoke( std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs) ) )
{
        return named_invoke( std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs) );
    }
}

```

append\_t::named\_operator::make\_operator<append\_t>append  
append\_t::named\_operator::make\_operator<append\_t>。

## named\_invoke lhs append\_trhs。

```

lhs*append_t half_apply◦ half_apply*rhsnamed_invoke( lhs, append_t, rhs ) ◦

append_t

```

**ADL**`named_invoke`。

## std :: array

```
namespace named_operator {
    template<class D>struct make_operator{constexpr make_operator(){}};

    template<class T, char, class O> struct half_apply { T&& lhs; };

    template<class Lhs, class Op>
    half_apply<Lhs, '*', Op> operator*( Lhs&& lhs, make_operator<Op> ) {
        return {std::forward<Lhs>(lhs)};
    }

    template<class Lhs, class Op, class Rhs>
    auto operator*( half_apply<Lhs, '*', Op>&& lhs, Rhs&& rhs )
    -> decltype( named_invoke( std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs) ) )
    {
        return named_invoke( std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs) );
    }
}
```

◦

## C◦

`lhs *element_wise<'+'*> rhs`◦

\*dot\*\*cross\* **product**◦

\*\*◦ **C ++extra** () S◦

->\*then\*`std::function monadic ->*bind*`◦ `Op`

```
namespace named_operator {
    template<class D>struct make_operator{constexpr make_operator(){}};

    template<class T, char, class O> struct half_apply { T&& lhs; };

    template<class Lhs, class Op>
    half_apply<Lhs, '*', Op> operator*( Lhs&& lhs, make_operator<Op> ) {
        return {std::forward<Lhs>(lhs)};
    }

    template<class Lhs, class Op, class Rhs>
    auto operator*( half_apply<Lhs, '*', Op>&& lhs, Rhs&& rhs )
    -> decltype( named_invoke( std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs) ) )
    {
        return named_invoke( std::forward<Lhs>(lhs.lhs), Op{}, std::forward<Rhs>(rhs) );
    }
}
```

## C ++17◦

<https://riptutorial.com/zh-TW/cplusplus/topic/562/>

# 133:

- ○ ○
- ○ ○
- ○ ○
- ○ Class c(value) Class○
  - Class c = value ○ Class○
  - Nonclass c = classObject ○ ○
  - R &r = classObject r ○ ○
  - Class c{1, 2, 3} ○ Class○

## Examples

○

```
void f(int x);
void f(double x);
f(42); // calls f(int)
```

CV○

```
void f(int x);
void f(double x);
f(42); // calls f(int)
```

“ T”“T”TT\*○

```
void f(int x);
void f(double x);
f(42); // calls f(int)
```

“ ”○

- void f(int a); f(42);

○

○ ○

◦

```
void f(int x);
struct S {
    void f(double x);
    void g() { f(42); } // calls S::f because global f is not visible here,
                        // even though it would be a better match
};
```

◦ ◦

```
void f(int x);
struct S {
    void f(double x);
    void g() { f(42); } // calls S::f because global f is not visible here,
                        // even though it would be a better match
};
```

explicit

```
void f(int x);
struct S {
    void f(double x);
    void g() { f(42); } // calls S::f because global f is not visible here,
                        // even though it would be a better match
};
```

```
struct A {
    A() = default;           // #1
    A(A const& ) = default; // #2

    template <class T>
    A(T&& );              // #3
};
```

A◦

```
struct A {
    A() = default;           // #1
    A(A const& ) = default; // #2

    template <class T>
    A(T&& );              // #3
};
```

```
struct A {
    A() = default;           // #1
    A(A const& ) = default; // #2

    template <class T>
    A(T&& );              // #3
};
```

#3#2**CV**◦

## SFINAE

```
struct A {  
    A() = default;           // #1  
    A(A const& ) = default; // #2  
  
    template <class T>  
    A(T&& );                // #3  
};
```

AAo - o

1. o o

2. o o

```
void f(char);           // (1)  
void f(int ) = delete; // (2)  
void f();               // (3)  
void f(int& );         // (4)  
  
f(4); // 1,2 are viable (even though 2 is deleted!)  
// 3 is not viable because the argument lists don't match  
// 4 is not viable because we cannot bind a temporary to  
//      a non-const lvalue reference
```

3. o F1F2 F1F2...

3.1. F1F2

```
void f(char);           // (1)  
void f(int ) = delete; // (2)  
void f();               // (3)  
void f(int& );         // (4)  
  
f(4); // 1,2 are viable (even though 2 is deleted!)  
// 3 is not viable because the argument lists don't match  
// 4 is not viable because we cannot bind a temporary to  
//      a non-const lvalue reference
```

3.2. F1F2

```
void f(char);           // (1)  
void f(int ) = delete; // (2)  
void f();               // (3)  
void f(int& );         // (4)  
  
f(4); // 1,2 are viable (even though 2 is deleted!)  
// 3 is not viable because the argument lists don't match  
// 4 is not viable because we cannot bind a temporary to  
//      a non-const lvalue reference
```

3.3. F1F2

```

void f(char);           // (1)
void f(int ) = delete; // (2)
void f();               // (3)
void f(int& );        // (4)

f(4); // 1,2 are viable (even though 2 is deleted!)
// 3 is not viable because the argument lists don't match
// 4 is not viable because we cannot bind a temporary to
//      a non-const lvalue reference

```

### 3.4. F1F2

```

void f(char);           // (1)
void f(int ) = delete; // (2)
void f();               // (3)
void f(int& );        // (4)

f(4); // 1,2 are viable (even though 2 is deleted!)
// 3 is not viable because the argument lists don't match
// 4 is not viable because we cannot bind a temporary to
//      a non-const lvalue reference

```

### 3.5. F1F2F1F2°

```

void f(char);           // (1)
void f(int ) = delete; // (2)
void f();               // (3)
void f(int& );        // (4)

f(4); // 1,2 are viable (even though 2 is deleted!)
// 3 is not viable because the argument lists don't match
// 4 is not viable because we cannot bind a temporary to
//      a non-const lvalue reference

```

◦ ◦

```

void f(char);           // (1)
void f(int ) = delete; // (2)
void f();               // (3)
void f(int& );        // (4)

f(4); // 1,2 are viable (even though 2 is deleted!)
// 3 is not viable because the argument lists don't match
// 4 is not viable because we cannot bind a temporary to
//      a non-const lvalue reference

```

```

void f(char);           // (1)
void f(int ) = delete; // (2)
void f();               // (3)
void f(int& );        // (4)

f(4); // 1,2 are viable (even though 2 is deleted!)
// 3 is not viable because the argument lists don't match
// 4 is not viable because we cannot bind a temporary to
//      a non-const lvalue reference

```

◦

```
void f(int x);
void f(short x);
signed char c = 42;
f(c); // calls f(int); promotion to int is better than conversion to short
short s = 42;
f(s); // calls f(short); exact match is better than promotion to int
```

floatdouble◦

```
void f(int x);
void f(short x);
signed char c = 42;
f(c); // calls f(int); promotion to int is better than conversion to short
short s = 42;
f(s); // calls f(short); exact match is better than promotion to int
```

◦

```
void f(int x);
void f(short x);
signed char c = 42;
f(c); // calls f(int); promotion to int is better than conversion to short
short s = 42;
f(s); // calls f(short); exact match is better than promotion to int
```

f◦

```
void f(int x);
void f(short x);
signed char c = 42;
f(c); // calls f(int); promotion to int is better than conversion to short
short s = 42;
f(s); // calls f(short); exact match is better than promotion to int
```

```
struct A { int m; };
struct B : A {};
struct C : B {};
```

◦ ◦

```
struct A { int m; };
struct B : A {};
struct C : B {};
```

void\*◦

```
struct A { int m; };
struct B : A {};
struct C : B {};
```

◦ “”◦ ◦

```
struct A { int m; };
struct B : A {};
struct C : B {};
```

◦

```
struct A { int m; };
struct B : A {};
struct C : B {};
```

## constnessvolatility

T\*const T\*◦

```
struct Base {};
struct Derived : Base {};
void f(Base* pb);
void f(const Base* pb);
void f(const Derived* pd);
void f(bool b);

Base b;
f(&b); // f(Base*) is better than f(const Base*)
Derived d;
f(&d); // f(const Derived*) is better than f(Base*) though;
        // constness is only a "tie-breaker" rule
```

T&const T&◦

```
struct Base {};
struct Derived : Base {};
void f(Base* pb);
void f(const Base* pb);
void f(const Derived* pd);
void f(bool b);

Base b;
f(&b); // f(Base*) is better than f(const Base*)
Derived d;
f(&d); // f(const Derived*) is better than f(Base*) though;
        // constness is only a "tie-breaker" rule
```

## constconstconst◦

```
struct Base {};
struct Derived : Base {};
void f(Base* pb);
void f(const Base* pb);
void f(const Derived* pd);
void f(bool b);

Base b;
```

```
f(&b); // f(Base*) is better than f(const Base*)
Derived d;
f(&d); // f(const Derived*) is better than f(Base*) though;
        // constness is only a "tie-breaker" rule
```

◦

```
struct Base {};
struct Derived : Base {};
void f(Base* pb);
void f(const Base* pb);
void f(const Derived* pd);
void f(bool b);

Base b;
f(&b); // f(Base*) is better than f(const Base*)
Derived d;
f(&d); // f(const Derived*) is better than f(Base*) though;
        // constness is only a "tie-breaker" rule
```

<https://riptutorial.com/zh-TW/cplusplus/topic/2021/>

## Examples

### std :: recursive\_mutex

```
- .  
.  
  
std::atomic_int temp{0};  
std::recursive_mutex _mutex;  
  
//launch_deferred launches asynchronous tasks on the same thread id  
  
auto future1 = std::async(  
    std::launch::deferred,  
    [&] ()  
    {  
        std::cout << std::this_thread::get_id() << std::endl;  
  
        std::this_thread::sleep_for(std::chrono::seconds(3));  
        std::unique_lock<std::recursive_mutex> lock(_mutex);  
        temp=0;  
    });  
  
auto future2 = std::async(  
    std::launch::deferred,  
    [&] ()  
    {  
        std::cout << std::this_thread::get_id() << std::endl;  
        while ( true )  
        {  
            std::this_thread::sleep_for(std::chrono::milliseconds(1));  
            std::unique_lock<std::recursive_mutex> lock(_mutex,  
std::try_to_lock);  
            if ( temp < INT_MAX )  
                temp++;  
  
            cout << temp << endl;  
        }  
    });  
future1.get();  
future2.get();
```

<https://riptutorial.com/zh-TW/cplusplus/topic/9929/>

# 135:

◦ ◦

## Examples

### ◦ C ++ 111100CodeReview◦

```
#include <iostream>
#include <vector>
#include <cmath>

int main()
{
    int l = 100;
    bool isprime;
    std::vector<int> primes;
    primes.push_back(2);
    for (int no = 3; no < l; no += 2) {
        isprime = true;
        for (int primecount=0; primes[primecount] <= std::sqrt(no); ++primecount) {
            if (no % primes[primecount] == 0) {
                isprime = false;
                break;
            } else if (primes[primecount] * primes[primecount] > no) {
                std::cout << no << "\n";
                break;
            }
        }
        if (isprime) {
            std::cout << no << " ";
            primes.push_back(no);
        }
    }
    std::cout << "\n";
}
```

3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

2◦ - ◦

```
#include <iostream>
#include <vector>
#include <cmath>

int main()
{
    int l = 100;
    bool isprime;
    std::vector<int> primes;
    primes.push_back(2);
    for (int no = 3; no < l; no += 2) {
        isprime = true;
        for (int primecount=0; primes[primecount] <= std::sqrt(no); ++primecount) {
            if (no % primes[primecount] == 0) {
```

```
        isprime = false;
        break;
    } else if (primes[primecount] * primes[primecount] > no) {
        std::cout << no << "\n";
        break;
    }
}
if (isprime) {
    std::cout << no << " ";
    primes.push_back(no);
}
std::cout << "\n";
}
```

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

if  $\circ \circ \circ$

```
#include <iostream>
#include <vector>
#include <cmath>

int main()
{
    int l = 100;
    bool isprime;
    std::vector<int> primes;
    primes.push_back(2);
    for (int no = 3; no < l; no += 2) {
        isprime = true;
        for (int primecount=0; primes[primecount] <= std::sqrt(no); ++primecount) {
            if (no % primes[primecount] == 0) {
                isprime = false;
                break;
            } else if (primes[primecount] * primes[primecount] > no) {
                std::cout << no << "\n";
                break;
            }
        }
        if (isprime) {
            std::cout << no << " ";
            primes.push_back(no);
        }
    }
    std::cout << "\n";
}
```

◦ primecount◦ no “”no◦ break◦

```
#include <iostream>
#include <vector>
#include <cmath>

int main()
{
    int l = 100;
    bool isprime;
```

```

    std::vector<int> primes;
    primes.push_back(2);
    for (int no = 3; no < l; no += 2) {
        isprime = true;
        for (int primecount=0; primes[primecount] <= std::sqrt(no); ++primecount) {
            if (no % primes[primecount] == 0) {
                isprime = false;
                break;
            } else if (primes[primecount] * primes[primecount] > no) {
                std::cout << no << "\n";
                break;
            }
        }
        if (isprime) {
            std::cout << no << " ";
            primes.push_back(no);
        }
    }
    std::cout << "\n";
}

```

## main“range-for”

```

#include <iostream>
#include <vector>
#include <cmath>

int main()
{
    int l = 100;
    bool isprime;
    std::vector<int> primes;
    primes.push_back(2);
    for (int no = 3; no < l; no += 2) {
        isprime = true;
        for (int primecount=0; primes[primecount] <= std::sqrt(no); ++primecount) {
            if (no % primes[primecount] == 0) {
                isprime = false;
                break;
            } else if (primes[primecount] * primes[primecount] > no) {
                std::cout << no << "\n";
                break;
            }
        }
        if (isprime) {
            std::cout << no << " ";
            primes.push_back(no);
        }
    }
    std::cout << "\n";
}

```

◦ ◦

**CC ++** goto cleanup ◦ goto ◦ **return** ◦ goto cleanup ◦

```

short calculate(VectorStr **data) {
    short result = FALSE;

```

```

VectorStr *vec = NULL;
if (!data)
    goto cleanup; //< Could become return false

// ... Calculation which 'new's VectorStr

result = TRUE;
cleanup:
    delete [] vec;
    return result;
}

```

## C ++RAII

```

short calculate(VectorStr **data) {
    short result = FALSE;
    VectorStr *vec = NULL;
    if (!data)
        goto cleanup; //< Could become return false

    // ... Calculation which 'new's VectorStr

    result = TRUE;
cleanup:
    delete [] vec;
    return result;
}

```

- std::unique\_ptr<std::vector<VectorStr>>。

<https://riptutorial.com/zh-TW/cplusplus/topic/7600/>

# 136:

C ++◦◦◦

- `asm string-literal ;`
- `noexcept //1`
- `noexcept constant-expression //2`
- `noexcept //2`
- `sizeof unary-expression`
- `sizeof type-id`
- `sizeof ... identifier //C ++ 11`
- `typename nested-name-specifier identifier //1`
- `typename nested-name-specifier template opt simple-template-id //1`
- `typename identifier opt //2`
- `typename ... identifier opt //2;C ++ 11`
- `typename identifier opt = type-id //2`
- `template < template-parameter-list > typename ... opt identifier opt //3`
- `template < template-parameter-list > typename identifier opt = id-expression //3`
  
- 
- 
- `asm`
- `auto C ++ 11 C ++ 11`
- `bool`
- `break`
- `case`
- `catch`
- `char`
- `char16_t C ++ 11`
- `char32_t C ++ 11`
- `class`
- `const`
- `constexpr C ++ 11`
- `const_cast`
- `continue`
- `decltype C ++ 11`
- `default`
- `delete C ++ 11`
- `do`
- `double`
- `dynamic_cast`
- `else`
- `enum`
- `explicit`
- `export`
- `extern`
- `false`
- `float`
- `for`

- friend
  - goto
  - if
  - inline C++ 11 C++ 17
  - int
  - long
  - mutable
  - namespace
  - new
  - noexcept C++ 11
  - nullptr C++ 11
  - operator
  - private
  - protected
  - public
  - register
  - reinterpret\_cast
  - return
  - short
  - signed
  - sizeof
  - static
  - static\_assert C++ 11
  - static\_cast
  - struct
  - switch
  - template
  - this
  - thread\_local C++ 11
  - throw
  - true
  - try
  - typedef
  - typeid
  - typename
  - union
  - unsigned
  - using
  - virtual
  - void
  - volatile
  - wchar\_t
  - while

finaloverride. .

and and\_eq bitand bitor compl not not\_eq or or\_eq xor xor\_eq &= & | o ~ ! != || |= ^^^= o

C ++。

- A vertical column of three solid black circular dots.

- 
- 
- 
- /
- 

## Examples

### ASM

```
asm o .
```

```
asm o C++asmconstexpr.
```

```
[[noreturn]] void halt_system() {
    asm("hlt");
}
```

1. o

```
class MyVector {
public:
    explicit MyVector(uint64_t size);
};

MyVector v1(100); // ok
uint64_t len1 = 100;
MyVector v2{len1}; // ok, len1 is uint64_t
int len2 = 100;
MyVector v3{len2}; // ill-formed, implicit conversion from int to uint64_t
```

C++11C++11 `explicit`.

