

# Derived C++

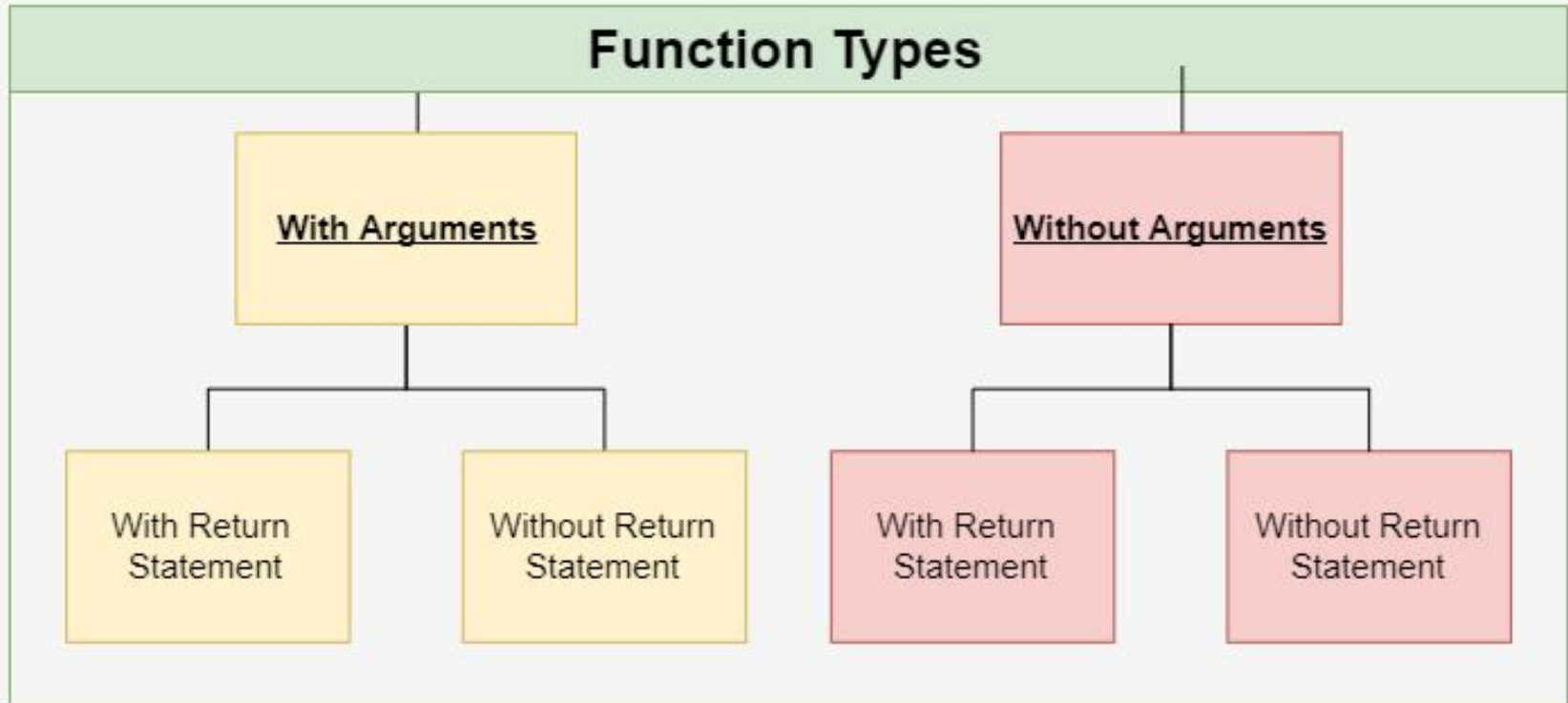


Derived Datatypes with Moatasem Elsayed

# Content

- *Functions*
- *Array*
- *pointers*
- *References*
- *auto*
- *Casting*
- *Lambda*
- *Const vs constexpr*

