Contents

I The Language **1**

1 Introduction **3**

1.1 BCPL to C 3
1.1.1 int, char 5
1.1.2 pointers 5
1.1.3 structure (struct) 7
1.1.4 || and && 7
1.1.5 unsigned, long, union, and *enum* types 7
1.1.6 Descendant of C language 8
1.1.7 What make C popular? 8
1.2 Standardization: ANSI C vs. ISO C 8
1.3 C89 or C90 9
1.3.1 <assert.h> 9
1.3.2 assignment 10
1.3.3 comments 11
1.3.4 'const' type qualifier 11
1.3.5 data types 14
1.3.6 'extern' 18
1.3.7 I/O library 21
1.3.8 international character sets 21
1.3.9 locale 21
1.3.10 typedef: define type alias 22
1.3.11 macros (pre-processors) 23
1.3.12 'static' keyword 24
1.3.13 'void' 25
1.3.14 'volatile' type specifier 25
1.3.15 variadic function 27
1.3.16 Using function 30
1.3.17 Using procedure 33
1.3.18 structural (loop/conditional) statements 33

DRAFT

1.3.19 non-structural statement . . . 33
1.3.20 namespace . . . 33
1.3.21 #define . . . 35
1.4 C94/C95 . . . 35
1.4.1 iso646.h . . . 35
1.5 C99 . . . 36
1.5.1 assertion . . . 36
1.5.2 Arrays . . . 36
1.5.3 comments . . . 36
1.5.4 (new) data types . . . 36
1.5.5 'inline' function . . . 38
1.5.6 'variadic' macro . . . 39
1.5.7 #pragma . . . 40
1.6 C11 . . . 41
1.6.1 Unicode . . . 42
1.6.2 _Noreturn . . . 42
1.6.3 multi-thread programming . . . 43
1.6.4 atomic operations . . . 43
1.7 POSIX C specification . . . 44
1.8 C Standard Library: implementation of the standard C language specification . . . 45
1.9 Simula to C++ . . . 47
1.10 C++98 . . . 48
1.10.1 C++ standard template library (STL) . . . 48
1.10.2 assert() in C++ . . . 50
1.10.3 zmq_assert() in zmq library . . . 50
1.10.4 containers . . . 51
1.10.5 – vector, dequeue, list . . . 51
1.10.6 – set, map . . . 51
1.10.7 – pair . . . 51
1.10.8 – hash_map, hash_set . . . 52
1.10.9 – multiset, multimap . . . 52
1.10.10 – user-defined containers . . . 52
1.10.11 'const' keyword . . . 52
1.10.12 name mangling: function overloading . . . 53
1.10.13 'inline' function . . . 54
1.10.14 macros . . . 54
1.10.15 ___cplusplus . . . 54
1.10.16 namespaces . . . 55
1.10.17 Template . . . 56

DRAFT

1.10.18 – how to write a template . . . 56
1.10.19 – type parameter, e.g. class template . . . 58
1.10.20 nothing . . . 61
1.10.21 – non-type parameter . . . 61
1.10.22 – Pre C++11 . . . 62
1.10.23 Allocators . . . 65
1.10.24 Iterators . . . 66
1.10.25 virtual function . . . 68
1.10.26 & symbol . . . 69
1.11 C++03 . . . 69
1.11.1 protected and static members . . . 69
1.11.2 inheritance from several classes . . . 71
1.11.3 'mutable' keyword . . . 71
1.11.4 new containers: map . . . 71
1.12 C++11 . . . 72
1.12.1 atomic functions . . . 72
1.12.2 'auto' keyword . . . 73
1.12.3 allocators . . . 74
1.12.4 attribute [[no return]] . . . 74
1.12.5 “no exception” specifier . . . 75
1.12.6 containers . . . 75
1.12.7 'inline namespace' . . . 75
1.12.8 Iterators . . . 75
1.12.9 lambda functions . . . 76
1.12.10 regular expression . . . 78
1.12.11 memory barrier (multi-threading) . . . 78
1.12.12 smart pointers . . . 79
1.12.13 move-only objects . . . 79
1.12.14 hash tables . . . 79
1.12.15 random number generators . . . 79
1.12.16 STL (standard threading library) . . . 79
1.12.17 new 'for loops' . . . 79
1.12.18 new container classes . . . 79
1.12.19 new macros . . . 79
1.12.20 raw string literal (verbatim string literal) . . . 79
1.12.21 variadic templates (C++11) . . . 80
1.12.22 union and array initialization lists . . . 80
1.12.23 new qualifiers (e.g. final, override, default) . . . 80
1.13 C++14 . . . 81
1.14 C++1z . . . 81