```
class MyVector {
public:
    explicit MyVector(uint64_t size);
};

MyVector v1(100); // ok
uint64_t len1 = 100;
MyVector v2{len1}; // ok, len1 is uint64_t
int len2 = 100;
MyVector v3{len2}; // ill-formed, implicit conversion from int to uint64_t
```

### C++11

2. o

```
class MyVector {
public:
```

```

    explicit MyVector(uint64_t size);
};

MyVector v1(100); // ok
uint64_t len1 = 100;
MyVector v2{len1}; // ok, len1 is uint64_t
int len2 = 100;
MyVector v3{len2}; // ill-formed, implicit conversion from int to uint64_t

```

## noexcept

### C++ 11

1. . noexcept .

```

#include <iostream>
#include <stdexcept>
void foo() { throw std::runtime_error("oops"); }
void bar() {}
struct S {};
int main() {
    std::cout << noexcept(foo()) << '\n'; // prints 0
    std::cout << noexcept(bar()) << '\n'; // prints 0
    std::cout << noexcept(1 + 1) << '\n'; // prints 1
    std::cout << noexcept(S()) << '\n'; // prints 1
}

```

bar() noexcept (bar()) **false** bar().

2. . . .

```

#include <iostream>
#include <stdexcept>
void foo() { throw std::runtime_error("oops"); }
void bar() {}
struct S {};
int main() {
    std::cout << noexcept(foo()) << '\n'; // prints 0
    std::cout << noexcept(bar()) << '\n'; // prints 0
    std::cout << noexcept(1 + 1) << '\n'; // prints 1
    std::cout << noexcept(S()) << '\n'; // prints 1
}

```

f4 f5f6. f6. f2. noexceptnoexcept(false) f1f3f3.

### C++ 17

noexcept f1 f2f3f4 f5f6. noexcept.

```

#include <iostream>
#include <stdexcept>
void foo() { throw std::runtime_error("oops"); }
void bar() {}
struct S {};
int main() {

```

```
    std::cout << noexcept(foo()) << '\n'; // prints 0
    std::cout << noexcept(bar()) << '\n'; // prints 0
    std::cout << noexcept(1 + 1) << '\n'; // prints 1
    std::cout << noexcept(S()) << '\n'; // prints 1
}
```

### 1. typename<sup>o</sup> std::decay<T> type typename<sup>o</sup> “““typename””

```
template <class T>
auto decay_copy(T&& r) -> typename std::decay<T>::type;
```

### 2. o class<sup>o</sup>

```
template <class T>
auto decay_copy(T&& r) -> typename std::decay<T>::type;
```

## C++ 17

### 3. typename class<sup>o</sup>

```
template <class T>
auto decay_copy(T&& r) -> typename std::decay<T>::type;
```

## sizeof

### o o size<sup>o</sup> std::size\_t<sup>o</sup>

o

- sizeof<sup>o</sup>
- sizeofvoid<sup>o</sup>
- sizeof<sup>T&T&&</sup> sizeof(T)<sup>o</sup>
- sizeof<sup>o</sup> sizeof0<sup>o</sup>
- char signed charunsigned char1<sup>o</sup>.char<sup>o</sup> 88char<sup>o</sup>

### expr sizeof( expr ) sizeof(T) Texpr<sup>o</sup>

```
int a[100];
std::cout << "The number of bytes in `a` is: " << sizeof a;
memset(a, 0, sizeof a); // zeroes out the array
```

## C++ 11

### sizeof...o

```
int a[100];
std::cout << "The number of bytes in `a` is: " << sizeof a;
memset(a, 0, sizeof a); // zeroes out the array
```

## **void C ++**

1. void。 void。 void”。
2. void \*constvolatile。 void。 void。
3. voidC ++。

```
void vobject;    // C2182
void *pv;      // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

## **C ++**

- 1.。

```
void vobject;    // C2182
void *pv;      // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

## **C ++**

1. virtual。

```
void vobject;    // C2182
void *pv;      // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

1. **type-specifiers**。

2. **member-function-declarator**。

3. **access-specifier** publicprotectedprivate。 virtual。

4. **base-class-name**

1. this。 。 this。

```
void vobject;    // C2182
void *pv;      // okay
```

```
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

this;sizeof。

```
void vobject; // C2182
void *pv; // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

## C++

1. C++trythrowcatch。
2. try。
3. throw - try。 throw。 std::exception。 std::exception。
4. trycatch。 catch。

```
void vobject; // C2182
void *pv; // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

try。 throw - 。 catch。 throwcatch。

```
void vobject; // C2182
void *pv; // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

## C++

1. . . . friend。。
- 2.。

```
void vobject; // C2182
void *pv; // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
```

```
pint = (int *)pv;
```

1.◦ ;◦

2.◦ ->◦

3.◦ ◦ ◦

```
void vobject;    // C2182
void *pv;        // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

```
void vobject;    // C2182
void *pv;        // okay
int *pint; int i;
int main() {
    pv = &i;
    // Cast optional in C required in C++
    pint = (int *)pv;
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4891/>

# 137:

C++<random> ◦ ◦

◦ ◦

◦ ◦ ◦

◦ ◦

## Examples

std::random\_device◦

```
#include <iostream>
#include <random>

int main()
{
    std::random_device crypto_random_generator;
    std::uniform_int_distribution<int> int_distribution(0, 9);

    int actual_distribution[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    for(int i = 0; i < 10000; i++) {
        int result = int_distribution(crypto_random_generator);
        actual_distribution[result]++;
    }

    for(int i = 0; i < 10; i++) {
        std::cout << actual_distribution[i] << " ";
    }

    return 0;
}
```

std::random\_device◦

std::random\_device ◦

entropy GCClibstdc ++LLVMlibc ++◦

◦ ◦ ◦

```
#include <iostream>
#include <random>

int main()
{
    std::default_random_engine pseudo_random_generator;
    std::uniform_int_distribution<int> int_distribution(0, 9);
```

```
int actual_distribution[10] = {0,0,0,0,0,0,0,0,0,0};

for(int i = 0; i < 10000; i++) {
    int result = int_distribution(pseudo_random_generator);
    actual_distribution[result]++;
}

for(int i = 0; i <= 9; i++) {
    std::cout << actual_distribution[i] << " ";
}

return 0;
}
```

[0,9]◦ ◦

```
std::uniform_int_distribution<T>; std::uniform_real_distribution<T>;
```

```
#include <iostream>
#include <random>

int main()
{
    std::default_random_engine pseudo_random_generator;
    std::uniform_int_distribution<int> int_distribution(0, 9);
    std::uniform_real_distribution<float> float_distribution(0.0, 1.0);
    std::discrete_distribution<int> rigged_dice({1,1,1,1,1,100});

    std::cout << int_distribution(pseudo_random_generator) << std::endl;
    std::cout << float_distribution(pseudo_random_generator) << std::endl;
    std::cout << (rigged_dice(pseudo_random_generator) + 1) << std::endl;

    return 0;
}
```

◦ ◦ rigged dice 05 5100 / 105 ◦

<https://riptutorial.com/zh-TW/cplusplus/topic/1541/>

# 138:

- //
  - variable.member\_function;
  - variable\_pointer-> member\_function;
- //
  - ret\_type class\_name :: member\_functioncv-qualifiers {
    - ;
    - }
- //
  - class class\_name {
    - virt-specifier ret\_type member\_functioncv-qualifiers virt-specifier-seq;
    - // virt-specifier“virtual”。
    - // cv-qualifiers“const”/“volatile”。
    - // virt-specifier-seq“”/“”。
    - }

staticclass / struct / union◦ static◦

◦

## Examples

classstruct◦ ; ::◦

```
class CL {
public:
    void definedInside() {}
    void definedOutside();
};
```

```
void CL::definedOutside() {}
```

. -> ;this ◦

```
class CL {
public:
    void definedInside() {}
    void definedOutside();
};
```

```
void CL::definedOutside() {}
```

◦ /◦

```

class CL {
public:
    void definedInside() {}
    void definedOutside();
};

void CL::definedOutside() {}

```

**gettermutator setter**◦

```

class Encapsulator {
    int encapsulated;

public:
    int get_encapsulated() const { return encapsulated; }
    void set_encapsulated(int e) { encapsulated = e; }

    void some_func() {
        do_something_with(encapsulated);
    }
};

```

encapsulated;get\_encapsulated()set\_encapsulated()◦◦ [gettersetter;◦ ]

◦

```

struct HiddenBase {
    void f(int) { std::cout << "int" << std::endl; }
    void f(bool) { std::cout << "bool" << std::endl; }
    void f(std::string) { std::cout << "std::string" << std::endl; }
};

struct HidingDerived : HiddenBase {
    void f(float) { std::cout << "float" << std::endl; }
};

// ...

HiddenBase hb;
HidingDerived hd;
std::string s;

hb.f(1);      // Output: int
hb.f(true);   // Output: bool
hb.f(s);      // Output: std::string;

hd.f(1.0f);   // Output: float
hd.f(3);      // Output: float
hd.f(true);   // Output: float
hd.f(s);      // Error: Can't convert from std::string to float.

```

hd.f(s);◦ using-declaration”◦

```

struct HiddenBase {
    void f(int) { std::cout << "int" << std::endl; }
    void f(bool) { std::cout << "bool" << std::endl; }
    void f(std::string) { std::cout << "std::string" << std::endl; }

```

```

};

struct HidingDerived : HiddenBase {
    void f(float) { std::cout << "float" << std::endl; }
};

// ...

HiddenBase hb;
HidingDerived hd;
std::string s;

hb.f(1);      // Output: int
hb.f(true);   // Output: bool
hb.f(s);      // Output: std::string;

hd.f(1.0f);   // Output: float
hd.f(3);      // Output: float
hd.f(true);   // Output: float
hd.f(s);      // Error: Can't convert from std::string to float.

```

## using。

```

struct HiddenBase {
    void f(int) { std::cout << "int" << std::endl; }
    void f(bool) { std::cout << "bool" << std::endl; }
    void f(std::string) { std::cout << "std::string" << std::endl; }
};

struct HidingDerived : HiddenBase {
    void f(float) { std::cout << "float" << std::endl; }
};

// ...

HiddenBase hb;
HidingDerived hd;
std::string s;

hb.f(1);      // Output: int
hb.f(true);   // Output: bool
hb.f(s);      // Output: std::string;

hd.f(1.0f);   // Output: float
hd.f(3);      // Output: float
hd.f(true);   // Output: float
hd.f(s);      // Error: Can't convert from std::string to float.

```

## publicprotected Using。

```

struct HiddenBase {
    void f(int) { std::cout << "int" << std::endl; }
    void f(bool) { std::cout << "bool" << std::endl; }
    void f(std::string) { std::cout << "std::string" << std::endl; }
};

struct HidingDerived : HiddenBase {
    void f(float) { std::cout << "float" << std::endl; }
}

```

```
};

// ...

HiddenBase hb;
HidingDerived hd;
std::string s;

hb.f(1);      // Output: int
hb.f(true);   // Output: bool
hb.f(s);      // Output: std::string;

hd.f(1.0f);   // Output: float
hd.f(3);      // Output: float
hd.f(true);   // Output: float
hd.f(s);      // Error: Can't convert from std::string to float.
```

◦

```
struct HiddenBase {
    void f(int) { std::cout << "int" << std::endl; }
    void f(bool) { std::cout << "bool" << std::endl; }
    void f(std::string) { std::cout << "std::string" << std::endl; }
};

struct HidingDerived : HiddenBase {
    void f(float) { std::cout << "float" << std::endl; }
};

// ...

HiddenBase hb;
HidingDerived hd;
std::string s;

hb.f(1);      // Output: int
hb.f(true);   // Output: bool
hb.f(s);      // Output: std::string;

hd.f(1.0f);   // Output: float
hd.f(3);      // Output: float
hd.f(true);   // Output: float
hd.f(s);      // Error: Can't convert from std::string to float.
```

`virtual . ;vtablevftable .`

```
struct Base {
    virtual void func() { std::cout << "In Base." << std::endl; }
};

struct Derived : Base {
    void func() override { std::cout << "In Derived." << std::endl; }
};

void slicer(Base x) { x.func(); }

// ...
```

```

Base b;
Derived d;

Base *pb = &b, *pd = &d; // Pointers.
Base &rb = b, &rd = d; // References.

b.func(); // Output: In Base.
d.func(); // Output: In Derived.

pb->func(); // Output: In Base.
pd->func(); // Output: In Derived.

rb.func(); // Output: In Base.
rd.func(); // Output: In Derived.

slicer(b); // Output: In Base.
slicer(d); // Output: In Base.

```

pdBase\* rdBase& func() Derived::func() Base::func() ;DerivedvtableBase::func() Derived::func() 。  
slicer()Base::func() Derived 。 DerivedBaseDerivedBase。

**virtual**virtual 。 virtual。

```

struct Base {
    virtual void func() { std::cout << "In Base." << std::endl; }
};

struct Derived : Base {
    void func() override { std::cout << "In Derived." << std::endl; }
};

void slicer(Base x) { x.func(); }

// ...

Base b;
Derived d;

Base *pb = &b, *pd = &d; // Pointers.
Base &rb = b, &rd = d; // References.

b.func(); // Output: In Base.
d.func(); // Output: In Derived.

pb->func(); // Output: In Base.
pd->func(); // Output: In Derived.

rb.func(); // Output: In Base.
rd.func(); // Output: In Derived.

slicer(b); // Output: In Base.
slicer(d); // Output: In Base.

```

;virtual 。

```

struct Base {
    virtual void func() { std::cout << "In Base." << std::endl; }
}

```

```

};

struct Derived : Base {
    void func() override { std::cout << "In Derived." << std::endl; }
};

void slicer(Base x) { x.func(); }

// ...

Base b;
Derived d;

Base *pb = &b, *pd = &d; // Pointers.
Base &rb = b, &rd = d; // References.

b.func(); // Output: In Base.
d.func(); // Output: In Derived.

pb->func(); // Output: In Base.
pd->func(); // Output: In Derived.

rb.func(); // Output: In Base.
rd.func(); // Output: In Derived.

slicer(b); // Output: In Base.
slicer(d); // Output: In Base.

```

## C++ 11

### C++ 11 `override`

```

struct Base {
    virtual void func() { std::cout << "In Base." << std::endl; }
};

struct Derived : Base {
    void func() override { std::cout << "In Derived." << std::endl; }
};

void slicer(Base x) { x.func(); }

// ...

Base b;
Derived d;

Base *pb = &b, *pd = &d; // Pointers.
Base &rb = b, &rd = d; // References.

b.func(); // Output: In Base.
d.func(); // Output: In Derived.

pb->func(); // Output: In Base.
pd->func(); // Output: In Derived.

rb.func(); // Output: In Base.
rd.func(); // Output: In Derived.

slicer(b); // Output: In Base.

```

```
slicer(d); // Output: In Base.
```

◦

virtual virtual◦

## C++11

override◦

```
struct Base {
    virtual void func() { std::cout << "In Base." << std::endl; }
};

struct Derived : Base {
    void func() override { std::cout << "In Derived." << std::endl; }
};

void slicer(Base x) { x.func(); }

// ...

Base b;
Derived d;

Base *pb = &b, *pd = &d; // Pointers.
Base &rb = b, &rd = d; // References.

b.func(); // Output: In Base.
d.func(); // Output: In Derived.

pb->func(); // Output: In Base.
pd->func(); // Output: In Derived.

rb.func(); // Output: In Base.
rd.func(); // Output: In Derived.

slicer(b); // Output: In Base.
slicer(d); // Output: In Base.
```

virtual virtual◦

```
struct Base {
    virtual void func() { std::cout << "In Base." << std::endl; }
};

struct Derived : Base {
    void func() override { std::cout << "In Derived." << std::endl; }
};

void slicer(Base x) { x.func(); }

// ...

Base b;
Derived d;
```

```

Base *pb = &b, *pd = &d; // Pointers.
Base &rb = b, &rd = d; // References.

b.func(); // Output: In Base.
d.func(); // Output: In Derived.

pb->func(); // Output: In Base.
pd->func(); // Output: In Derived.

rb.func(); // Output: In Base.
rd.func(); // Output: In Derived.

slicer(b); // Output: In Base.
slicer(d); // Output: In Base.