# Recap on C



# Basic function

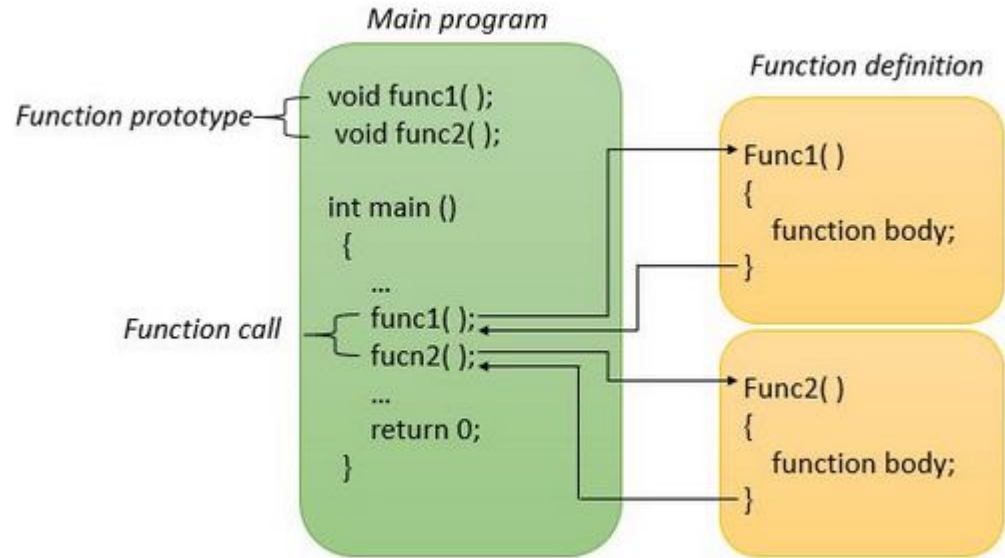
```
#include <iostream>

// Function declaration
int add(int a, int b) {
    return a + b;
}

int main() {
    // Function call
    int result = add(5, 3);
    std::cout << "Result: " << result << std::endl;
    return 0;
}
```

```
// Declaration
int add(int a, int b);

// Definition
int add(int a, int b) {
    return a + b;
}
```



# Default Parameters:

```
#include <iostream>

void printMessage(std::string message = "Hello, World!") {
    std::cout << message << std::endl;
}

int main() {
    printMessage();           // Prints default message
    printMessage("Hi there!"); // Prints custom message
    return 0;
}
```

# Tricky

```
3 void fun(int x = 2, int y);
4 void fun(int x, int y)
5 {
6     std::cout << x << " " << y << std::endl;
7 }
8
9 int main()
10 {
11     fun(2);
12     return 0;
13 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
10
moatasem@moatasem-Inspiron-3542:~/c++/workspace$ g++ main.cpp
main.cpp:3:25: error: default argument missing for parameter 2 of 'void fun(int, int)'
   3 | void fun(int x = 2, int y);
     |                    ~~~~~^
main.cpp:3:14: note: ...following parameter 1 which has a default argument
   3 | void fun(int x = 2, int y);
     |          ~~~~~^~~~~
moatasem@moatasem-Inspiron-3542:~/c++/workspace$
```

# Function Overloading:

```
#include <iostream>

int multiply(int a, int b) {
    return a * b;
}

double multiply(double a, double b) {
    return a * b;
}

int main() {
    int result1 = multiply(3, 4);
    double result2 = multiply(2.5, 3.0);

    std::cout << "Result 1: " << result1 << std::endl;
    std::cout << "Result 2: " << result2 << std::endl;

    return 0;
}
```

# demangle

```
● moatasem@CAI1-L14000:~/vsomeIp$ objdump -t --demangle | grep fun
00000000000001295 l      F .text 00000000000000019      _GLOBAL__sub_I__Z3funi
000000000000011e3 g      F .text 0000000000000003d      fun(int, int)
000000000000011a9 g      F .text 0000000000000003a      fun(int)

● moatasem@CAI1-L14000:~/vsomeIp$ objdump -t | grep fun
00000000000001295 l      F .text 00000000000000019      _GLOBAL__sub_I__Z3funi
000000000000011e3 g      F .text 0000000000000003d      _Z3funii
000000000000011a9 g      F .text 0000000000000003a      _Z3funi

○ moatasem@CAI1-L14000:~/vsomeIp$
```



# Tricks

error: call of overloaded 'fun(int)'

is ambiguous

```
3
4 void fun(int x, int y = 3)
5 {
6     std::cout << x << " " << y << std::endl;
7 }
8 void fun(int x, float y = 2)
9 {
10    std::cout << "hello float" << std::endl;
11 }
12 int main()
13 {
14     fun(2);
15     fun(2, 3);
16     return 0;
17 }
```