DRAFT

1.15 C++ Standard Library 82
1.16 Checking version of C/C++ specification 83
1.17 extern “C”: mixing C++ with C code 83
1.17.1 Mixing C++ code with C code 83
1.17.2 Calling C code from C++ 83
1.17.3 Make C++ codes callable to both C and C++ . . . 85
1.18 Comments in C vs. C++ 86
1.19 Scope in C: block scope, file scope and program scope 87
1.20 Scope in C++: block scope, file scope 89
1.21 Local variables in C vs. C++ 89
1.22 Global variables in C vs. C++ 90
1.23 Declaration vs. Definition of an identifier 90
1.24 Aliasing rules 94
1.25 Coding styles 97
1.25.1 Blink coding style 97
1.25.2 Chromium coding styles (Google C++ style) 97
1.25.3 header and source file 97
1.25.4 #pragma once vs. #ifndef (guards) 98
1.25.5 #pragma warning 99
1.25.6 #pragma managed 100
1.26 Tips and Tricks 101
1.26.1 Forward declaration 101
1.26.2 xvalue, glvalue, prvalue (C++11) 104
1.26.3 Rvalue and Lvalue 104
1.26.4 value type vs. reference type 107
1.26.5 delegate (VC++/CLI Windows) 107
1.26.6 Rule of Three (C++) 111
1.26.7 RAII (C++) 112
1.26.8 NUXI problem - little-endian or big-endian byte order 115
1.26.9 Calling conventions 116
1.26.10 C++98-ABI compatibility with C++11 116
1.26.11 Invalid conversion from void* to 118
1.26.12 Incomplete last line 118
1.26.13 Generic programming 119
1.26.14 mmap() 119

2 Memory Management by the O/S 121
2.1 Persistent memory vs. Non-persistent memory 122
2.2 Physical memory vs. Logical memory 122
2.3 Pagefile (Windows) vs. Swap space (Linux) 125

DRAFT

2.4 Thread Local Storage (TLS) . . . 128
2.4.1 C++ . . . 128
2.4.2 C# . . . 128
2.5 Memory in Win32 . . . 128
2.5.1 Virtual Memory . . . 129
2.5.2 Heap memory . . . 130
2.5.3 Shared memory . . . 130
2.5.4 File mapping . . . 130

3 X86 and X86-64 133
3.1 Machine code vs. Assembly code . . . 133
3.2 Instruction set . . . 138
3.3 X86 history . . . 138
3.3.1 Registers in CPUs . . . 141
3.3.2 RAM . . . 144
3.3.3 Instructions . . . 145
3.3.4 Clock cycles . . . 147
3.4 x86 function calls . . . 147
3.4.1 Floating-Point Unit (FPU) stack . . . 147
3.4.2 Calling Conventions . . . 149
3.4.3 Compiler Options . . . 152
3.4.4 Debuging information . . . 152
3.5 x86-64 . . . 153
3.5.1 Registers . . . 154
3.5.2 Instructions . . . 154
3.6 Manufactoring process: 22nm, 32nm ??? . . . 155
3.6.1 Chip design . . . 155
3.6.2 Wafer fabrication . . . 155
3.6.3 Die separation . . . 157
3.6.4 Packaging . . . 157
3.6.5 Testing . . . 157
3.7 Memory fences . . . 157
3.8 Memory alignment . . . 157
3.8.1 Order of scalar data definition . . . 158
3.8.2 Order of struct data definition . . . 159
3.8.3 Compiler reordering, memory racing . . . 160
3.8.4 alignof (C++11),std::alignment_of<T> (C++11) . 160
3.9 Physical vs. Virtual Memory (VM) . . . 163
3.10 Heap and Stack . . . 167
3.11 ABI . . . 172

DRAFT

DRAFT

3.12 X86-64 fuction calls . . . 173
3.12.1 Calling conventions . . . 174
3.12.2 Compiler Options . . . 175
3.13 Physical vs. Logical memory space . . . 175
3.14 Read-only memory . . . 180
3.15 Static memory space . . . 180
3.16 Bootstrap loader and OS . . . 181
3.17 User-mode and Kernel-mode . . . 182
3.18 Data alignment and granularity . . . 182
3.19 64-bit systems . . . 184
3.19.1 Alpha . . . 184
3.19.2 Xeon . . . 184
3.19.3 Virtual memory structure . . . 187
3.19.4 AMD's Jaguar . . . 187
3.20 Intel Itanium . . . 187
3.21 Intel Nehalem (45nm) . . . 189
3.22 * Intel Westmere (Nehalem-C) . . . 189
3.23 Sandybridge (32nm) . . . 190
3.23.1 uCop cache . . . 190
3.24 Ivybridge . . . 190
3.24.1 Ivybridge-EP . . . 191
3.24.2 Ivybridge-EX . . . 191
3.25 Intel Xeon . . . 191
3.25.1 5000 series . . . 191
3.26 Intel Xeon Phi processors . . . 192
3.27 Intel Xeon Scalable Processors . . . 193
3.28 SSE . . . 193
3.29 SSE2 . . . 193
3.30 SSE3 . . . 193
3.31 SSSE3 . . . 193
3.32 SSE4 . . . 194
3.33 AVX . . . 194

4 Library 195
4.1 Object file (.o, .obj) . . . 195
4.1.1 format of object file: COFF, ELF, PIE, DWARF . . . 196
4.1.2 assembly code . . . 197
4.1.3 relocatable code . . . 198
4.1.4 position-independent code (shared object files) . . . 200
4.2 Executable files . . . 201