```

◦

## Const

this **CV***const* ◦ ◦ ◦ *constconstconst*◦

*constconst const*◦ *const constconstconst -ness*◦ **getter***const*◦

```

class ConstIncorrect {
    Field fld;

public:
    ConstIncorrect(const Field& f) : fld(f) {} // Modifies.

    const Field& get_field() { return fld; } // Doesn't modify; should be const.
    void set_field(const Field& f) { fld = f; } // Modifies.

    void do_something(int i) { // Modifies.
        fld.insert_value(i);
    }
    void do_nothing() {} // Doesn't modify; should be const.
};

class ConstCorrect {
    Field fld;

public:
    ConstCorrect(const Field& f) : fld(f) {} // Not const: Modifies.

    const Field& get_field() const { return fld; } // const: Doesn't modify.
    void set_field(const Field& f) { fld = f; } // Not const: Modifies.

    void do_something(int i) { // Not const: Modifies.
        fld.insert_value(i);
    }
    void do_nothing() const {} // const: Doesn't modify.
};

// ...

const ConstIncorrect i_cant_do_anything(make_me_a_field());
// Now, let's read it...
Field f = i_cant_do_anything.get_field();

```

```
// Error: Loses cv-qualifiers, get_field() isn't const.  
i_cant_do_anything.do_nothing();  
// Error: Same as above.  
// Oops.  
  
const ConstCorrect but_i_can(make_me_a_field());  
// Now, let's read it...  
Field f = but_i_can.get_field(); // Good.  
but_i_can.do_nothing(); // Good.
```

ConstIncorrectConstCorrectConstCorrect **cv-qualifying**。 constconstconst。

<https://riptutorial.com/zh-TW/cplusplus/topic/5661/>

# 139:

## C/ C++

- #include
- #define
- #if #ifdef
- /
- ◦ ◦

## Examples

### ◦ ◦ 2003 C ++§3.2◦

“”◦ #ifndef #endif #define #ifndef #endif◦

```
// Foo.h
#ifndef FOO_H_INCLUDED
#define FOO_H_INCLUDED

class Foo      //  a class definition
{
};

#endif
```

◦

◦ FOO\_H\_INCLUDED◦ ◦

◦

### C ++#pragma once◦ ISO C ++◦

```
// Foo.h
#ifndef FOO_H_INCLUDED
#define FOO_H_INCLUDED

class Foo      //  a class definition
{
};

#endif
```

#pragma once#pragma - -◦ #pragma once◦

C ++#include◦

◦

- ◦
- - ◦

- BasicPremiumPro - ◦

a

```
#ifdef _WIN32
#include <windows.h> // and other windows system files
#endif
#include <cstdio>

bool remove_file(const std::string &path)
{
#ifdef _WIN32
    return DeleteFile(path.c_str());
#elif defined(_POSIX_VERSION) || defined(__unix__)
    return (0 == remove(path.c_str()));
#elif defined(__APPLE__)
    //TODO: check if NSAPI has a more specific function with permission dialog
    return (0 == remove(path.c_str()));
#else
#error "This platform is not supported"
#endif
}
```

\_WIN32 \_\_APPLE\_\_ \_\_unix\_\_◦

b

```
#ifdef _WIN32
#include <windows.h> // and other windows system files
#endif
#include <cstdio>

bool remove_file(const std::string &path)
{
#ifdef _WIN32
    return DeleteFile(path.c_str());
#elif defined(_POSIX_VERSION) || defined(__unix__)
    return (0 == remove(path.c_str()));
#elif defined(__APPLE__)
    //TODO: check if NSAPI has a more specific function with permission dialog
    return (0 == remove(path.c_str()));
#else
#error "This platform is not supported"
#endif
}
```

c◦

```
#ifdef _WIN32
```

```

#include <windows.h> // and other windows system files
#endif
#include <cstdio>

bool remove_file(const std::string &path)
{
#ifdef _WIN32
    return DeleteFile(path.c_str());
#elif defined(_POSIX_VERSION) || defined(__unix__)
    return (0 == remove(path.c_str()));
#elif defined(__APPLE__)
    //TODO: check if NSAPI has a more specific function with permission dialog
    return (0 == remove(path.c_str()));
#else
#error "This platform is not supported"
#endif
}

```

- gcc -E

```

#endif _WIN32
#include <windows.h> // and other windows system files
#endif
#include <cstdio>

bool remove_file(const std::string &path)
{
#ifdef _WIN32
    return DeleteFile(path.c_str());
#elif defined(_POSIX_VERSION) || defined(__unix__)
    return (0 == remove(path.c_str()));
#elif defined(__APPLE__)
    //TODO: check if NSAPI has a more specific function with permission dialog
    return (0 == remove(path.c_str()));
#else
#error "This platform is not supported"
#endif
}

```

**Sample.cpp**#define OPTIMISE\_FOR\_OS\_X#define TESTING\_MODE 1**Sample.cpp**◦ . . OENABLE\_EXTRA\_DEBUGGING = 0-DENABLE\_EXTRA\_DEBUGGING = 1◦ . . #ifndef  
#error

```

#endif _WIN32
#include <windows.h> // and other windows system files
#endif
#include <cstdio>

bool remove_file(const std::string &path)
{
#ifdef _WIN32
    return DeleteFile(path.c_str());
#elif defined(_POSIX_VERSION) || defined(__unix__)
    return (0 == remove(path.c_str()));
#elif defined(__APPLE__)
    //TODO: check if NSAPI has a more specific function with permission dialog
    return (0 == remove(path.c_str()));

```

```
#else
#error "This platform is not supported"
#endif
}
```

◦ ◦ ◦

```
// This is an object-like macro
#define PI 3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r) (PI*(r)*(r))

// They can be used like this:
double pi_macro = PI;
double area_macro = AREA(4.6);
```

## QtQObjectQ\_OBJECT。

◦ ◦

---

◦ ◦

```
// This is an object-like macro
#define PI 3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r) (PI*(r)*(r))

// They can be used like this:
double pi_macro = PI;
double area_macro = AREA(4.6);
```

AREA()◦ ◦

```
// This is an object-like macro
#define PI 3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r) (PI*(r)*(r))
```

```

// They can be used like this:
double pi_macro    = PI;
double area_macro = AREA(4.6);

.

// This is an object-like macro
#define PI      3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r)  (PI*(r)*(r))

// They can be used like this:
double pi_macro    = PI;
double area_macro = AREA(4.6);

```

.

---

“”;

```

// This is an object-like macro
#define PI      3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r)  (PI*(r)*(r))

// They can be used like this:
double pi_macro    = PI;
double area_macro = AREA(4.6);

```

if...elseelseif。 “”。

```

// This is an object-like macro
#define PI      3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r)  (PI*(r)*(r))

// They can be used like this:

```

```
double pi_macro    = PI;
double area_macro = AREA(4.6);
```

◦

```
// This is an object-like macro
#define PI           3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r)      (PI*(r)*(r))

// They can be used like this:
double pi_macro    = PI;
double area_macro = AREA(4.6);
```

◦ ◦ ◦ ◦

```
// This is an object-like macro
#define PI           3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r)      (PI*(r)*(r))

// They can be used like this:
double pi_macro    = PI;
double area_macro = AREA(4.6);
```

◦

```
// This is an object-like macro
#define PI           3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r)      (PI*(r)*(r))

// They can be used like this:
double pi_macro    = PI;
double area_macro = AREA(4.6);
```

◦ 0 do-while◦ ◦

```
// This is an object-like macro
#define PI 3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r) (PI*(r)*(r))

// They can be used like this:
double pi_macro = PI;
double area_macro = AREA(4.6);
```

;“Varargs”\_VA\_ARGS\_。

```
// This is an object-like macro
#define PI 3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r) (PI*(r)*(r))

// They can be used like this:
double pi_macro = PI;
double area_macro = AREA(4.6);
```

\_\_VA\_ARGS\_\_。

```
// This is an object-like macro
#define PI 3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesnt know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r) (PI*(r)*(r))

// They can be used like this:
double pi_macro = PI;
double area_macro = AREA(4.6);
```

◦ Visual Studio ◦ GCC ◦ VA ARGS ◦ ## ◦ ◦

```
// This is an object-like macro
#define PI 3.14159265358979

// This is a function-like macro.
// Note that we can use previously defined macros
```

```

// in other macro definitions (object-like or function-like)
// But watch out, its quite useful if you know what you're doing, but the
// Compiler doesn't know which type to handle, so using inline functions instead
// is quite recommended (But e.g. for Minimum/Maximum functions it is quite useful)
#define AREA(r) (PI*(r)*(r))

// They can be used like this:
double pi_macro = PI;
double area_macro = AREA(4.6);

```

◦ ◦

## gcc3.0.0◦

```

#if __GNUC__ < 3
#error "This code requires gcc > 3.0.0"
#endif

```

## Apple◦

```

#if __GNUC__ < 3
#error "This code requires gcc > 3.0.0"
#endif

```

◦ ◦

## C ++

- \_\_LINE\_\_#line◦
- \_\_FILE\_\_#line◦
- \_\_DATE\_\_"Mmm dd yyyy" *Mmm*std::asctime()◦
- \_\_TIME\_\_"hh:mm:ss"◦
- \_\_cplusplusC ++C ++◦ 199711L**C ++ 98**C ++ 03 201103L**C ++ 11**201402L**C ++ 14**◦

## C ++ 11

- \_\_STDC\_HOSTED\_\_1 0◦

## C ++ 17

- \_\_STDCPP\_DEFAULT\_NEW\_ALIGNMENT\_\_size\_t - operator new◦
- \_\_STDC\_\_CC◦

## C ++ 11

- \_\_STDC\_VERSION\_\_C\_cplusplusC ++◦
- \_\_STDC\_MB\_MIGHT\_NEQ\_WC\_\_1 **if** (uintmax\_t)'x' != (uintmax\_t)L'x'
- wchar\_t**Unicode**\_\_STDC\_ISO\_10646\_\_ yyyyymmL**Unicode**◦
- \_\_STDCPP\_STRICT\_POINTER\_SAFETY\_\_1

- `__STDCPP_THREADS_1 -`

`__func__ . .`

`. .`

- **GCC**
- **Microsoft Visual C ++**
- 
- **C ++**

```
#ifdef __cplusplus // if compiled by C++ compiler
extern "C"{ // C code has to be decorated
    // C library header declarations here
}
#endif
```

## C ++ 11

```
#ifdef __cplusplus // if compiled by C++ compiler
extern "C"{ // C code has to be decorated
    // C library header declarations here
}
#endif
```

```
#ifdef __cplusplus // if compiled by C++ compiler
extern "C"{ // C code has to be decorated
    // C library header declarations here
}
#endif
```

X-

◦

X-macro◦

```
#define LIST \
    X(dog) \
    X(cat) \
    X(racoon)

// class Animal {
// public:
//     void say();
// };

#define X(name) Animal name;
LIST
#undef X

int main() {
#define X(name) name.say();
    LIST
```

```
#undef X

    return 0;
}
```

```
#define LIST \
X(dog) \
X(cat) \
X(racoon)

// class Animal {
// public:
//     void say();
// };

#define X(name) Animal name;
LIST
#undef X

int main() {
#define X(name) name.say();
    LIST
#undef X

    return 0;
}
```

100°

[https://en.wikipedia.org/wiki/X\\_Macro](https://en.wikipedia.org/wiki/X_Macro)

## X-macros

---

LISTseaminglyX

```
#define LIST \
X(dog) \
X(cat) \
X(racoon)

// class Animal {
// public:
//     void say();
// };

#define X(name) Animal name;
LIST
#undef X

int main() {
#define X(name) name.say();
    LIST
#undef X

    return 0;
}
```

```

#define LIST \
    X(dog) \
    X(cat) \
    X(racoon)

// class Animal {
// public:
//     void say();
// };

#define X(name) Animal name;
LIST
#undef X

int main() {
#define X(name) name.say();
    LIST
#undef X

    return 0;
}

```

MACRO -

```

#define LIST \
    X(dog) \
    X(cat) \
    X(racoon)

// class Animal {
// public:
//     void say();
// };

#define X(name) Animal name;
LIST
#undef X

int main() {
#define X(name) name.say();
    LIST
#undef X

    return 0;
}

```

LIST LIST。

```

#define LIST \
    X(dog) \
    X(cat) \
    X(racoon)

// class Animal {
// public:
//     void say();
// };

```

```
#define X(name) Animal name;
LIST
#undef X

int main() {
#define X(name) name.say();
    LIST
#undef X

    return 0;
}
```

```
#define LIST \
X(dog) \
X(cat) \
X(racoon)

// class Animal {
// public:
//     void say();
// };

#define X(name) Animal name;
LIST
#undef X

int main() {
#define X(name) name.say();
    LIST
#undef X

    return 0;
}
```

## #pragma

C ++#pragma once◦ ISO C ++◦

```
// Foo.h
#pragma once

class Foo
{
```

#pragma once#pragma - -◦ #pragma once◦

- - #pragma once◦◦ #pragma onceinclude guard◦

WindowsMFC #pragma onceinclude guardsVisual Studioadd class add dialog add windows◦ C ++
Windows◦

#◦◦

```
// preprocessor will convert the parameter x to the string literal x
```

```
#define PRINT(x) printf(#x "\n")

PRINT(This line will be converted to string by preprocessor);
// Compiler sees
printf("This line will be converted to string by preprocessor""\n");
```

printf()。

- **print**◦

```
// preprocessor will convert the parameter x to the string literal x
#define PRINT(x) printf(#x "\n")

PRINT(This line will be converted to string by preprocessor);
// Compiler sees
printf("This line will be converted to string by preprocessor""\n");
```

◦

```
// preprocessor will convert the parameter x to the string literal x
#define PRINT(x) printf(#x "\n")

PRINT(This line will be converted to string by preprocessor);
// Compiler sees
printf("This line will be converted to string by preprocessor""\n");
```

---

## #**operatorToken** pasting◦

```
// preprocessor will convert the parameter x to the string literal x
#define PRINT(x) printf(#x "\n")

PRINT(This line will be converted to string by preprocessor);
// Compiler sees
printf("This line will be converted to string by preprocessor""\n");
```

```
// preprocessor will convert the parameter x to the string literal x
#define PRINT(x) printf(#x "\n")

PRINT(This line will be converted to string by preprocessor);
// Compiler sees
printf("This line will be converted to string by preprocessor""\n");
```

<https://riptutorial.com/zh-TW/cplusplus/topic/1098/>

## Examples

```
{ ... }◦ ◦
```

```
{
    int x = 100;
    // ^
    // Scope of `x` begins here
    //
} // <- Scope of `x` ends here
```

```
◦
```

```
{
    int x = 100;
    // ^
    // Scope of `x` begins here
    //
} // <- Scope of `x` ends here
```

```
extern◦ /◦
```

```
// File my_globals.h:

#ifndef __MY_GLOBALS_H__
#define __MY_GLOBALS_H__

extern int circle_radius; // Promise to the compiler that circle_radius
                        // will be defined somewhere

#endif
```

```
// File my_globals.h:

#ifndef __MY_GLOBALS_H__
#define __MY_GLOBALS_H__

extern int circle_radius; // Promise to the compiler that circle_radius
                        // will be defined somewhere

#endif
```

```
// File my_globals.h:

#ifndef __MY_GLOBALS_H__
#define __MY_GLOBALS_H__

extern int circle_radius; // Promise to the compiler that circle_radius
                        // will be defined somewhere
```

```
#endif

// File my_globals.h:

#ifndef __MY_GLOBALS_H__
#define __MY_GLOBALS_H__

extern int circle_radius; // Promise to the compiler that circle_radius
                         // will be defined somewhere

#endif
```

<https://riptutorial.com/zh-TW/cplusplus/topic/3453/>

# 141:

C ++CC ++C。 GNU C

C。 C“#include”。

◦  
• ◦ ◦  
• ◦ ◦

C。

/。

C ++/。

C ++C ++ 20。

## Examples

// filename。

```
// my_function.h

/* Note how this header contains only a declaration of a function.
 * Header functions usually do not define implementations for declarations
 * unless code must be further processed at compile time, as in templates.
 */

/* Also, usually header files include preprocessor guards so that every header
 * is never included twice.
 *
 * The guard is implemented by checking if a header-file unique preprocessor
 * token is defined, and only including the header if it hasn't been included
 * once before.
 */
#ifndef MY_FUNCTION_H
#define MY_FUNCTION_H

// global_value and my_function() will be
// recognized as the same constructs if this header is included by different files.
const int global_value = 42;
int my_function();

#endif // MY_FUNCTION_H
```

```
// my_function.h
```

```
/* Note how this header contains only a declaration of a function.  
 * Header functions usually do not define implementations for declarations  
 * unless code must be further processed at compile time, as in templates.  
 */  
  
/* Also, usually header files include preprocessor guards so that every header  
 * is never included twice.  
 *  
 * The guard is implemented by checking if a header-file unique preprocessor  
 * token is defined, and only including the header if it hasn't been included  
 * once before.  
 */  
#ifndef MY_FUNCTION_H  
#define MY_FUNCTION_H  
  
// global_value and my_function() will be  
// recognized as the same constructs if this header is included by different files.  
const int global_value = 42;  
int my_function();  
  
#endif // MY_FUNCTION_H
```