PROBLEMS 2

OUTPUT

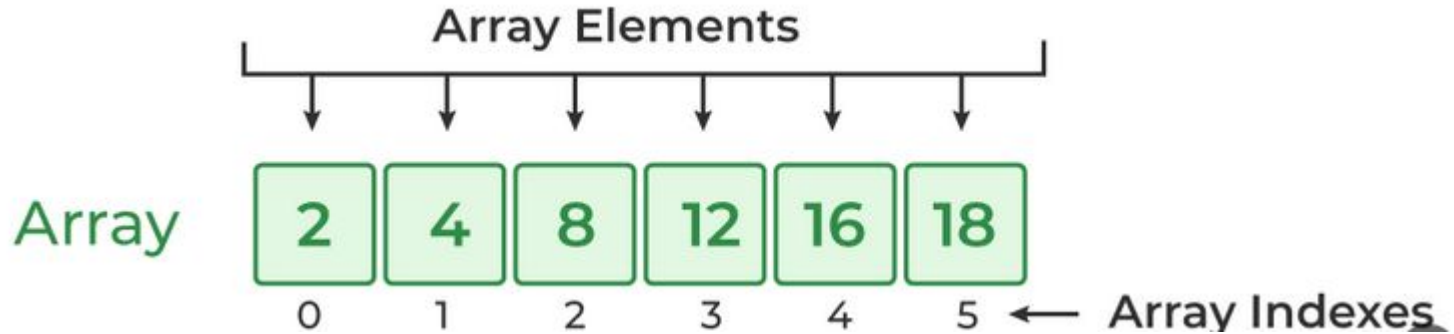
DEBUG CONSOLE

TERMINAL

JUPYTER

```
14 |     fun(2);
    |           ^
main.cpp:4:6: note: candidate: 'void fun(int, int)'
4 | void fun(int x, int y = 3)
    |           ^
main.cpp:8:6: note: candidate: 'void fun(int, float)'
```

# Arrays



## Declaration

To declare an array, specify the type of its elements and the number of elements it can hold:

```
int numbers[5]; // Declare an integer array with 5 elements
```

# Initialization

Arrays can be initialized when declared:

```
int numbers[5] = {1, 2, 3, 4, 5}; // Initialize array elements
```

Or you can omit the size, and the compiler will determine it from the number of elements:

```
int numbers[] = {1, 2, 3, 4, 5}; // Compiler determines size as 5
```

## Accessing Elements

```
int thirdNumber = numbers[2]; // Access the third element (3)
```

# Iterating through an Array

```
for (int i = 0; i < 5; ++i) {  
    cout << numbers[i] << " ";  
}
```

Use Ranged loop

```
int main() {  
    int numbers[] = {[0]=1, [1]=2, [2]=3, [3]=4, [4]=5}  
    for (int value : numbers) {  
        std::cout << value << " ";  
    }  
    return 0;  
}
```

# Multidimensional Arrays

Arrays can have more than one dimension:

```
int main() {  
    int matrix[3][3] = {  
        [0]={ [0]=1, [1]=2, [2]=3},  
        [1]={ [0]=4, [1]=5, [2]=6},  
        [2]={ [0]=7, [1]=8, [2]=9}  
    };  
  
    for (int i = 0; i < 3; ++i) {  
        // Iterate through columns  
        for (int j = 0; j < 3; ++j) {  
            std::cout << matrix[i][j] << " ";  
        }  
        std::cout << std::endl; // Move to the next line after each row  
    }  
}
```

# Array Size

```
int main() {  
    int numbers[] = {[0]=1, [1]=2, [2]=3, [3]=4, [4]=5};  
  
    int size = sizeof(numbers) / sizeof(numbers[0]); // Calculate si
```