---

- my\_function.h

```
// my_function.h  
  
/* Note how this header contains only a declaration of a function.  
 * Header functions usually do not define implementations for declarations  
 * unless code must be further processed at compile time, as in templates.  
 */  
  
/* Also, usually header files include preprocessor guards so that every header  
 * is never included twice.  
 *  
 * The guard is implemented by checking if a header-file unique preprocessor  
 * token is defined, and only including the header if it hasn't been included  
 * once before.  
 */  
#ifndef MY_FUNCTION_H  
#define MY_FUNCTION_H  
  
// global_value and my_function() will be  
// recognized as the same constructs if this header is included by different files.  
const int global_value = 42;  
int my_function();  
  
#endif // MY_FUNCTION_H
```

---

---

- /◦

```
// my_function.h  
  
/* Note how this header contains only a declaration of a function.  
 * Header functions usually do not define implementations for declarations  
 * unless code must be further processed at compile time, as in templates.
```

```

*/
/* Also, usually header files include preprocessor guards so that every header
 * is never included twice.
 */
/* The guard is implemented by checking if a header-file unique preprocessor
 * token is defined, and only including the header if it hasn't been included
 * once before.
 */
#ifndef MY_FUNCTION_H
#define MY_FUNCTION_H

// global_value and my_function() will be
// recognized as the same constructs if this header is included by different files.
const int global_value = 42;
int my_function();

#endif // MY_FUNCTION_H

```

main.cpp

```

// my_function.h

/* Note how this header contains only a declaration of a function.
 * Header functions usually do not define implementations for declarations
 * unless code must be further processed at compile time, as in templates.
 */

/* Also, usually header files include preprocessor guards so that every header
 * is never included twice.
 */
/* The guard is implemented by checking if a header-file unique preprocessor
 * token is defined, and only including the header if it hasn't been included
 * once before.
 */
#ifndef MY_FUNCTION_H
#define MY_FUNCTION_H

// global_value and my_function() will be
// recognized as the same constructs if this header is included by different files.
const int global_value = 42;
int my_function();

#endif // MY_FUNCTION_H

```

- 
- 
- 

```

// templated_function.h

template <typename T>
T* null_T_pointer() {
    T* type_point = NULL; // or, alternatively, nullptr instead of NULL
    // for C++11 or later
    return type_point;

```

}

<https://riptutorial.com/zh-TW/cplusplus/topic/7211/>

# 142: /

- variable.member\_var = constant;
- variable.member\_function;
- variable\_pointer-> member\_var = constant;
- variable\_pointer-> member\_function;

struct class struct public class private。 C ++。 C ++。

## Examples

- class struct union class 。 “”。
- “”
- “”
- **typedef“”**
- 

class struct class class “” struct union class “public” 。

```
struct Vector
{
    int x;
    int y;
    int z;
};
// are equivalent to
class Vector
{
public:
    int x;
    int y;
    int z;
};
```

```
struct Vector
{
    int x;
    int y;
    int z;
};
// are equivalent to
class Vector
{
public:
    int x;
    int y;
    int z;
};
```

◦

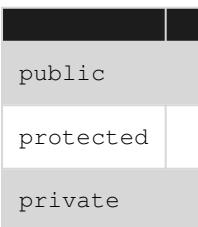
```

struct Vector
{
    int x;
    int y;
    int z;
};

// are equivalent to
class Vector
{
public:
    int x;
    int y;
    int z;
};

```

◦



```

classprivate structpublic

struct MyStruct { int x; };
class MyClass { int x; };

MyStruct s;
s.x = 9; // well formed, because x is public

MyClass c;
c.x = 9; // ill-formed, because x is private

```

## gettersetter

```

struct MyStruct { int x; };
class MyClass { int x; };

MyStruct s;
s.x = 9; // well formed, because x is public

MyClass c;
c.x = 9; // ill-formed, because x is private

```

protectedcalculateValue()Plus2BasePlus2Base FortyTwo

```

struct MyStruct { int x; };
class MyClass { int x; };

MyStruct s;
s.x = 9; // well formed, because x is public

MyClass c;
c.x = 9; // ill-formed, because x is private

```

```
friend.  
public protectedprivate. .  
/  
/B/A BA B/A A/.
```

```
struct A
{
public:
    int p1;
protected:
    int p2;
private:
    int p3;
};

//Make B inherit publicly (default) from A
struct B : A
{
};
```

/3

- public
  - private
  - protected

```
publicstructprivateclass.  
classstruct . structclassclassstruct.
```

```
struct A
{
public:
    int p1;
protected:
    int p2;
private:
    int p3;
};

//Make B inherit publicly (default) from A
struct B : A
{
};
```

private

```
struct A  
{  
public:  
    int p1;
```

```
protected:  
    int p2;  
private:  
    int p3;  
};  
  
//Make B inherit publicly (default) from A  
struct B : A  
{  
};
```

protected

```
struct A
{
public:
    int p1;
protected:
    int p2;
private:
    int p3;
};

//Make B inherit publicly (default) from A
struct B : A
{
};
```

protected◦ protected”◦

## OOP “IS-A” . . .

Liskov /°

““HAS-A”。 StackVector。◦

7

virtual

```
struct A{};  
struct B: public virtual A{};
```

BAA

```
struct A{};  
struct B: public virtual A{}
```

/\*A(88)\*/CA

object C C A A::member88 5 B8

8

```
struct A{};  
struct B: public virtual A{};
```

BCA DBC **D2A** AD BC °

DA°

```
struct A{};  
struct B: public virtual A{};
```

°

```
class A {};  
class B : public A {};
```

```
class A {};  
class B : public A {};
```

CAB°

**class** **struct** °

°

° °

```
class A {};  
class B : public A {};
```

**base1****base2**

```
class A {};  
class B : public A {};
```

```
.  
  
struct SomeStruct {  
    int a;  
    int b;  
    void foo() {}  
};  
  
SomeStruct var;  
// Accessing member variable a in var.  
std::cout << var.a << std::endl;  
// Assigning member variable b in var.  
var.b = 1;  
// Calling a member function.  
var.foo();
```

->° .

```
struct SomeStruct {  
    int a;  
    int b;  
    void foo() {}  
};  
  
SomeStruct var;  
// Accessing member variable a in var.  
std::cout << var.a << std::endl;  
// Assigning member variable b in var.  
var.b = 1;  
// Calling a member function.  
var.foo();
```

::◦ .->◦

```
struct SomeStruct {  
    int a;  
    int b;  
    void foo() {}  
};  
  
SomeStruct var;  
// Accessing member variable a in var.  
std::cout << var.a << std::endl;  
// Assigning member variable b in var.  
var.b = 1;  
// Calling a member function.  
var.foo();
```

->.\*◦

\*pap pa◦ pa◦ \*pa\*(pa)◦ pa◦ a◦

(\*p).a

->◦◦ (\*p).ap->a◦

::◦◦ .->◦◦

```
class A {  
public:  
    int move();  
    int turn();  
};  
  
class B : private A {  
public:  
    using A::turn;  
};  
  
B b;  
b.move(); // compile error  
b.turn(); // OK
```

AA

```
class A {
public:
    int move();
    int turn();
};

class B : private A {
public:
    using A::turn;
};

B b;
b.move(); // compile error
b.turn(); // OK
```

## AB

```
class A {
public:
    int move();
    int turn();
};

class B : private A {
public:
    using A::turn;
};

B b;
b.move(); // compile error
b.turn(); // OK
```

```
class A {
public:
    int move();
    int turn();
};

class B : private A {
public:
    using A::turn;
};

B b;
b.move(); // compile error
b.turn(); // OK
```

```
class A {
public:
    int move();
    int turn();
};

class B : private A {
public:
    using A::turn;
};
```

```
B b;  
b.move(); // compile error  
b.turn(); // OK
```

## C++ 11

final.

```
class A final {  
};
```

```
class A final {  
};
```

```
class A final {  
};
```

friend.

```
class Animal{  
private:  
    double weight;  
    double height;  
public:  
    friend void printWeight(Animal animal);  
    friend class AnimalPrinter;  
    // A common use for a friend function is to overload the operator<< for streaming.  
    friend std::ostream& operator<<(std::ostream& os, Animal animal);  
};  
  
void printWeight(Animal animal)  
{  
    std::cout << animal.weight << "\n";  
}  
  
class AnimalPrinter  
{  
public:  
    void print(const Animal& animal)  
    {  
        // Because of the `friend class AnimalPrinter;` declaration, we are  
        // allowed to access private members here.  
        std::cout << animal.weight << ", " << animal.height << std::endl;  
    }  
}  
  
std::ostream& operator<<(std::ostream& os, Animal animal)  
{  
    os << "Animal height: " << animal.height << "\n";  
    return os;  
}  
  
int main() {  
    Animal animal = {10, 5};  
    printWeight(animal);
```

```
    AnimalPrinter aPrinter;
    aPrinter.print(animal);

    std::cout << animal;
}
```

```
class Animal{
private:
    double weight;
    double height;
public:
    friend void printWeight(Animal animal);
    friend class AnimalPrinter;
    // A common use for a friend function is to overload the operator<< for streaming.
    friend std::ostream& operator<<(std::ostream& os, Animal animal);
};

void printWeight(Animal animal)
{
    std::cout << animal.weight << "\n";
}

class AnimalPrinter
{
public:
    void print(const Animal& animal)
    {
        // Because of the `friend class AnimalPrinter;` declaration, we are
        // allowed to access private members here.
        std::cout << animal.weight << ", " << animal.height << std::endl;
    }
}

std::ostream& operator<<(std::ostream& os, Animal animal)
{
    os << "Animal height: " << animal.height << "\n";
    return os;
}

int main() {
    Animal animal = {10, 5};
    printWeight(animal);

    AnimalPrinter aPrinter;
    aPrinter.print(animal);

    std::cout << animal;
}
```

/

class struct class / struct " " " " . .

```
struct Outer {
    struct Inner { };
};
```

◦

```
struct Outer {  
    struct Inner { };  
};
```

◦ class / struct ◦

```
struct Outer {  
    struct Inner { };  
};
```

◦

```
struct Outer {  
    struct Inner { };  
};
```

---

## C++ 11

C++ 11 static;

C++ 11

C++ 11 friend;

```
struct Outer {  
    struct Inner { };  
};
```

◦

```
struct Outer {  
    struct Inner { };  
};
```

;◦ ◦

```
struct Outer {  
    struct Inner { };  
};
```

---

◦ ◦

```
struct Outer {  
    struct Inner { };  
};
```

◦ ◦ typedef◦

```
struct Outer {  
    struct Inner { };  
};
```

◦

```
struct Outer {  
    struct Inner { };  
};
```

- **typedef**

```
struct Outer {  
    struct Inner { };  
};
```

BaseOuterDerivedOuterInner BaseInner\_DerivedInner\_ ◦ ◦

class struct ◦

```
struct IHaveATypedef {  
    typedef int MyTypedef;  
};  
  
struct IHaveATemplateTypedef {  
    template<typename T>  
    using MyTemplateTypedef = std::vector<T>;  
};
```

◦ **typedef** ◦

```
struct IHaveATypedef {  
    typedef int MyTypedef;  
};  
  
struct IHaveATemplateTypedef {  
    template<typename T>  
    using MyTemplateTypedef = std::vector<T>;  
};
```

- **typedef** **typedef** ◦

```
struct IHaveATypedef {  
    typedef int MyTypedef;  
};  
  
struct IHaveATemplateTypedef {  
    template<typename T>  
    using MyTemplateTypedef = std::vector<T>;  
};
```

◦

```
struct IHaveATypedef {
    typedef int MyTypedef;
};

struct IHaveATemplateTypedef {
    template<typename T>
    using MyTemplateTypedef = std::vector<T>;
};
```

◦

```
struct IHaveATypedef {
    typedef int MyTypedef;
};

struct IHaveATemplateTypedef {
    template<typename T>
    using MyTemplateTypedef = std::vector<T>;
};
```

SomeComplexType`typedef friend Something::MyHelper`◦

`decltype`◦

```
struct IHaveATypedef {
    typedef int MyTypedef;
};

struct IHaveATemplateTypedef {
    template<typename T>
    using MyTemplateTypedef = std::vector<T>;
};
```

`decltype SomethingElse::helper``typedef`◦ `helper`◦

◦ `typename`◦ `typedef`◦ ◦

◦

```
struct IHaveATypedef {
    typedef int MyTypedef;
};

struct IHaveATemplateTypedef {
    template<typename T>
    using MyTemplateTypedef = std::vector<T>;
};
```

◦ **C ++12**◦

```
struct IHaveATypedef {
    typedef int MyTypedef;
```

```

};

struct IHaveATemplateTypedef {
    template<typename T>
    using MyTemplateTypedef = std::vector<T>;
};

```

## C++ 11 `typedef "";` type

```

struct IHaveATypedef {
    typedef int MyTypedef;
};

struct IHaveATemplateTypedef {
    template<typename T>
    using MyTemplateTypedef = std::vector<T>;
};

```

◦

```

struct IHaveATypedef {
    typedef int MyTypedef;
};

struct IHaveATemplateTypedef {
    template<typename T>
    using MyTemplateTypedef = std::vector<T>;
};

```

`static` ;◦

```

class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str; // Static data member (static member variable).
    static int static_func();      // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.';

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)

```

```

// (&two.num_instances == &three.num_instances)

;° static °

class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str;    // Static data member (static member variable).
    static int static_func();         // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.';

// ...

```

Example one, two, three;  
// Each Example has its own "i", such that:  
// (&one.i != &two.i)  
// (&one.i != &three.i)  
// (&two.i != &three.i).  
// All three Examples share "num\_instances", such that:  
// (&one.num\_instances == &two.num\_instances)  
// (&one.num\_instances == &three.num\_instances)  
// (&two.num\_instances == &three.num\_instances)

```

void °

class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str;    // Static data member (static member variable).
    static int static_func();         // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.';

// ...

```

Example one, two, three;  
// Each Example has its own "i", such that:  
// (&one.i != &two.i)  
// (&one.i != &three.i)  
// (&two.i != &three.i).  
// All three Examples share "num\_instances", such that:

```
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)
```

◦ static ◦

```
class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str;    // Static data member (static member variable).
    static int static_func();         // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.';

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)
```

---

constvolatile ◦

```
class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str;    // Static data member (static member variable).
    static int static_func();         // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.';

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
```

```
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)
```

## C++ 11

### C++ 11 LiteralTypeconstexprconstexpr;

```
class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str;    // Static data member (static member variable).
    static int static_func();         // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.';

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)
```

### constconstexprodr . .

```
class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str;    // Static data member (static member variable).
    static int static_func();         // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.';

// ...
```

```
Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)
```

:::

```
class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str; // Static data member (static member variable).
    static int static_func();      // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.";

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)
```

◦ ◦

```
class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str; // Static data member (static member variable).
    static int static_func();      // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.";
```

```
// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)
```

◦

```
class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str;    // Static data member (static member variable).
    static int static_func();         // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.';

// ...
```

Example one, two, three;

// Each Example has its own "i", such that:

// (&one.i != &two.i)  
// (&one.i != &three.i)  
// (&two.i != &three.i).

// All three Examples share "num\_instances", such that:

// (&one.num\_instances == &two.num\_instances)  
// (&one.num\_instances == &three.num\_instances)  
// (&two.num\_instances == &three.num\_instances)

**mutable ;const◦**

```
class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str;    // Static data member (static member variable).
    static int static_func();         // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
```

```
std::string Example::static_str = "Hello.";  
  
// ...  
  
Example one, two, three;  
// Each Example has its own "i", such that:  
// (&one.i != &two.i)  
// (&one.i != &three.i)  
// (&two.i != &three.i).  
// All three Examples share "num_instances", such that:  
// (&one.num_instances == &two.num_instances)  
// (&one.num_instances == &three.num_instances)  
// (&two.num_instances == &three.num_instances)
```

◦

```
class Example {  
    static int num_instances;           // Static data member (static member variable).  
    int i;                            // Non-static member variable.  
  
public:  
    static std::string static_str;     // Static data member (static member variable).  
    static int static_func();         // Static member function.  
  
    // Non-static member functions can modify static member variables.  
    Example() { ++num_instances; }  
    void set_str(const std::string& str);  
};  
  
int      Example::num_instances;  
std::string Example::static_str = "Hello."  
  
// ...  
  
Example one, two, three;  
// Each Example has its own "i", such that:  
// (&one.i != &two.i)  
// (&one.i != &three.i)  
// (&two.i != &three.i).  
// All three Examples share "num_instances", such that:  
// (&one.num_instances == &two.num_instances)  
// (&one.num_instances == &three.num_instances)  
// (&two.num_instances == &three.num_instances)
```

this◦

```
class Example {  
    static int num_instances;           // Static data member (static member variable).  
    int i;                            // Non-static member variable.  
  
public:  
    static std::string static_str;     // Static data member (static member variable).  
    static int static_func();         // Static member function.  
  
    // Non-static member functions can modify static member variables.  
    Example() { ++num_instances; }  
    void set_str(const std::string& str);
```

```

};

int      Example::num_instances;
std::string Example::static_str = "Hello.";

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)

```

this.

```

class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str; // Static data member (static member variable).
    static int static_func();     // Static member function.

    // Non-static member functions can modify static member variables.
    Example() { ++num_instances; }
    void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.";

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)

```

thisconstvolatile **ref-qualifiers**。

```

class Example {
    static int num_instances;           // Static data member (static member variable).
    int i;                            // Non-static member variable.

public:
    static std::string static_str; // Static data member (static member variable).
    static int static_func();     // Static member function.

    // Non-static member functions can modify static member variables.

```

```

Example() { ++num_instances; }
void set_str(const std::string& str);
};

int      Example::num_instances;
std::string Example::static_str = "Hello.";

// ...

Example one, two, three;
// Each Example has its own "i", such that:
// (&one.i != &two.i)
// (&one.i != &three.i)
// (&two.i != &three.i).
// All three Examples share "num_instances", such that:
// (&one.num_instances == &two.num_instances)
// (&one.num_instances == &three.num_instances)
// (&two.num_instances == &three.num_instances)

```

### ;◦ thread\_local C++11◦

◦ ◦  
◦

```

class CL {
public:
    void member_function() {}
};

```

```

class CL {
public:
    void member_function() {}
};

```

;◦

```

class CL {
public:
    void member_function() {}
};

```

### CV/ref-qualified ;cv-qualifier◦ cv◦ cvcv◦

```

class CL {
public:
    void member_function() {}
};

```

### C++11

ref-qualifiersvaluecv-qualifiers◦

```
class CL {  
public:  
    void member_function() {}  
};
```

## CVref。

```
class CL {  
public:  
    void member_function() {}  
};
```

;◦

```
class CL {  
public:  
    void member_function() {}  
};
```

◦

/

*struct*

```
void foo()  
{  
    struct /* No name */ {  
        float x;  
        float y;  
    } point;  
  
    point.x = 42;  
}
```

```
void foo()  
{  
    struct /* No name */ {  
        float x;  
        float y;  
    } point;  
  
    point.x = 42;  
}
```

```
void foo()  
{  
    struct /* No name */ {  
        float x;  
        float y;  
    } point;  
  
    point.x = 42;  
}
```

*struct*

```
void foo()
{
    struct /* No name */ {
        float x;
        float y;
    } point;
    point.x = 42;
}
```

*struct*。

## C++ 11

- *lambda**struct*。
- *decltype**struct*

```
void foo()
{
    struct /* No name */ {
        float x;
        float y;
    } point;
    point.x = 42;
}
```

- *struct*

```
void foo()
{
    struct /* No name */ {
        float x;
        float y;
    } point;
    point.x = 42;
}
```

/ <https://riptutorial.com/zh-TW/cplusplus/topic/508/>

# 143:

201411C ++N3922。 C ++。

## Examples

C ++ 17。 。 lambdas`make_pair() make_tuple() back_inserter()`。

C ++ 17

```
std::pair p(2, 4.5);      // std::pair<int, double>
std::tuple t(4, 3, 2.5); // std::tuple<int, int, double>
std::copy_n(vil.begin(), 3,
    std::back_insert_iterator(vi2)); // constructs a back_insert_iterator<std::vector<int>>
std::lock_guard lk(mtx); // std::lock_guard<decltype(mtx)>
```

```
std::pair p(2, 4.5);      // std::pair<int, double>
std::tuple t(4, 3, 2.5); // std::tuple<int, int, double>
std::copy_n(vil.begin(), 3,
    std::back_insert_iterator(vi2)); // constructs a back_insert_iterator<std::vector<int>>
std::lock_guard lk(mtx); // std::lock_guard<decltype(mtx)>
```

```
template<typename T>
void f(ParamType param);

f(expr);
```

1 ParamType。 。 expr。 exprParamTypeT

```
template<typename T>
void f(ParamType param);

f(expr);
```

2 ParamType。 expr**rvalue1**。 exprTParamType。

```
template<typename T>
void f(ParamType param);

f(expr);
```

3 ParamType。 expr。 expr**const**。 volatileT。

```
template<typename T>
void f(ParamType param);

f(expr);
```

C ++ 11

```
auto °
```

```
auto x = 27;           // (x is neither a pointer nor a reference), x's type is int
const auto cx = x;    // (cx is neither a pointer nor a reference), cs's type is const int
const auto& rx = x;   // (rx is a non-universal reference), rx's type is a reference to a
const int

auto&& uref1 = x;     // x is int and lvalue, so uref1's type is int&
auto&& uref2 = cx;    // cx is const int and lvalue, so uref2's type is const int &
auto&& uref3 = 27;    // 27 is an int and rvalue, so uref3's type is int&&
```

```
auto x = 27;           // (x is neither a pointer nor a reference), x's type is int
const auto cx = x;    // (cx is neither a pointer nor a reference), cs's type is const int
const auto& rx = x;   // (rx is a non-universal reference), rx's type is a reference to a
const int

auto&& uref1 = x;     // x is int and lvalue, so uref1's type is int&
auto&& uref2 = cx;    // cx is const int and lvalue, so uref2's type is const int &
auto&& uref3 = 27;    // 27 is an int and rvalue, so uref3's type is int&&
```

```
auto std::initializer_list<T>° T °
```

```
auto °
```

```
auto x = 27;           // (x is neither a pointer nor a reference), x's type is int
const auto cx = x;    // (cx is neither a pointer nor a reference), cs's type is const int
const auto& rx = x;   // (rx is a non-universal reference), rx's type is a reference to a
const int

auto&& uref1 = x;     // x is int and lvalue, so uref1's type is int&
auto&& uref2 = cx;    // cx is const int and lvalue, so uref2's type is const int &
auto&& uref3 = 27;    // 27 is an int and rvalue, so uref3's type is int&&
```

## C++ 14

### C++ 11 C++ 14

#### 1. return°

```
auto x = 27;           // (x is neither a pointer nor a reference), x's type is int
const auto cx = x;    // (cx is neither a pointer nor a reference), cs's type is const
int
const auto& rx = x;   // (rx is a non-universal reference), rx's type is a reference to
a const int

auto&& uref1 = x;     // x is int and lvalue, so uref1's type is int&
auto&& uref2 = cx;    // cx is const int and lvalue, so uref2's type is const int &
auto&& uref3 = 27;    // 27 is an int and rvalue, so uref3's type is int&&
```

#### 2. lambda lambda lambda °

```
auto x = 27;           // (x is neither a pointer nor a reference), x's type is int
const auto cx = x;    // (cx is neither a pointer nor a reference), cs's type is const
int
```

```
const auto& rx = x;      // (rx is a non-universal reference), rx's type is a reference to  
a const int  
  
auto&& uref1 = x;       // x is int and lvalue, so uref1's type is int&  
auto&& uref2 = cx;       // cx is const int and lvalue, so uref2's type is const int &  
auto&& uref3 = 27;        // 27 is an int and rvalue, so uref3's type is int&&
```

```
decltype(auto) decltypeauto::
```

```
auto x = 27;            // (x is neither a pointer nor a reference), x's type is int  
const auto cx = x;      // (cx is neither a pointer nor a reference), cs's type is const int  
const auto& rx = x;      // (rx is a non-universal reference), rx's type is a reference to a  
const int  
  
auto&& uref1 = x;        // x is int and lvalue, so uref1's type is int&  
auto&& uref2 = cx;        // cx is const int and lvalue, so uref2's type is const int &  
auto&& uref3 = 27;         // 27 is an int and rvalue, so uref3's type is int&&
```

C++ 03 autoC.

<https://riptutorial.com/zh-TW/cplusplus/topic/7863/>

# 144:

C++ 11 `auto`.

`auto const &constexpr`  $\circ$  `auto STL`.

## Examples

$\circ$

```
auto a = 1;           //      a = int
auto b = 2u;          //      b = unsigned int
auto c = &a;           //      c = int*
const auto d = c;    //      d = const int*
const auto& e = b;   //      e = const unsigned int&

auto x = a + b       //      x = int, #compiler warning unsigned and signed

auto v = std::vector<int>;    //      v = std::vector<int>
```

`&constconstexprauto`

```
auto a = 1;           //      a = int
auto b = 2u;          //      b = unsigned int
auto c = &a;           //      c = int*
const auto d = c;    //      d = const int*
const auto& e = b;   //      e = const unsigned int&

auto x = a + b       //      x = int, #compiler warning unsigned and signed

auto v = std::vector<int>;    //      v = std::vector<int>
```

## Lambda

`autolambda`  $\circ$

```
auto DoThis = [](int a, int b) { return a + b; };
//      Do this is of type (int)(*DoThis)(int, int)
//      else we would have to write this long
int(*pDoThis)(int, int)= [](int a, int b) { return a + b; };

auto c = Dothis(1, 2);    //      c = int
auto d = pDothis(1, 2);  //      d = int

//      using 'auto' shortens the definition for lambda functions
```

`lambda`.

```
auto DoThis = [](int a, int b) { return a + b; };
//      Do this is of type (int)(*DoThis)(int, int)
//      else we would have to write this long
```

```
int(*pDoThis)(int, int)= [](int a, int b) { return a + b; };

auto c = Dothis(1, 2);      //      c = int
auto d = pDothis(1, 2);    //      d = int

//      using 'auto' shortens the definition for lambda functions
```

## autofor

```
std::map<int, std::string> Map;
for (auto pair : Map)           //      pair = std::pair<int, std::string>
for (const auto pair : Map)     //      pair = const std::pair<int, std::string>
for (const auto& pair : Map)    //      pair = const std::pair<int, std::string>&
for (auto i = 0; i < 1000; ++i) //      i = int
for (auto i = 0; i < Map.size(); ++i) //      Note that i = int and not size_t
for (auto i = Map.size(); i > 0; --i) //      i = size_t
```

<https://riptutorial.com/zh-TW/cplusplus/topic/8233/>

**145:**

- `std::function<R(A...)>` C++◦

## Examples

- ○

```
#include <iostream>

class Printable
{
public:
    template <typename T>
    Printable(T value) : pValue(new Value<T>(value)) {}
    ~Printable() { delete pValue; }
    void print(std::ostream &os) const { pValue->print(os); }

private:
    Printable(Printable const &) /* in C++1x: =delete */; // not implemented
    void operator = (Printable const &) /* in C++1x: =delete */; // not implemented
    struct ValueBase
    {
        virtual ~ValueBase() = default;
        virtual void print(std::ostream &) const = 0;
    };
    template <typename T>
    struct Value : ValueBase
    {
        Value(T const &t) : v(t) {}
        virtual void print(std::ostream &os) const { os << v; }
        T v;
    };
    ValueBase *pValue;
};
```

```
#include <iostream>

class Printable
{
public:
    template <typename T>
    Printable(T value) : pValue(new Value<T>(value)) {}
    ~Printable() { delete pValue; }
    void print(std::ostream &os) const { pValue->print(os); }

private:
    Printable(Printable const &)           /* in C++1x: =delete */; // not implemented
    void operator = (Printable const &) /* in C++1x: =delete */; // not implemented
    struct ValueBase
```

```

{
    virtual ~ValueBase() = default;
    virtual void print(std::ostream &) const = 0;
};

template <typename T>
struct Value : ValueBase
{
    Value(T const &t) : v(t) {}
    virtual void print(std::ostream &os) const { os << v; }
    T v;
};

ValueBase *pValue;
};

```

◦

### Printable

```

#include <iostream>

class Printable
{
public:
    template <typename T>
    Printable(T value) : pValue(new Value<T>(value)) {}
    ~Printable() { delete pValue; }
    void print(std::ostream &os) const { pValue->print(os); }

private:
    Printable(Printable const &) /* in C++1x: =delete */; // not implemented
    void operator = (Printable const &) /* in C++1x: =delete */; // not implemented
    struct ValueBase
    {
        virtual ~ValueBase() = default;
        virtual void print(std::ostream &) const = 0;
    };
    template <typename T>
    struct Value : ValueBase
    {
        Value(T const &t) : v(t) {}
        virtual void print(std::ostream &os) const { os << v; }
        T v;
    };
    ValueBase *pValue;
};

```

```

#include <iostream>

class Printable
{
public:
    template <typename T>
    Printable(T value) : pValue(new Value<T>(value)) {}
    ~Printable() { delete pValue; }
    void print(std::ostream &os) const { pValue->print(os); }

private:
    Printable(Printable const &) /* in C++1x: =delete */; // not implemented
    void operator = (Printable const &) /* in C++1x: =delete */; // not implemented

```

```

struct ValueBase
{
    virtual ~ValueBase() = default;
    virtual void print(std::ostream &) const = 0;
};

template <typename T>
struct Value : ValueBase
{
    Value(T const &t) : v(t) {}
    virtual void print(std::ostream &os) const { os << v; }
    T v;
};

ValueBase *pValue;
};

```

## vtable

### C++ Regular Pseudo-Regular

- - o . std::

### C++ 11 std::hash

## vtable

```

using dtor_unique_ptr = std::unique_ptr<void, void(*)(void*)>;
template<class T, class...Args>
dtor_unique_ptr make_dtor_unique_ptr(Args&&... args) {
    return {new T(std::forward<Args>(args)...), [](void* self){ delete static_cast<T*>(self); }};
}

struct regular_vtable {
    void(*copy_assign)(void* dest, void const* src); // T&=(T const&)
    void(*move_assign)(void* dest, void* src); // T&=(T&&)
    bool(*equals)(void const* lhs, void const* rhs); // T const&==T const&
    bool(*order)(void const* lhs, void const* rhs); // std::less<T>{}(T const&, T const&)
    std::size_t(*hash)(void const* self); // std::hash<T>{}(T const&)
    std::type_info const&(*type)(); // typeid(T)
    dtor_unique_ptr(*clone)(void const* self); // T(T const&)
};

template<class T>
regular_vtable make_regular_vtable() noexcept {
    return {
        [] (void* dest, void const* src){ *static_cast<T*>(dest) = *static_cast<T const*>(src); },
        [] (void* dest, void* src){ *static_cast<T*>(dest) = std::move(*static_cast<T*>(src)); },
        [] (void const* lhs, void const* rhs){ return *static_cast<T const*>(lhs) == *static_cast<T const*>(rhs); },
        [] (void const* lhs, void const* rhs){ return std::less<T>{}(*static_cast<T const*>(lhs), *static_cast<T const*>(rhs)); },
        [] (void const* self){ return std::hash<T>{}(*static_cast<T const*>(self)); },
        [] ()->decltype(auto){ return typeid(T); },
        [] (void const* self){ return make_dtor_unique_ptr<T>(*static_cast<T const*>(self)); }
    };
}

template<class T>
regular_vtable const* get_regular_vtable() noexcept {
    static const regular_vtable vtable=make_regular_vtable<T>();
}

```

```

    return &vtable;
}

struct regular_type {
    using self=regular_type;
    regular_vtable const* vtable = 0;
    dtor_unique_ptr ptr{nullptr, [](void*){}};

    bool empty() const { return !vtable; }

    template<class T, class...Args>
    void emplace(Args&&... args) {
        ptr = make_dtor_unique_ptr<T>(std::forward<Args>(args)...);
        if (ptr)
            vtable = get_regular_vtable<T>();
        else
            vtable = nullptr;
    }
    friend bool operator==(regular_type const& lhs, regular_type const& rhs) {
        if (lhs.vtable != rhs.vtable) return false;
        return lhs.vtable->equals( lhs.ptr.get(), rhs.ptr.get() );
    }
    bool before(regular_type const& rhs) const {
        auto const& lhs = *this;
        if (!lhs.vtable || !rhs.vtable)
            return std::less<regular_vtable const*>{}(lhs.vtable,rhs.vtable);
        if (lhs.vtable != rhs.vtable)
            return lhs.vtable->type().before(rhs.vtable->type());
        return lhs.vtable->order( lhs.ptr.get(), rhs.ptr.get() );
    }
    // technically friend bool operator< that calls before is also required

    std::type_info const* type() const {
        if (!vtable) return nullptr;
        return &vtable->type();
    }
    regular_type(regular_type&& o):
        vtable(o.vtable),
        ptr(std::move(o.ptr))
    {
        o.vtable = nullptr;
    }
    friend void swap(regular_type& lhs, regular_type& rhs){
        std::swap(lhs.ptr, rhs.ptr);
        std::swap(lhs.vtable, rhs.vtable);
    }
    regular_type& operator=(regular_type&& o) {
        if (o.vtable == vtable) {
            vtable->move_assign(ptr.get(), o.ptr.get());
            return *this;
        }
        auto tmp = std::move(o);
        swap(*this, tmp);
        return *this;
    }
    regular_type(regular_type const& o):
        vtable(o.vtable),
        ptr(o.vtable?o.vtable->clone(o.ptr.get()):dtor_unique_ptr{nullptr, [](void*){}})
    {
        if (!ptr && vtable) vtable = nullptr;
    }
}