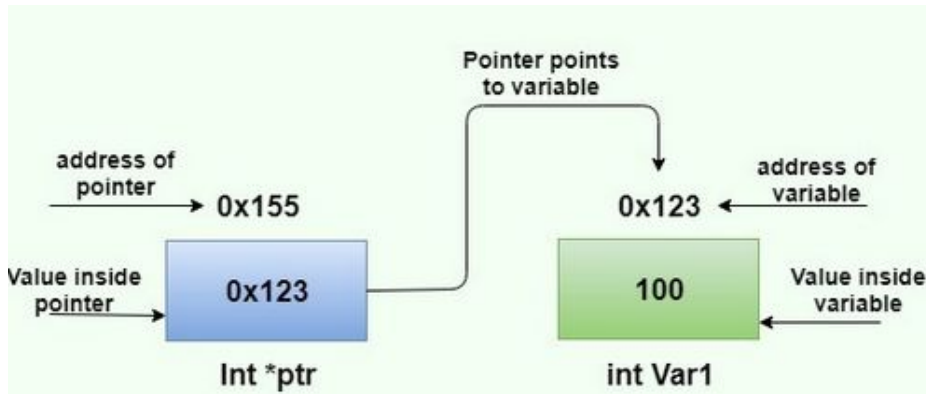
## Arrays and Functions

```
void printArray(int arr[], int size) {  
    for (int i = 0; i < size; ++i) {  
        std::cout << arr[i] << " ";  
    }  
}  
  
int main() {  
    int numbers[] = {[0]=1, [1]=2, [2]=3, [3]=4, [4]=5};  
    int size =  
        sizeof(numbers) / sizeof(numbers[0]); // Calculate the size of the array  
  
    printArray(arr: numbers, size);  
    return 0;  
}
```

# Pointers

## Basics of Pointers

- A pointer is a variable that stores a memory address.
- Declare a pointer with the `*` symbol



```
int x = 10;  
int *ptr = &x; // ptr now holds the address of x
```

## Dereferencing Operator (\*)

Use the dereferencing operator `*` to access the value pointed to by the pointer:

```
int value = *ptr; // value now holds the value of x (10)
```

# Pass by pointer

```
// Example 4: Pointers and Functions
void modifyValue(int *ptr) {
    *ptr = 100;
}
```

```
int num = 50;
modifyValue(&num);
std::cout << "Modified num: " << num << std::endl;
```



# Cont ..

## Null Pointers

Pointers can be set to `nullptr` (C++11) to indicate no valid address:

```
int *ptr = nullptr; // Initialize pointer to nullptr
```

## Dynamic Memory Allocation

Use `new` to allocate memory on the heap:

```
int *ptr = new int; // Allocates memory for an integer
*ptr = 42; // Assign a value to the memory location
```

## Memory Deallocation

Always release dynamically allocated memory with `delete`:

```
delete ptr; // Frees the memory allocated with new
ptr = nullptr; // Point the pointer to nullptr after deletion
```

## Arrays and Pointers

- Arrays are closely related to pointers:

cpp

```
int arr[5] = {1, 2, 3, 4, 5};  
int *ptr = arr; // Pointer points to the first element of the array
```

## Pointer Arithmetic

- Pointers can be incremented and decremented:

cpp

```
int *nextPtr = ptr + 1; // Points to the next element in the array 2
```

## Pointers to Functions

- Pointers can point to functions:

```
void func() { cout << "Hello!"; }  
void (*funcPtr)() = &func; // Pointer to the function  
(*funcPtr)(); // Calling the function through the pointer
```

# References vs. Pointers

## References vs. Pointers

- References are aliases for existing variables, while pointers can be reassigned:  
cpp

```
int x = 10;  
int &ref = x; // Reference to x  
int *ptr = &x; // Pointer to x
```

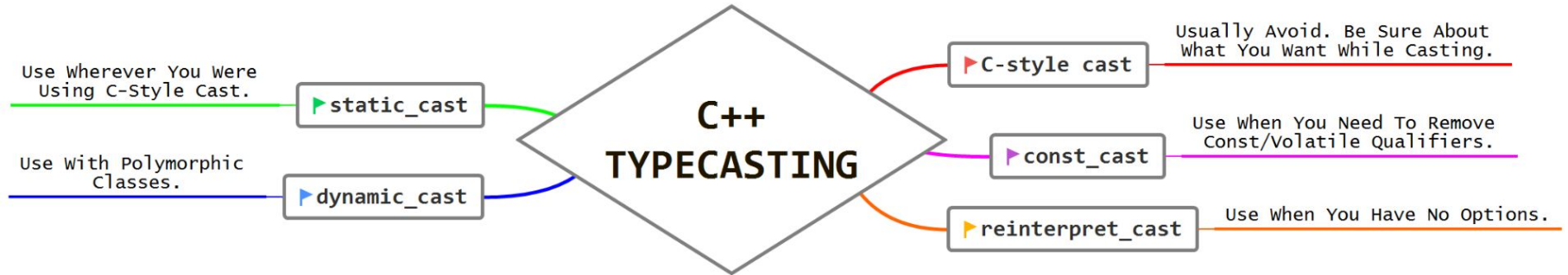
```
int x{10};  
int &y = x;  
  
int *ptr = &x; // lvalue {it has a certin name , that you did }  
  
y = 3;  
std::cout << "ref y= " << y << " x= " << x << "ptr " << *ptr << std::endl;  
  
// change reference  
int n = 0;  
y = n;  
y = 12;  
std::cout << "x =" << x << "n=" << n << std::endl;  
// unitialized referance is error  
// int &y2; // Error  
  
return 0;
```