```

```

regular_type& operator=(regular_type const& o) {
    if (o.vtable == vtable) {
        vtable->copy_assign(ptr.get(), o.ptr.get());
        return *this;
    }
    auto tmp = o;
    swap(*this, tmp);
    return *this;
}
std::size_t hash() const {
    if (!vtable) return 0;
    return vtable->hash(ptr.get());
}
template<class T,
         std::enable_if_t<!std::is_same<std::decay_t<T>, regular_type>{}, int>* =nullptr
>
regular_type(T&& t) {
    emplace<std::decay_t<T>>(std::forward<T>(t));
}
};

namespace std {
template<>
struct hash<regular_type> {
    std::size_t operator()( regular_type const& r )const {
        return r.hash();
    }
};
template<>
struct less<regular_type> {
    bool operator()( regular_type const& lhs, regular_type const& rhs ) const {
        return lhs.before(rhs);
    }
};
}
}

```

◦

`std::map<std::unordered_map`

```

using dtor_unique_ptr = std::unique_ptr<void, void(*)(void*)>;
template<class T, class...Args>
dtor_unique_ptr make_dtor_unique_ptr(Args&&... args) {
    return {new T(std::forward<Args>(args)...), [](void* self){ delete static_cast<T*>(self); }};
}
struct regular_vtable {
    void(*copy_assign)(void* dest, void const* src); // T&=(T const&)
    void(*move_assign)(void* dest, void* src); // T&=(T&&)
    bool(*equals)(void const* lhs, void const* rhs); // T const&==T const&
    bool(*order)(void const* lhs, void const* rhs); // std::less<T>{}(T const&, T const&)
    std::size_t(*hash)(void const* self); // std::hash<T>{}(T const&)
    std::type_info const&(*type)(); // typeid(T)
    dtor_unique_ptr(*clone)(void const* self); // T(T const&)
};

template<class T>
regular_vtable make_regular_vtable() noexcept {
    return {
        [] (void* dest, void const* src){ *static_cast<T*>(dest) = *static_cast<T const*>(src); },
        [] (void* dest, void* src){ *static_cast<T*>(dest) = std::move(*static_cast<T*>(src)); },
    };
}

```

```

    [](void const* lhs, void const* rhs){ return *static_cast<T const*>(lhs) == *static_cast<T
const*>(rhs); },
    [](void const* lhs, void const* rhs) { return std::less<T>{}(*static_cast<T
const*>(lhs),*static_cast<T const*>(rhs)); },
    [](void const* self){ return std::hash<T>{}(*static_cast<T const*>(self)); },
    []()>decltype(auto){ return typeid(T); },
    [](void const* self){ return make_dtor_unique_ptr<T>(*static_cast<T const*>(self)); }
};

}

template<class T>
regular_vtable const* get_regular_vtable() noexcept {
    static const regular_vtable vtable=make_regular_vtable<T>();
    return &vtable;
}

struct regular_type {
    using self=regular_type;
    regular_vtable const* vtable = 0;
    dtor_unique_ptr ptr{nullptr, [](void*){}};

    bool empty() const { return !vtable; }

    template<class T, class...Args>
    void emplace(Args&&... args) {
        ptr = make_dtor_unique_ptr<T>(std::forward<Args>(args)...);
        if (ptr)
            vtable = get_regular_vtable<T>();
        else
            vtable = nullptr;
    }
    friend bool operator==(regular_type const& lhs, regular_type const& rhs) {
        if (lhs.vtable != rhs.vtable) return false;
        return lhs.vtable->equals( lhs.ptr.get(), rhs.ptr.get() );
    }
    bool before(regular_type const& rhs) const {
        auto const& lhs = *this;
        if (!lhs.vtable || !rhs.vtable)
            return std::less<regular_vtable const*>{}(lhs.vtable,rhs.vtable);
        if (lhs.vtable != rhs.vtable)
            return lhs.vtable->type().before(rhs.vtable->type());
        return lhs.vtable->order( lhs.ptr.get(), rhs.ptr.get() );
    }
    // technically friend bool operator< that calls before is also required

    std::type_info const* type() const {
        if (!vtable) return nullptr;
        return &vtable->type();
    }
    regular_type(regular_type&& o):
        vtable(o.vtable),
        ptr(std::move(o.ptr))
    {
        o.vtable = nullptr;
    }
    friend void swap(regular_type& lhs, regular_type& rhs){
        std::swap(lhs.ptr, rhs.ptr);
        std::swap(lhs.vtable, rhs.vtable);
    }
    regular_type& operator=(regular_type&& o) {
        if (o.vtable == vtable) {
            vtable->move_assign(ptr.get(), o.ptr.get());
        }

```

```

    return *this;
}
auto tmp = std::move(o);
swap(*this, tmp);
return *this;
}
regular_type(regular_type const& o):
    vtable(o.vtable),
    ptr(o.vtable?o.vtable->clone(o.ptr.get()):dtor_unique_ptr{nullptr, [](void*){}})
{
    if (!ptr && vtable) vtable = nullptr;
}
regular_type& operator=(regular_type const& o) {
    if (o.vtable == vtable) {
        vtable->copy_assign(ptr.get(), o.ptr.get());
        return *this;
    }
    auto tmp = o;
    swap(*this, tmp);
    return *this;
}
std::size_t hash() const {
    if (!vtable) return 0;
    return vtable->hash(ptr.get());
}
template<class T,
         std::enable_if_t<!std::is_same<std::decay_t<T>, regular_type>{}, int>* =nullptr
>
regular_type(T& t) {
    emplace<std::decay_t<T>>(std::forward<T>(t));
}
};

namespace std {
    template<>
    struct hash<regular_type> {
        std::size_t operator()( regular_type const& r )const {
            return r.hash();
        }
    };
    template<>
    struct less<regular_type> {
        bool operator()( regular_type const& lhs, regular_type const& rhs ) const {
            return lhs.before(rhs);
        }
    };
}

```

2

any regular type.

regular typept.r°

make dtor unique ptr. °

## ``std :: function``

std::function<>

```

ptrslambda std::function<R(Args...)>

std::packaged_task<Sig> std::packaged_task<void(Args...)>std::packaged_task<R(Args...)>.

task std::function<details::task_pimpl<...>clone>

template<class Sig>
struct task;

// putting it in a namespace allows us to specialize it nicely for void return value:
namespace details {
    template<class R, class...Args>
    struct task_pimpl {
        virtual R invoke(Args&&...args) const = 0;
        virtual ~task_pimpl() {};
        virtual const std::type_info& target_type() const = 0;
    };
}

// store an F. invoke(Args&&...) calls the f
template<class F, class R, class...Args>
struct task_pimpl_impl:task_pimpl<R,Args...> {
    F f;
    template<class Fin>
    task_pimpl_impl( Fin&& fin ):f(std::forward<Fin>(fin)) {}
    virtual R invoke(Args&&...args) const final override {
        return f(std::forward<Args>(args)...);
    }
    virtual const std::type_info& target_type() const final override {
        return typeid(F);
    }
};

// the void version discards the return value of f:
template<class F, class...Args>
struct task_pimpl_impl<F,void,Args...>:task_pimpl<void,Args...> {
    F f;
    template<class Fin>
    task_pimpl_impl( Fin&& fin ):f(std::forward<Fin>(fin)) {}
    virtual void invoke(Args&&...args) const final override {
        f(std::forward<Args>(args)...);
    }
    virtual const std::type_info& target_type() const final override {
        return typeid(F);
    }
};

template<class R, class...Args>
struct task<R(Args...)> {
    // semi-regular:
    task()=default;
    task(task&&)=default;
    // no copy

private:
    // aliases to make some SFINAE code below less ugly:
    template<class F>
    using call_r = std::result_of_t<F const&(Args...)>;
    template<class F>
    using is_task = std::is_same<std::decay_t<F>, task>;

```

```

public:
    // can be constructed from a callable F
    template<class F,
        // that can be invoked with Args... and converted-to-R:
        class= decltype( (R)(std::declval<call_r<F>>() ) ),
        // and is not this same type:
        std::enable_if_t<!is_task<F>{}>, int>* = nullptr
    >
    task(F&& f):
        m_pImpl( make_pimpl(std::forward<F>(f)) )
    {}

    // the meat: the call operator
    R operator()(Args... args) const {
        return m_pImpl->invoke( std::forward<Args>(args)... );
    }
    explicit operator bool() const {
        return (bool)m_pImpl;
    }
    void swap( task& o ) {
        std::swap( m_pImpl, o.m_pImpl );
    }
    template<class F>
    void assign( F&& f ) {
        m_pImpl = make_pimpl(std::forward<F>(f));
    }
    // Part of the std::function interface:
    const std::type_info& target_type() const {
        if (!*this) return typeid(void);
        return m_pImpl->target_type();
    }
    template< class T >
    T* target() {
        return target_impl<T>();
    }
    template< class T >
    const T* target() const {
        return target_impl<T>();
    }
    // compare with nullptr :
    friend bool operator==( std::nullptr_t, task const& self ) { return !self; }
    friend bool operator==( task const& self, std::nullptr_t ) { return !self; }
    friend bool operator!=( std::nullptr_t, task const& self ) { return !!self; }
    friend bool operator!=( task const& self, std::nullptr_t ) { return !!self; }
private:
    template<class T>
    using pimpl_t = details::task_pimpl_impl<T, R, Args...>;
    template<class F>
    static auto make_pimpl( F&& f ) {
        using dF=std::decay_t<F>;
        using pImpl_t = pimpl_t<dF>;
        return std::make_unique<pImpl_t>(std::forward<F>(f));
    }
    std::unique_ptr<details::task_pimpl<R,Args...>> m_pImpl;
    template< class T >
    T* target_impl() const {
        return dynamic_cast<pimpl_t<T>*>(m_pImpl.get());
    }
};

```

1

**SBO**task(task&&) std::aligned\_storage\_t m\_pImpl unique\_ptr emplace\_move\_to( void\* ) = 0 task\_pimpl  
emplace\_move\_to( void\* ) = 0 。

SBO.

T

8

“””””

array\_viewT。

```

// helper traits for SFINAE:
template<class T>
using data_t = decltype( std::declval<T>().data() );

template<class Src, class T>
using compatible_data = std::integral_constant<bool, std::is_same< data_t<Src>, T* >{} || std::is_same< data_t<Src>, std::remove_const_t<T>*>{}>;

template<class T>
struct array_view {
    // the core of the class:
    T* b=nullptr;
    T* e=nullptr;
    T* begin() const { return b; }
    T* end() const { return e; }

    // provide the expected methods of a good contiguous range:
    T* data() const { return begin(); }
    bool empty() const { return begin()==end(); }
    std::size_t size() const { return end()-begin(); }

    T& operator[](std::size_t i) const{ return begin()[i]; }
    T& front() const{ return *begin(); }
    T& back() const{ return *(end()-1); }

    // useful helpers that let you generate other ranges from this one
    // quickly and safely:
    array_view without_front( std::size_t i=1 ) const {
        i = (std::min)(i, size());
        return {begin()+i, end()};
    }
    array_view without_back( std::size_t i=1 ) const {
        i = (std::min)(i, size());
        return {begin(), end()-i};
    }

    // array_view is plain old data, so default copy:
    array_view(array_view const&)=default;
    // generates a null, empty range:
    array_view()=default;

    // final constructor:

```

```

array_view(T* s, T* f):b(s),e(f) {}
// start and length is useful in my experience:
array_view(T* s, std::size_t length):array_view(s, s+length) {}

// SFINAE constructor that takes any .data() supporting container
// or other range in one fell swoop:
template<class Src,
         std::enable_if_t< compatible_data<std::remove_reference_t<Src>&, T >{}>, int>* =nullptr,
         std::enable_if_t< !std::is_same<std::decay_t<Src>, array_view >{}, int>* =nullptr
>
array_view( Src&& src ):
    array_view( src.data(), src.size() )
{ }

// array constructor:
template<std::size_t N>
array_view( T(&arr)[N] ):array_view(arr, N) {}

// initializer list, allowing {} based:
template<class U,
         std::enable_if_t< std::is_same<const U, T>{}, int>* =nullptr
>
array_view( std::initializer_list<U> il ):array_view(il.begin(), il.end()) {}
};

array_view.data() T.size() T S°

```

std::vector<T> std::string<T> std::array<T, N> **a** T[37] {} T\* x.data() size\_t x.size() °

"°.

°

**ADL** datasize °

## std :: any

**C ++ 14** boost::any ° **C ++ 17** std::any °

```

const auto print =
    make_any_method<void(std::ostream&) > ([](auto& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print)(std::cout);

```

°

@dyp@cpplearner °

```

const auto print =
    make_any_method<void(std::ostream&) > ([](auto& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

```

```
(a->*print) (std::cout);
```

## traitany\_method

```
any_method any_method
```

```
const auto print =
make_any_method<void(std::ostream&)>([] (auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print) (std::cout);
```

```
any_method_function::type o any_method_function::operator() tag_t<T>any_method_function::type any& T
o o
```

```
const auto print =
make_any_method<void(std::ostream&)>([] (auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print) (std::cout);
```

## vtable1 o

```
super_any o super_any_tsuper_any o
```

```
const auto print =
make_any_method<void(std::ostream&)>([] (auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print) (std::cout);
```

## super anySFINAE

```
const auto print =
make_any_method<void(std::ostream&)>([] (auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print) (std::cout);
```

```
any_method o any_method o const
```

```
const auto print =
make_any_method<void(std::ostream&)>([] (auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print) (std::cout);
```

C ++ 17

```
const auto print =
    make_any_method<void(std::ostream&)>([](auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print)(std::cout);
```

lambda. . lambda lambda.

```
const auto print =
    make_any_method<void(std::ostream&)>([](auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print)(std::cout);
```

C++17

```
const auto print =
    make_any_method<void(std::ostream&)>([](auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print)(std::cout);
```

any ° any any

```
const auto print =
    make_any_method<void(std::ostream&)>([](auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print)(std::cout);
```

any method const super any

```
const auto print =
    make_any_method<void(std::ostream&)>([](auto&& p, std::ostream& t){ t << p << "\n"; });

super_any<decltype(print)> a = 7;

(a->*print)(std::cout);
```

```
const auto print =
    make_any_method<void(std::ostream&)>([](auto&& p, std::ostream& t){ t << p << "\n"; });
super_any<decltype(print)> a = 7;
```

1

SO。

<https://riptutorial.com/zh-TW/cplusplus/topic/2872/>

146:

○ ○ C++11 ○ ○

## Examples

C++ 11

`type_traits`。

1

1

```
template <class T> struct is_foo;
```

`foois_foo<T>std::integral_constant<bool,true> std::true_type std::integral_constant<bool,false> std::false_type。`

```
static constexpr bool value
```

Tfoo true false

operator bool

value

C ++ 14

```
bool operator()
```

value

|            |                                    |
|------------|------------------------------------|
| value_type | bool                               |
| type       | std::integral_constant<bool,value> |

```
static_assert(std::enable_if_t<std::is_pointer<
```

```
template <class T> struct is_foo;
```

```
std::add_pointer<std::underlying_type<type>>::type* std::add_pointer<int>::type
```

## std :: is\_same

### C ++ 11

std::is\_same<T, T>。 truefalse。

```
// Prints true on most x86 and x86_64 compilers.  
std::cout << std::is_same<int, int32_t>::value << "\n";  
// Prints false on all compilers.  
std::cout << std::is_same<float, int>::value << "\n";  
// Prints false on all compilers.  
std::cout << std::is_same<unsigned int, int>::value << "\n";
```

typedef std::is\_same。 int == int32\_t。

```
// Prints true on most x86 and x86_64 compilers.  
std::cout << std::is_same<int, int32_t>::value << "\n";  
// Prints false on all compilers.  
std::cout << std::is_same<float, int>::value << "\n";  
// Prints false on all compilers.  
std::cout << std::is_same<unsigned int, int>::value << "\n";
```

---

### std::is\_same。

std::is\_same。

int。

```
// Prints true on most x86 and x86_64 compilers.  
std::cout << std::is_same<int, int32_t>::value << "\n";  
// Prints false on all compilers.  
std::cout << std::is_same<float, int>::value << "\n";  
// Prints false on all compilers.  
std::cout << std::is_same<unsigned int, int>::value << "\n";
```

### C ++ 11

。

int char long unsigned int true。

```
std::cout << std::is_integral<int>::value << "\n"; // Prints true.  
std::cout << std::is_integral<char>::value << "\n"; // Prints true.  
std::cout << std::is_integral<float>::value << "\n"; // Prints false.
```

true。 float double long double

```
std::cout << std::is_integral<int>::value << "\n"; // Prints true.  
std::cout << std::is_integral<char>::value << "\n"; // Prints true.  
std::cout << std::is_integral<float>::value << "\n"; // Prints false.
```

## Enum

enum class**true**◦

```
std::cout << std::is_integral<int>::value << "\n"; // Prints true.  
std::cout << std::is_integral<char>::value << "\n"; // Prints true.  
std::cout << std::is_integral<float>::value << "\n"; // Prints false.
```

◦

```
std::cout << std::is_integral<int>::value << "\n"; // Prints true.  
std::cout << std::is_integral<char>::value << "\n"; // Prints true.  
std::cout << std::is_integral<float>::value << "\n"; // Prints false.
```

**true**enum class◦

```
std::cout << std::is_integral<int>::value << "\n"; // Prints true.  
std::cout << std::is_integral<char>::value << "\n"; // Prints true.  
std::cout << std::is_integral<float>::value << "\n"; // Prints false.
```

## C ++ 11

◦ ◦

◦ ◦

◦

```
template<typename T>  
inline T FastDivideByFour(cont T &var) {  
    // Will give an error if the inputted type is not an unsigned integral type.  
    static_assert(std::is_unsigned<T>::value && std::is_integral<T>::value,  
        "This function is only designed for unsigned integral types.");  
    return (var >> 2);  
}
```

◦

```
template<typename T>  
inline T FastDivideByFour(cont T &var) {  
    // Will give an error if the inputted type is not an unsigned integral type.  
    static_assert(std::is_unsigned<T>::value && std::is_integral<T>::value,  
        "This function is only designed for unsigned integral types.");  
    return (var >> 2);  
}
```

**volatile****true**◦

```
template<typename T>  
inline T FastDivideByFour(cont T &var) {  
    // Will give an error if the inputted type is not an unsigned integral type.  
    static_assert(std::is_unsigned<T>::value && std::is_integral<T>::value,
```

```
    "This function is only designed for unsigned integral types.");
    return (var >> 2);
}
```

true。

```
template<typename T>
inline T FastDivideByFour(cont T &var) {
    // Will give an error if the inputted type is not an unsigned integral type.
    static_assert(std::is_unsigned<T>::value && std::is_integral<T>::value,
        "This function is only designed for unsigned integral types.");
    return (var >> 2);
}
```

true。

```
template<typename T>
inline T FastDivideByFour(cont T &var) {
    // Will give an error if the inputted type is not an unsigned integral type.
    static_assert(std::is_unsigned<T>::value && std::is_integral<T>::value,
        "This function is only designed for unsigned integral types.");
    return (var >> 2);
}
```

<https://riptutorial.com/zh-TW/cplusplus/topic/4750/>

# 147:

Tdynamic\_cast<T> static\_cast<T> reinterpret\_cast<T>const\_cast<T>◦

C++ T(expr) C(T) expr ◦

- *simple-type-specifier* ( )
- *simple-type-specifier* ( )
- *simple-type-specifier braced-init-list*
- *typename-specifier* ( )
- *typename-specifier* ( )
- *typename-specifier braced-init-list*
- dynamic\_cast <*type-id*> ( )
- static\_cast <*type-id*> ( )
- reinterpret\_cast <*type-id*> ( )
- const\_cast <*type-id*> ( )
- (*type-id*) *cast-expression*
- dynamic\_cast<Derived&>(base) ◦ ◦
- static\_cast<string&&>(s) ◦
- (int)x **prvalue** ◦

reinterpret\_cast“”

- **“”** ◦
- ◦

static\_cast

- 
- ◦ ◦
- void ◦

```
// on some compilers, suppresses warning about x being unused
static_cast<void>(x);
```

- ◦
- ◦ ◦
- **void\*T\*** ◦

C++ 11

- **Xstd::move** ◦ ◦

# Examples

static\_cast<sup>o</sup> static\_cast<sup>o</sup>

```
struct Base {};
struct Derived : Base {};
Derived d;
Base* p1 = &d;
Derived* p2 = p1; // error; cast required
Derived* p3 = static_cast<Derived*>(p1); // OK; p2 now points to Derived object
Base b;
Base* p4 = &b;
Derived* p5 = static_cast<Derived*>(p4); // undefined behaviour since p4 does not
// point to a Derived object
```

static\_cast<sup>o</sup>

```
struct Base {};
struct Derived : Base {};
Derived d;
Base* p1 = &d;
Derived* p2 = p1; // error; cast required
Derived* p3 = static_cast<Derived*>(p1); // OK; p2 now points to Derived object
Base b;
Base* p4 = &b;
Derived* p5 = static_cast<Derived*>(p4); // undefined behaviour since p4 does not
// point to a Derived object
```

dynamic\_cast<sup>o</sup> o std::bad\_cast std::bad\_cast<sup>o</sup>

```
struct Base {};
struct Derived : Base {};
Derived d;
Base* p1 = &d;
Derived* p2 = p1; // error; cast required
Derived* p3 = static_cast<Derived*>(p1); // OK; p2 now points to Derived object
Base b;
Base* p4 = &b;
Derived* p5 = static_cast<Derived*>(p4); // undefined behaviour since p4 does not
// point to a Derived object
```

const\_cast constconst<sup>o</sup> const\_cast const-correct<sup>o</sup> const char\*

```
void bad_strlen(char* );
const char* s = "hello, world!";
bad_strlen(s); // compile error
bad_strlen(const_cast<char*>(s)); // OK, but it's better to make bad_strlen accept const char*
```

const\_cast constconst<sup>o</sup>

const\_cast C ++const<sup>o</sup> o

```
void bad_strlen(char* );
```

```
const char* s = "hello, world!";
bad_strlen(s); // compile error
bad_strlen(const_cast<char*>(s)); // OK, but it's better to make bad_strlen accept const char*
```

reinterpret\_cast**resp**◦ **resp**◦ ◦ ◦

```
int x = 42;
char* p = static_cast<char*>(&x); // error: static_cast cannot perform this conversion
char* p = reinterpret_cast<char*>(&x); // OK
*p = 'z'; // maybe this modifies x (see below)
```

## C++11

reinterpret\_cast◦ ◦

```
int x = 42;
char* p = static_cast<char*>(&x); // error: static_cast cannot perform this conversion
char* p = reinterpret_cast<char*>(&x); // OK
*p = 'z'; // maybe this modifies x (see below)
```

## C++11

reinterpret\_cast**resp**◦ ◦

```
int x = 42;
char* p = static_cast<char*>(&x); // error: static_cast cannot perform this conversion
char* p = reinterpret_cast<char*>(&x); // OK
*p = 'z'; // maybe this modifies x (see below)
```

reinterpret\_cast**C++11**◦

reinterpret\_cast◦

reinterpret\_cast**reinterpret**\_cast◦

reinterpret\_cast**void**\* ◦ ◦ ◦

long**unsigned long**◦

## C++11

std::intptr\_t**std::uintptr\_t** **void**\* ◦ ◦

reinterpret\_cast◦ ◦

```
void register_callback(void (*fp)(void*), void* arg); // probably a C API
void my_callback(void* x) {
    std::cout << "the value is: " << reinterpret_cast<long>(x); // will probably compile
}
long x;
std::cin >> x;
register_callback(my_callback,
```

```
reinterpret_cast<void*>(x)); // hopefully this doesn't lose information...
```

- static\_cast◦ ◦

```
class C {
    std::unique_ptr<int> p;
public:
    explicit C(int* p) : p(p) {}
};

void f(C c);
void g(int* p) {
    f(p); // error: C::C(int*) is explicit
    f(static_cast<C>(p)); // ok
    f(C(p)); // equivalent to previous line
    C c(p); f(c); // error: C is not copyable
}
```

static\_cast。 static\_cast

- 66

```
const double x = 3.14;
printf("%d\n", static_cast<int>(x)); // prints 3
// printf("%d\n", x); // undefined behaviour; printf is expecting an int here
// alternative:
// const int y = x; printf("%d\n", y);
```

double°

- ```
const double x = 3.14;
printf("%d\n", static_cast<int>(x)); // prints 3
// printf("%d\n", x); // undefined behaviour; printf is expecting an int here
// alternative:
// const int y = x; printf("%d\n", y);
```

static cast $\circ\circ$

- `:static cast`。

C++ 11

- 1

```
enum class Format {
    TEXT = 0,
    PDF = 1000,
    OTHER = 2000,
};

Format f = Format::PDF;
int a = f;                                // error
```

```
int b = static_cast<int>(f);           // ok; b is 1000
char c = static_cast<char>(f);         // unspecified, if 1000 doesn't fit into char
double d = static_cast<double>(f);     // d is 1000.0... probably
```

- - ○ ○
  - <= C++ 14> = C++ 17○

```
enum class Format {
    TEXT = 0,
    PDF = 1000,
    OTHER = 2000,
};

Format f = Format::PDF;
int a = f;                      // error
int b = static_cast<int>(f);    // ok; b is 1000
char c = static_cast<char>(f);  // unspecified, if 1000 doesn't fit into char
double d = static_cast<double>(f); // d is 1000.0... probably
```

## C++ 11

- ○

```
enum class Format {
    TEXT = 0,
    PDF = 1000,
    OTHER = 2000,
};

Format f = Format::PDF;
int a = f;                      // error
int b = static_cast<int>(f);    // ok; b is 1000
char c = static_cast<char>(f);  // unspecified, if 1000 doesn't fit into char
double d = static_cast<double>(f); // d is 1000.0... probably
```

static\_cast○ ○

○

○ static\_cast○ ○

```
struct A {};
struct B { int x; };
struct C : A, B { int y; double z; };
int B::*p1 = &B::x;
int C::*p2 = p1;                  // ok; implicit conversion
int B::*p3 = p2;                  // error
int B::*p4 = static_cast<int B::*>(p2); // ok; p4 is equal to p1
int A::*p5 = static_cast<int A::*>(p2); // undefined; p2 points to x, which is a member
                                         // of the unrelated class B
double C::*p6 = &C::z;
double A::*p7 = static_cast<double A::*>(p6); // ok, even though A doesn't contain z
int A::*p8 = static_cast<int A::*>(p6);       // error: types don't match
```

\*T \*

**C++** void\*T\* T= static\_cast<T>(T);

## C++ 11

TT=

```
// allocating an array of 100 ints, the hard way
int* a = malloc(100*sizeof(*a)); // error; malloc returns void*
int* a = static_cast<int*>(malloc(100*sizeof(*a))); // ok
// int* a = new int[100]; // no cast needed
// std::vector<int> a(100); // better

const char c = '!';
const void* p1 = &c;
const char* p2 = p1; // error
const char* p3 = static_cast<const char*>(p1); // ok; p3 points to c
const int* p4 = static_cast<const int*>(p1); // unspecified in C++03;
// possibly unspecified in C++11 if
// alignof(int) > alignof(char)
char* p5 = static_cast<char*>(p1); // error: casting away constness
```

## C

C“C (NewType) variable。

### C++

- `const_cast<NewType>(variable)`
- `static_cast<NewType>(variable)`
- `const_cast<NewType>(static_cast<const NewType>(variable))`
- `reinterpret_cast<const NewType>(variable)`
- `const_cast<NewType>(reinterpret_cast<const NewType>(variable))`

NewType(expression) . .

### C++C++.

`reinterpret_cast` .

<https://riptutorial.com/zh-TW/cplusplus/topic/3090/>

S. No		Contributors
1	C ++	Adhokshaj Mishra, ankit dassor, aquirdturtle, ArchbishopOfBanterbury, Bakhtiar Hasan, Bart van Nierop, Ben H, Bo Persson, Brandon, Brian, BullshitPingu, cb4, celtschk, Cheers and hth. - Alf, chrisb2244, Cody Gray, Community, cpaticio, Curious, Daemon, Daksh Gupta, Danh, darkpsychic, David Bippes, David G., DeepCoder, Dim_ov, dlemstra, Donald Duck, Dr t, Dylan Little, Edward, emlai, Erick Q., ethanwu10, Fantastic Mr Fox, Florian, GIRISH kuniyal, greatwolf, honk, Humam Helfawi, Hurkyl, Ilyas Mimouni, Isak Combrinck, itzmukeshy7, Jason Watkins, JedaiCoder, Jerry Coffin, Jim Clark, Johan Lundberg, Jon Harper, jotik, Justin, Justin Time, JVAPen, K48, Ken Y-N, Keshav Sharma, kiner_shah, krOoze, Leandros, maccard, Malcolm, Malick, Manan Sharma, manetsus, manlio, Marco A., Mark Gardner, MasterHD, Matt, Matt Lord, mnoronha, Muhammad Aladdin, Mustaghees, muXXmit2X, mynameisausten, Nathan Osman, Neil A., Nemanja Boric, neuro, Nicol Bolas, CrazyPython, Optimus Prime, Pavel Strakhov, Peter, Praetorian, Qchmq, Quirk, RamenChef, Rushikesh Deshpande, SajithP, Sam Cristall, Serikov, Shoe, SirGuy, Soapy, Soul_man, theo2003, トトロ, qoq, Tom K, TriskalJM, Trizzle, UncleZeiv, VermillionAzure, Ville-Valtteri Tiittanen, Walter, Wen Qin, Wexiwa, πάντα ρεῖ, ハスカル
2	Arithmitic Metaprogramming	Meena Alfons
3	C ++ 11	NonNumeric
4	C ++ Streams	Ami Tavyory, didiz, JVAPen, mpromonet, Sergey
5	C ++	didiz
6	C ++	4444, JVAPen, lorro, mindriot
7	C ++	elvis.dukaj, VermillionAzure
8	C ++	John Bargman
9	c ++/	Daemon, Nicol Bolas, Владимир Стрелец
10	C ++	Gaurav Sehgal
11	C ++	celtschk, R_Kapp

12	C ++“””	Error - Syntactical Remorse, Henkersmann
13	C ++	John DiFini
14	C ++	Adam Trhon, JVApén, King's jester, Misgevolution
15	constexpr	Ajay, Brian, diegodfrf, mtb, Null
16	Const	amanuel2, Justin Time
17	const	Barry, Jarod42, Jatin, Justin, Podgorskiy, tenpercent, ThyReaper
18	C	パスカル
19	decltype	Ajay
20	ISO C ++	Bakhtiar Hasan, Barry, C.W.Holeman II, ComicSansMS, didiz, diegodfrf, Guillaume Pascal, Ivan Kush, Johan Lundberg, Justin Time, JVApén, manlio, Marco A., MSalters, Nicol Bolas, sth, vishal
21	Lambda	Adi Lester, Aganju, Ajay, alain, anderas, Andrea Corbelli, Barry, bcmpinc, Brian, Christopher Oezbek, cpplearner, derekerdmann, Edd, Falias, Firas Moalla, honk, Jean-Baptiste Yunès, Johan Lundberg, Johannes Schaub - litb, John Slegers, JVApén, Loki Astari, Loufylouf, M. Viaz, Mike Dvorkin, Nicol Bolas, Patryk, Praetorian, Rakete1111, RamenChef, Ryan Haining, Sergio, Serikov, Snowhawk, teivaz, Ville-Valtteri Tiittanen, Yakk, ygram
22	Pimpl	Danh, Daniele Pallastrelli, emlai, Jordan Chapman, JVApén, manlio, Stephen Cross, Yakk
23	RAII	Barry, defube, Jarod42, JVApén, Loki Astari, Niall, Nicol Bolas, RamenChef, Sumurai8, Tannin
24	RTTI	Brian, deepmax, Pankaj Kumar Boora, Roland, Savas Mikail KAPLAN
25	SFINAE	Barry, Fox, Jarod42, Jason R, Jonathan Lee, Luc Danton, sp2danny, SU3, w1th0utnam3, Xosdy, Yakk
26	static_assert	Jarod42, JVApén, Iorro, Marco A., Richard Dally, T.C.
27	std :: function	elimad, Evgeniy, Nicol Bolas, Tarod
28	std :: setstd :: multiset	G-Man, JVApén, Mikitori