# auto

```
1
auto var = 10;
auto var2 = 10.5;
auto var3 = 'a';
auto var4 = "const char*";
auto var5 = std::string("hello String ");
// with const
const int x = 10;
auto var6 = x; // var is int
/** to make it const
const auto var7 = x;
// with reference const
auto &var8 = x;
// with pointer const
auto ptr = &x;
// std::initializer_list
auto lst = {1, 2, 3, 4, 5};
// error
// auto lst{1, 2, 3, 4};
return 0;
}
```

```
22
23 // Data d;
24 // printf("welcome to c programming ");
25 // d.HelloWorld();
26 return 0;
27 }*/
28 int main()
29 {
30 int var = 10;
31 double var2 = 10.5;
32 char var3 = 'a';
33 const char * var4 = "const char*";
34 std::basic_string<char> var5 = std::basic_string<char>(std::basic_string<char>("h
35 const int x = 10;
36 int var6 = x;
37 const int var7 = x;
38 const int & var8 = x;
39 const int * ptr = &x;
40 std::initializer_list<int> lst = std::initializer_list<int>{1, 2, 3, 4, 5};
41 return 0;
42 }
43
```

# C++ casting



## **Recap C style cast**

# Static\_cast

Static\_cast:

This is the simplest type of cast which can be used. It is a compile time cast. It does things like implicit conversions between types (such as int to float, or pointer to void\*), and it can also call explicit conversion functions (or implicit ones). GeeksforGeeks

```
int a = 10;
char c = 'a';

// pass at compile time, may fail at run time
int* q = (int*)&c;
int* p = static_cast<int*>(&c); // error: invalid static_cast from type 'char*' to type 'int*'
```

```
int value=10;
float x=static_cast<float>(value);
```

Rule 5-0-5

Rule 5-0-6

# const\_cast

const\_cast is used to cast away the constness of variables

```
const int val = 10;
const int *ptr = &val;
// int*ptr1=ptr #ERROR
int *ptr1 = const_cast <int *>(ptr);
// int*ptr1=static_cast<int*>(ptr); // ERROR static_cast cannot cast away const or other type qualifiersC/C++(694)
```

**Rule 5-2-5 (Required)**

**A cast shall not remove any *const* or *volatile* qualification from the type of a pointer or reference.**

```
void f ( const char * p )
{
    *const_cast< char * >( p ) = '\0'; // Non-compliant
    std::cout <<p<<std::endl; //Segmentation fault
}
int main ( )
{
    f ( "Hello World!" );
}
```



# reinterpret\_cast

It is used to convert a [pointer of some data type](#) into a [pointer of another data type](#), even if the data types before and after conversion are different.

```
int* p = new int(65);  
char* ch = reinterpret_cast<char*>(p);
```

```
char* x=new char(5);
```

a value of type "char \*" cannot be used to initialize an entity of type "int \*"

[View Problem \(Alt+F8\)](#) [Quick Fix... \(Ctrl+.\)](#)

```
int*ptr=reinterpret_cast<char*>(x);
```

# Use it correctly

1. `reinterpret_cast` is a very special and dangerous type of casting operator. And is suggested to use it using proper data type i.e., (pointer data type should be same as original data type).
2. It can typecast any pointer to any other data type.

<b>Rule 5-2-7</b>	<b>(Required)</b>	<b>An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly.</b>
-------------------	-------------------	--

[Unspecified 5.2.10(7)]

## Rationale

The result of converting from a pointer to an unrelated type is unspecified.

```
class A {
public:
    int x=10;
    void fun_a()
    {
        cout << " In class A\n";
    }
};

class B {
public:
    int x=12;
    void fun_b()
    {
        cout << " In class B\n";
    }
};

int main()
{
    // creating object of class B
    B* x = new B();
    // converting the pointer to object
    A* new_a = reinterpret_cast<A*>(x);
    // accessing the function of class A
    new_a->fun_a(); // In class A
    std::cout << new_a->x << std::endl; // 12
    return 0;
}
```

# **dynamic\_cast**

Very important and we will postponed after OOP

# In practise

- Avoid C-style casts. Be sure about what you want while casting.
- Use `static_cast` wherever you were using C-style cast.
- Use `dynamic_cast` with polymorphic classes.
- Use `const_cast` when you need to remove `const` or `volatile` qualifiers.
- Use `reinterpret_cast` when you have no options.

Note: `const_cast` and `reinterpret_cast` should generally be avoided because they can be harmful if used incorrectly. Don't use it unless you have a very good reason to use them.

# Lambda expression

C++ 11 introduced lambda expressions

```
[ capture clause ] (parameters) -> return-type  
{  
    definition of method  
}
```

```
std::vector<int> v{1,2,3,4,5,6,7,8};  
// Lambda expression to print vector  
std::for_each(v.begin(), v.end(), [](int i)  
{  
    std::cout << i << " ";  
});
```

# C++ insight, check that after Class

```
6
7 // Driver Code
8 int main()
9 {
10     auto func = [] (int first, int second)
11     {
12         return first + second;
13     };
14     func(2,3);
15     return 0;
16 }
17
```

```
7 // Driver Code
8 int main()
9 {
10
11     class __lambda_10_15
12     {
13     public:
14         inline int operator()(int first, int second) const
15         {
16             return first + second;
17         }
18     }
19
```

```
37     func.operator()(2, 3);
```

# Capture list

```
1
2
3
4
5
6 int main()
7 {
8     int x=10;
9     int y=12;
10    [&x,y]() {
11        x=1;
12        return x+y;
13    }();
14
15    return 0;
16 }

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```
int main(){
    int temp=10;
    [](int number){
        return number +temp; // 1- Error need to capture it
    };

    [temp](int number){ // capture value
        // temp =1*2; // 2- capture it with const temp
        return number +temp; // 3- read only
    };

    [&temp](int number){ // capture reference
        temp =1*2; // 4- change temp of temp itself
        return number +temp;
    };

    [&](int number){ // capture all by reference
        temp =1*2; // 4- change temp of temp itself
        return number +temp;
    };

    [=](int number){ //5- capture all by value
        return number +temp;
    };

    [=,&temp](int number){ //6- capture all by value except temp by references
        return number +temp;
    };
}
```

# Non-static class data member

```
1 // 7- ERROR need to capture
2 class Data{
3 public:
4 int value;
5 int x;
6 void fun(){
7     [](){
8         // std::cout <<value<<std::endl;    // 7- ERROR need to capture
9     };
10    [this]() {
11        std::cout <<value<<std::endl;    // 8- this captured the class member by reference
12        value=12;
13    };
14    [x,this]() {    // 9- ERROR once you use this nothing before or after it
15        std::cout <<value <<std::endl;
16        value=12;
17    };
18    auto a=this->x;
19    auto b=this->value;
20    [&a,b]() {    // 10-Only objects with automatic storage duration can be captured by a
21        std::cout <<b <<std::endl;    //lambda in C++11 (i.e. local variables and function parameters). If you want the effect of capturing a
22        a=12;    //non-static class data member, you can either capture the this pointer as i
23    };
24 }
25 };
26 int main(){
```