29	TypeDef	Brian
30	ODR	Brian, Jarod42
31		Adrien Descamps, Barry, ChrisN, hello, honk, Johan Lundberg, Justin Time, JVApén, Loki Astari, mpromonet, Nicol Bolas, Nirmal4G, NonNumeric, Null, Peter, relgukxilef, Scott Weldon, T.C., TriskalJM, Venemo
32	C ++C ++ 11C ++ 14 C ++ 17C ++	wasthishelpful
33		didiz, hyoslee, JVApén
34		Ajay, Perette Barella
35		A. Sarid, Barry, Cody Gray, CroCo, FedeWar, Jarod42, JVApén, manlio, tambre, Tarod, Trevor Hickey, Алексей Неудачин
36		Loki Astari, Mads Marquart, manlio, txtechhelp, Алексей Неудачин
37	OpenMP	Andrea Chua, JVApén, Nicol Bolas, Sumurai8
38	std :: unordered_map	tulak.hord
39		Edward, marcinj, Naor Hadar, RamenChef
40		Brian
41		Alexey Guseynov, Brian, callyalater, Dr t, Jahid, Jarod42, Johan Lundberg, jotik, Martin Ba, Nemanja Boric, Null, Peter, Rakete1111, Ronen Ness
42		didiz
43		JVApén, Nicol Bolas
44		Barry, ChemiCalChems, Curious, fefe, Johannes Schaub - litb, mnoronha, Nicol Bolas, Praetorian, SirGuy
45		chema989, ralismark
46		anderas, Barry, Brian, celtschk, Colin Basnett, DawidPi, deepmax, dmi_, Holt, Jarod42, Justin, manlio, Matthieu M., Nicol Bolas, Oz., rhynodegreat, rtmh, sth, TartanLlama, Venki, W.F., ysdx, πάντα ρέτ
47		Andrei, Brian, callyalater, Daksh Gupta, Galik, JVApén,

		madduci, nnrales, RamenChef, ThyReaper
48		amanuel2, Aravind .KEN, Bim, Brian, legends2k
49		Brian
50		Denkkar, Fantastic Mr Fox, Jarod42, Nicol Bolas, SajithP, stackptr, T.C.
51		Ami Tavory, paul-g
52		Johannes Schaub - litb, Kunal Tyagi, RamenChef
53		Bakhtiar Hasan, Barry, Cody Gray, CoffeeandCode, didiz, Galik, Jatin, Johannes Schaub - litb, Justin Time, JVApén, Rakete1111, Sean, Sumurai8, Tim Straubinger
54		JVApén, Stephen
55		Fanael, Johannes Schaub - litb
56		Marco A.
57		Andrea Corbelli, Asu, Daksh Gupta, darkpsychic, rockoder
58		JVApén, turoni
59		Barry, Community, Dean Seo, start2learn, T.C., tenpercent
60		anderas, Andrea Chua, Barry, Brian, DeepCoder, emlai, Isak Combrinck, Jarod42, Jérémie Roy, Johannes Schaub - litb, Julien-L, JVApén, Nicol Bolas, Null, Rakete1111, randag, Roland, T.C., tenpercent, Yakk
61		deepmax, Galik, Jarod42, Johan Lundberg, JVApén, Stradigos
62		amanuel2, Brian, Kerrek SB, RamenChef
63		A. Sarid, Christophe, Jarod42, Jeremi Podlasek, Justin Time, manlio
64	CRTP	Barry, Brian, Gabriel, honk, Nicol Bolas, Ryan Haining
65		Brian, Nikola Vasilev, RamenChef
66		Brian, start2learn
67		In silico, Johannes Schaub - litb, Nicol Bolas, Roland
68		Abhinav Gauniyal

69		2501, Bo Persson, Brian, Dutow, Jahid, Jarod42, jotik, Justin Time, Iz96, manlio, Nicol Bolas, Peter
70		Brian, Justin Time
71		Brian, Marco A., Nicol Bolas
72		ibrahim5253, JVAPen, Kerrek SB, MathSquared, SingerOfTheFall
73		manlio, ThyReaper, txtechhelp
74		honk, Jonathan Mee, Justin, JVAPen
75	/GCC	Asu, immerhart
76		Vijayabhaskarreddy CH, Yakk
77		aaronsnoswell, Bakhtiar Hasan, Barry, bitek, celtschk, Christopher Oezbek, DeepCoder, Dr t, Ela782, Fantastic Mr Fox, Galik, honk, J_T, Jarod42, Johan Lundberg, Johannes Schaub - litb, John Slegers, Jon Chesterfield, Kevin Katzke, Let_Me_Be, Loki Astari, M. Sadeq H. E., manetsus, Menasheh, Michael Gaskill, mnoronha, Niall, Nicol Bolas, Null, Peter, Rakete1111, Richard Forrest, Ryan Hilbert, Stephen, T.C., templatetypedef, tenpercent, user3384414, Ville-Valtteri Tiittanen, Yakk, Ze Rubeus, パスカル
78		ankit dassor, anotherGatsby, Barry, ChemiCalChems, Chris, ChrisN, Christian Rau, ColleenV, Debanjan Dhar, DrZoo, Edward, emlai, holmicz, honk, Johannes Schaub - litb, Justin Time, L.V.Rao, manlio, Nicholas, Nicol Bolas, Null, Ped7g, pmelanson, Pyves, Rakete1111, Sergey, sp2danny, user1336087, VladimirS, Yakk
79		AndyG, Barry, cplearn, Firas Moalla, Marco A., Rakete1111, T.C., Yakk
80		John Burger, start2learn
81		Baron, daB0bby, FedeWar, Hindrik Stegenga, Nicol Bolas, Nitinkumar Ambekar, Pietro Saccardi, Reverie Wisp, West
82		anatolyg, Barry, Daniel, Ivan Kush, maccard, manetsus, manlio, MikeMB, MKAROL, Nicol Bolas, Patrick, Ravi Chandra, SajithP, timrau, Trevor Hickey
83		diegodfrf, JVAPen
84		Cheers and hth. - Alf, Isak Combrinck, manlio, Matthew Brien,

85	I / O.	anderas, ankit dassor, Anonymous1847, AProgrammer, Bakhtiar Hasan, bitek, Chachmu, ComicSansMS, didiz, Dietmar Kühl, Dr t, Emanuel Vintilă, Galik, honk, Hurkyl, Jérémie Bolduc, John Strood, JVApén, Loki Astari, manlio, Mathieu K., MikeMB, mindriot, Nicol Bolas, nwp, patmanpato, Rakete1111, RomCoo, Serikov, sheng09, shrike, svgsprn, Алексей Неудачин
86		Abyx, Ajay, Alexey Voytenko, anderas, Barry, CaffeineToCode, Christopher Oezbek, Cody Gray, ComicSansMS, cplearner, Daksh Gupta, Danh, Daniele Pallastrelli, DeepCoder, Edward, emlai, foxcub, Francis Cugler, honk, Jack Zhou, Jared Payne, Jarod42, Johan Lundberg, Johannes Schaub - litb, jotik, Justin, JVApén, Kerrek SB, King's jester, Loki Astari, manlio, Marco A., MC93, Menasheh, Meysam, PcAF, Rakete1111, Reuben Thomas, Richard Dally, rodrigo, Roland, sami1592, sth, Sumurai8, tysonite, user3684240, Xirema, Yakk
87		Perette Barella, Sergey
88		didiz, Nicol Bolas
89		jotik, Roland, ThyReaper
90		Ami Tavory, AndreiM, Ben Steffan, Brian, Cody Gray, cshu, Dovahkiin, Elias Kosunen, emlai, Emma X, FedeWar, fefe, ggrr, GIRISH kuniyal, Hiura, Jeremi Podlasek, Johannes Schaub - litb, JVApén, kd1508, Ken Y-N, manetsus, manlio, Marco A., Mat, mceo, Motti, Naor Hadar, nbro, Nicol Bolas, Peter, Rakete1111, ralismark, RamenChef, Sebastian Ärleryd, Tannin, Trevor Hickey, Tyler Durden
91		AndreiM, Brian, Jarod42, Yakk
92		Ami Tavory, celtschk, Florian, Jahid, Jason Watkins, Justin, JVApén, Nathan Osman, RamenChef, VermillionAzure
93		Ami Tavory, Barry, Daniel, Duly Kinsky, Edgar Rokyan, Guillaume Pascal, Jarod42, NinjaDeveloper, Patryk Obara, Peter, Riom
94		Barry, Benjy Kessler, Brian, callyalater, cb4, celtschk, CodeMouse92, Colin Basnett, DeepCoder, Diligent Key Presser, Eldritch Cheese, eXPerience, FedeWar, Gabriel, Greg, Holt, honk, J_T, Johannes Schaub - litb, Justin, JVApén, Loki Astari, M. Viaz, manlio, Maxito, MSalters, Nicol Bolas, Pontus Gagge, Praetorian, Rakete1111, Ricardo Amores, Ryan

		Haining, Sergey, SirGuy, Smeeheey, Sumurai8, user1887915, W.F., WMios, Wolf, панта ѡєї
95		Artalus, Barry, celtschk, Daniele Pallastrelli, Edward, Igor Oks, Jarod42, Johan Lundberg, manlio, Yakk
96		Nicol Bolas, Владимир Стрелец
97		anotherGatsby, Brian, JVApenn, mkluwe, Qchmq, RamenChef, Tejendra
98		Xirema
99		Barry, krOoze, OliPro007, Reuben Thomas, TriskalJM
100	C ++	Antonio Barreto, datosh, didiz, Jarod42, JVApenn, Nikola Vasilev
101		Brian, Cid1025, Jarod42, Roland, sigalor, sth
102	std :: integer_sequence	Dietmar Kühl
103	std :: string	1337ninja, 3442, Andrea Corbelli, Barry, Bim, caps, Christopher Oezbek, cplearn, crea7or, Curious, drov, Edward, Emil Rowland, emlai, Fantastic Mr Fox, fbrereto, ggrr, Holt, honk, immerhart, Jack, Jahid, Jerry Coffin, Jonathan Mee, jotik, JPNotADragon, jpo38, Justin, JVApenn, Ken Y-N, Leandros, Loki Astari, manetsus, manlio, Marc.2377, Matthew, Matthieu M., Meysam, Michael Gaskill, mpromonet, Niall, Null, Rakete1111, RamenChef, Richard Dally, SajithP, Serikov, sigalor, Skipper, Soapy, sth, T.C., Tharindu Kumara, Trevor Hickey, user1336087, user2176127, W.F., Wolf, Yakk
104	std ::iomanip	kiner_shah, Nikola Vasilev, Yakk
105	std ::	demonplus, Marco A.
106	std ::Modifiers	Nikola Vasilev
107	std ::	Nikola Vasilev
108	std ::	Barry, diegodfrf, Jahid, Jared Payne, JVApenn, Null, Yakk
109	std ::	Andrea Corbelli, ankit dassor, ChrisN, CinCout, ComicSansMS, CygnusX1, davidsheldon, diegodfrf, Fantastic Mr Fox, foxcub, Galik, honk, jmmut, manetsus, manlio, Meysam, Naveen Mittal, Null, Peter, Richard Dally, rick112358, Savan Morya, user1336087, vdaras, VolkA, Wyzard
110	std ::	Ajay, Bim, demonplus, kiner_shah, Nikola Vasilev, Ravi

Chandra		
111	std ::	2power10, A. Sarid, Aaron Stein, alain, Alex Logan, Ami Tavory, anatolyg, anderas, Andy, AndyG, arunmoezhi, Bakhtiar Hasan, Barry, Benjamin Lindley, bluefog, bone, CHess, CinCout, Cody Gray, Colin Basnett, ComicSansMS, cute_ptr, Daksh Gupta, Daniel, Daniel Stradowski, Dario, David G., David Yaw, DeepCoder, diegodfrf, dkg, Dr t, Duly Kinsky, Ed Cottrell, Edward, ehudt, emlai, enrico.bacis, Falias, Fantastic Mr Fox, Fox, foxcub, gaazkam, Galik, gartenriese, granmirupa, Holt, honk, Hurkyl, iliketocode, immerhart, Isak Combrinck, Jarod42, Jason Watkins, JHBonarius, Johan Lundberg, John Slegers, jotik, jpo38, JVApén, Kevin Katzke, krOoze, Loki Astari, lordjohn cena, manetsus, manlio, Marco A., Matt, Michael Gaskill, Misha Brukman, MotKohn, Motti, mtk, NageN, Niall, Nicol Bolas, Null, patmanpato, Paul Beckingham, paul-g, Ped7g, Praetorian, Pyves, R. Martinho Fernandes, Rakete1111, Randy Taylor, Richard Dally, Roddy, Romain Vincent, Rushikesh Deshpande, Ryan Hilbert, Saint-Martin, Samar Yadav, Samer Tufail, Sayakiss, Serikov, Shoe, silvergasp, Skipper, solidcell, Stephen, sth, strangeqargo, T.C., Tamarous, theo2003, Tom, towi, Trevor Hickey, TriskalJM, user1336087, user2176127, Ville-Valtteri Tiittanen, Vladimir Gamalyan, Wolf, Yakk
112	std ::	Yakk
113	std ::	CinCout, Daksh Gupta, Dinesh Khandelwal, Error - Syntactical Remorse, Malcolm, Nikola Vasilev, plasmacel
114		Barry, Cheers and hth. - Alf, ChemiCalChems, David Doria, didiz, Guillaume Racicot, Justin Time, JVApén
115		Alejandro, amchacon, Brian, CaffeineToCode, ComicSansMS, Dair, defube, didiz, Diligent Key Presser, Galik, James Adkison, james large, Jason Watkins, Jeremi Podlasek, mpromonet, Niall, nwp, Rakete1111, Stephen Cross, Sumurai8, Yakk, ysdx, Yuushi
116		didiz, Galik, JVApén
117		4444, Adhokshaj Mishra, Ami Tavory, ArchbishopOfBanterbury, Barry, Ben Steffan, celtschk, Curious, Donald Duck, Dr t, elvis.dukaj, Fantastic Mr Fox, Florian, greatwolf, Griffin, Isak Combrinck, Jahid, Jarod42, Jason Watkins, Johan Lundberg, jotik, Justin, Justin Time, JVApén, madduci, Malick, manetsus, manlio, Matt, Michael Gaskill, Morten Kristensen, MSD, muXXmit2X, n.m., Nathan Osman, Nemanja Boric, Peter, Quirk

		, Richard Dally, Sergey, Tharindu Kumara, Toby, Trygve Laugstøl, VermillionAzure
118		Brian
119		0x5f3759df, Daksh Gupta, Johan Lundberg, Justin Time, Motti, Sergey, T.C.
120		Ajay, BigONotation, celtschk, defube, Jarod42, Jonathan Lee, Roland, T.C., Yakk
121	Elision	Nicol Bolas, TartanLlama
122		amanuel2, Roland
123		Brian, RamenChef, start2learn
124		Anonymous1847
125		Brian, Justin Time, Omnifarious, RamenChef
126		Cheers and hth. - Alf, sorosh_sabz
127		Brian, Daniel Käfer, Emmanuel Mathi-Amorim, marquesm91, RamenChef
128		Barry, chrisb2244, cute_ptr, Daniel Jour, Edgar Rokyan, EvgeniyZh, fbrereto, Gal Dreiman, Gaurav Kumar Garg, GIRISH kuniyal, honk, Hurkyl, JPNotADragon, JVOpen, Mike H-R, Null, Oz., Sergey, Serikov, tilz0R, Yakk
129		Brian, define cindy const, Torbjörn
130		amanuel2, Justin Time, RamenChef
131		an0o0nym, Brian, didiz, JVOpen, start2learn, turoni
132		ArchbishopOfBanterbury, Archie Gertsman, Ates Goral, Barry, Brian, Candlemancer, chrisb2244, defube, enzom83, James Adkison, Rakete1111, Sergey, start2learn, Xeverous, Yakk
133		Barry, Brian, didiz, Johannes Schaub - litb
134		didiz
135		asantacreu, Cody Gray, Edward, Jarod42, JVOpen, RamenChef
136		ADITYA, ammanuel2, Brian, Danh, John London, Justin Time, JVOpen, Kerrek SB, Loki Astari, manlio, Marco A., Nicol Bolas, OliPro007, Rakete1111, RamenChef, Roland, start2learn

137		Ha., manlio, merlinND, Sumurai8
138		Justin Time, RamenChef
139		alain, callyalater, Cheers and hth. - Alf, CygnusX1, Fantastic Mr Fox, Fox, Francisco P., Ian Ringrose, immerhart, InitializeSahib, Johan Lundberg, Justin, Justin Time, Ken Y-N, Kieran Chandler, krOoze, manlio, Marco A., Maxito, n.m., Nicol Bolas, Peter, phandinhlan, Richard Dally, Sean, signal, silvergasp, Sumurai8, T.C., Tanjim Hossain, tenpercent, The Philomath, Владимир Стрелец, パスカル
140		deepmax, Error - Syntactical Remorse
141		RamenChef, VermillionAzure
142 /		Alexey Voytenko, anderas, aquirdturtle, Brian, callyalater, chrisb2244, Colin Basnett, Dan Hulme, darkpsychic, Dragma, Fantastic Mr Fox, Firas Moalla, Jarod42, Jerry Coffin, jotik, Justin Time, Kerrek SB, Nicol Bolas, Null, OliPro007, PcAF, Ph03n1x, pingul, Rakete1111, Sándor Mátyás Márton, Sergey, silvergasp, Skywrath, Yakk
143		Barry, Brian, Emmanuel Mathi-Amorim
144		Andrea Chua, Jim Clark
145		Brian, celtschk, greatwolf, Jarod42, Yakk
146		Jarod42, silvergasp, TartanLlama
147		4444, Brian, JVApén, Nikola Vasilev