# Memory Usage

```
2 void test(){
3     std::cout<<"Hello "<<std::endl;
4 }
5
6 int main(){
7     // Test d;
8     test();
9 }
```

```
int main(){
// Test d;
// d();
[ ](){
    std::cout<<"Hello "<<std::endl;
}();
}
```

```
moatasem@CAI1-L14000:~/vsomeIp$ !g++
g++ test.cpp
moatasem@CAI1-L14000:~/vsomeIp$ !size
size a.out
      text      data      bss      dec       hex filename
      2677       664       280      3621       e25 a.out
moatasem@CAI1-L14000:~/vsomeIp$ objdump -x --demangle | grep -i test
0000000000000000 1      df *ABS* 0000000000000000      test.cpp
000000000000123d 1      F .text 0000000000000019      _GLOBAL__sub_I__Z4t
00000000000011a9 g      F .text 0000000000000033      test()
moatasem@CAI1-L14000:~/vsomeIp$
```

```
moatasem@CAI1-L14000:~/vsomeIp$ !g++
g++ test.cpp
moatasem@CAI1-L14000:~/vsomeIp$ !size
size a.out
      text      data      bss      dec       hex filename
      2850       672       280      3802       eda a.out
moatasem@CAI1-L14000:~/vsomeIp$ objdump -x --demangle | grep -i lam
00000000000011ca 1      F .text 0000000000000037      main::{lambda()#1}::operator()() const
moatasem@CAI1-L14000:~/vsomeIp$
```

# Corner cases

## auto

```
moatasem@CAI1-L14000:~/vsomeIp$ g++ test.cpp -std=c++11 && ./a.out
test.cpp: In function 'int main()':
test.cpp:39:14: error: use of 'auto' in lambda parameter declaration only available with '-std=c++14'
   39 | auto fn= [](auto x, auto y){
      |           ^~~~~
//in C++14
auto fn= [](auto x, auto y){
    return x+y;
};
std::cout << typeid(fn(2,3)).name()<<std::endl;
std::cout << typeid(fn(2,3.5F)).name()<<std::endl;
std::cout << typeid(fn(2,3.5)).name()<<std::endl;
```

## Initialization

```
//C++14
[&v=temp,x=temp]() { // v by reference , x by value
    std::cout <<v<<std::endl;
    v=12;
}();
```

## mutable

```
int main(){
    int temp=10;

    [temp]()mutable{
        temp=15;
        std::cout <<temp<<std::endl;
    }();
}
```

No  
Misra  
Rules

# When we use it ?

## 1- passing function as argument    2- traceability    3- return function

C++ / Algorithm library

### std::sort

Defined in header <algorithm>

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );           (1) (until C++20)
template< class RandomIt >
constexpr void sort( RandomIt first, RandomIt last ); (since C++20)
template< class ExecutionPolicy, class RandomIt >
void sort( ExecutionPolicy&& policy,
           RandomIt first, RandomIt last );           (2) (since C++17)
template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp ); (until C++20) (3)
```

### std::find, std::find\_if, std::find\_if\_not

Defined in header <algorithm>

```
template< class InputIt, class T >
InputIt find( InputIt first, InputIt last, const T& value ); (1) (until C++20)
template< class InputIt, class T >
constexpr InputIt find( InputIt first, InputIt last, const T& value ); (since C++20)
template< class ExecutionPolicy, class ForwardIt, class T >
ForwardIt find( ExecutionPolicy&& policy,
               ForwardIt first, ForwardIt last, const T& value ); (2) (since C++17)
template< class InputIt, class UnaryPredicate >
InputIt find_if( InputIt first, InputIt last, UnaryPredicate p ); (until C++20)
template< class InputIt, class UnaryPredicate >
constexpr InputIt find_if( InputIt first, InputIt last, UnaryPredicate p ); (since C++20)
```

```
auto fn(){
    return [](int x){
        return x+1;
    };
}

int main()
{
    std::cout <<fn()(1);
    return 0;
}
```

# const

## 1- Linkage

```
c++ : Internal Linkage  
c:   External Linkage
```

```
*/
```

```
const int ll = 0;
```

```
int main()
```

```
{
```

```
    //*****2- Switch *****
```

```
    int var = MOATASEM;
```

```
    const int x = 97;
```

```
    switch (var)
```

```
{
```

```
    case x:
```

```
        std::cout << x << std::endl;
```

```
        break;
```

```
}
```

```
    //*****3- Array *****
```

```
    /*
```

```
        #define MAX_FOOS 10
```

```
        int foos[MAX_FOOS];
```

```
    */
```

```
    // const int max_foos = 12;
```

```
    // int foos[max_foos];
```

```
    // std::cout << std::size(foos) << std::endl;
```

```
    //*****4- With pointer *****
```

```
    // Error const with int
```

```
    const int var2 = 10;
```

```
    int *ptr = &var2;
```

# Constexpr vs const

```
const int var1 = 5;  
constexpr int var2 = 7;
```



```
int getRandomNo()  
{  
    return rand() % 10;  
}  
  
int main()  
{  
    const int varB = getRandomNo(); // OK  
    constexpr int varC = getRandomNo(); // not OK! compilation error
```



Value of **varB** would not anymore compile time. While statement with **varC** will throw compilation error. **The reason is constexpr will always accept a strictly compile-time value.**

# constexpr function

```
//  
constexpr int sum(int x, int y)  
{  
    x=1; // 1- Error in c++11 must be one statment  
    // std::cout<<"Hi"<<std::endl; //2- Error only const experssions  
    for(int i=0;i<3;i++) //3- aslong as experssions are not depend on runtime no problem  
    {  
        x+=i;  
    }  
    // for(int i=0;i<rand();i++){ //4- Error because of rand()  
  
    return x + y; //5- return must be const expr  
}  
  
int main()  
{  
    const int result = sum(10, 20); // 6- Here, you can use constexpr  
    cout << result;  
    return 0;  
}
```

```
19 {  
(gdb) disassemble /s  
Dump of assembler code for function main():  
test.cpp:  
19 {  
=> 0x0000000008001189 <+0>: endbr64  
0x000000000800118d <+4>: push %rbp  
0x000000000800118e <+5>: mov %rsp,%rbp  
0x0000000008001191 <+8>: sub $0x10,%rsp  
  
20     const int result = sum(10, 20); // Here, you can use constexpr as well  
0x0000000008001195 <+12>: movl $0x1e,-0x4(%rbp)  
--Type <RET> for more, q to quit, c to continue without paging--  
  
21     cout << result;  
0x000000000800119c <+19>: mov $0x1e,%esi  
0x00000000080011a1 <+24>: lea 0x2e98(%rip),%rdi # 0x8004040 <_ZSt4cout@@GLIBCXX_3.4>  
0x00000000080011a8 <+31>: callq 0x8001090 <_ZNSolsEi@plt>  
  
22     return 0;  
0x00000000080011ad <+36>: mov $0x0,%eax
```



**If there is no errors in constexpr scope but its argument workin on runtime dependent so it will be a normal function**

```
constexpr int result = sum(10, rand());
```

No  
Misra  
Rules

```
{  
  
    const int result = sum(10, rand());  
    cout << result;  
    return 0;  
}
```

```
moatasem@CAI1-L14000:~/vsomeIp$ g++ -g test.cpp -std=c++11
```

```
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t --demangle a.out | grep -i "sum"  
00000000000001247 w F .text 00000000000000038 sum(int, int)
```

```
0x000000000080011b1 <+8>: sub $0x10,%rsp
```

28           const int result = sum(10, rand());       // 5- Here, you can use constexpr as well

```
0x000000000080011b5 <+12>: callq 0x8001080 <rand@plt>
```

```
0x000000000080011ba <+17>: mov %eax,%esi
```

```
0x000000000080011bc <+19>: mov $0xa,%edi
```

```
0x000000000080011c1 <+24>: callq 0x8001247 <sum(int, int)>
```

```
0x000000000080011c6 <+29>: mov %eax,-0x4(%rbp)
```

# Tasks

- 1- create a function to find the maximum number of array
- 2- create a function to search to the number in the array which number is taken from user
- 3- delete number in array
- 4- merge two arrays together
- 5-find the even and odd numbers in the array

**Simple Lambda:** Write a lambda function to calculate the square of a given number.

**Sort with Lambda:** Use lambda functions to sort an array of integers in ascending and descending order.



# Develop a simple address book program that allows users to add, update, and search for contacts.

```
Welcome to your favorite address book!
What do you want to do?
    | List      | Lists all users
    | Add       | Adds an user
    | Delete    | Deletes an user
    | Delete all | Removes all users
    | Search    | Search or a user
    | Close     | Closes the address book

list
No contacts found
What do you want to do?
    | List      | Lists all users
    | Add       | Adds an user
    | Delete    | Deletes an user
    | Delete all | Removes all users
    | Search    | Search or a user
    | Close     | Closes the address book
```