4.3 Page Table . . . 205
4.4 What is a library? . . . 205
4.5 Libraries in Linux . . . 206
4.5.1 A static library (.a) . . . 206
4.5.2 A shared library (.so) . . . 207
4.5.3 CUDA shared library . . . 209
4.6 Libraries in Windows . . . 209
4.6.1 Static (.lib) . . . 209
4.6.2 Import library (.lib) . . . 209
4.6.3 Shared library (.dll) . . . 209
4.7 ELF . . . 210
4.8 PE executable format . . . 210
4.8.1 DLL . . . 211
4.8.2 .NET assembly . . . 211
4.9 PIE . . . 212

5 Locale - Internationalization - Globalization 213

6 Data types 217
6.1 Introduction . . . 217
6.1.1 aggregate (a special type of class): C++98/03 . . . 217
6.1.2 aggregate in C++11 . . . 219
6.1.3 Why we use aggregate data type . . . 219
6.1.4 Initialize data . . . 220
6.1.5 zero-initialized . . . 221
6.1.6 default-initialized . . . 221
6.1.7 value-initialized (since C++2003) . . . 221
6.1.8 POD type (Plain-Old Data type) . . . 221
6.1.9 Name lookup . . . 222
6.1.10 Qualified and unqualified types . . . 224
6.2 Macro values . . . 225
6.3 Header files . . . 226
6.3.1 stddef.h (cstddef) . . . 226
6.4 Type check . . . 227
6.4.1 typeof() . . . 227
6.4.2 decltype (C++11) . . . 227
6.4.3 typeid: <typeinfo> (C++11) - RTTI . . . 228
6.4.4 <type_traits> (C++11) . . . 230
6.5 Portable primitive integer data types . . . 230
6.5.1 Avoid using pre-defined data type . . . 235

DRAFT

6.5.2 Avoid using unsigned type 235
6.5.3 std::vector<int>::size_type vs. std::size_t 236
6.5.4 short, int, long, long long 236
6.6 Data of fixed size (C++11) 237
6.6.1 8-bit type 237
6.6.2 16-bit type 237
6.6.3 32-bit type 237
6.7 Boolean (BOOL, BOOLEAN, VARIANT_BOOL) 237
6.7.1 in Windows (Visual C++, C++/CLI) 237
6.8 Union 238
6.8.1 Initialize data 241
6.8.2 Usage 242
6.9 Struct 244
6.9.1 C 244
6.9.2 C++ 245
6.9.3 access specifier(s): public, private, protected 246
6.10 memory layout in struct in C++ and class 246
6.10.1 standard layout class (C++) 247
6.11 Struct vs. Class 247
6.12 Bit shift 249
6.13 Bitfields 250
6.13.1 C 250
6.13.2 – define a bitfield using struct 250
6.13.3 – define a bitfield using macros 251
6.13.4 – bitfield not portable 251
6.13.5 – bitfield padding (alignment): unnamed bitfield . . 252
6.13.6 – bit twiddling: combine bitfields 253
6.13.7 – extract bitfield 254
6.13.8 C++ 254
6.14 Enum 254
6.14.1 using #define 254
6.14.2 old-style enum (in C) 255
6.14.3 — major limitation of enum in C 257
6.14.4 * enum inside namespace 257
6.14.5 * enum inside class 258
6.14.6 typdef enum (in C) 259
6.14.7 old-style enum in C++ 259
6.14.8 enum class (C++11) 260
6.14.9 enum struct (C++11) 261
6.14.10 Good practice 262

DRAFT

6.14.11 - Default as a descriptive name 262
6.14.12 - map user-input (an integer) to a variable of a given enum 262
6.14.13 - index an enum 263
6.14.14 - traverse through an enum? 263
6.15 float vs. double 265
6.16 128-bit floating-point (long double) 265
6.17 Complex 266
6.17.1 C99 266
6.17.2 C++ 266
6.18 Software-emulated floating-point 267
6.19 sig_atomic_t 267
6.20 'const' 267
6.20.1 'const' with function definition 267
6.20.2 Positions of 'const' in relation to start (*) pointers and data type 269
6.21 Programming skils 270
6.21.1 Mixing 'char' and 'int' NOT RECOMMENDED . . 270
6.22 Windows Data Types 270

7 Array 273
7.1 array with compile-time size 274
7.2 (a pointer to an array) or (an array whose elements are pointer to. . .) 274
7.3 copy C-array 275
7.4 2D C-array (higher dimension) 275
7.5 maximum number of dimension in array 277
7.6 std::vector (C++, dynamics size) 277
7.7 VLA (size at compile-time) 277
7.8 array with alloca() 278
7.9 std::array (size known at compile-time): C++11 279
7.10 std::dynarray (C++14) 280
7.11 std::valarray (C++98) 280
7.12 Blitz++ 282
7.13 Loop throught high-dimensional array quickly 282
7.14 Element-wise operations 282

8 Data types for character and string 285
8.1 Character 285
8.1.1 MSE 285

DRAFT

8.1.2 locale . . . 286
8.1.3 C character . . . 286
8.1.4 * narrow-character . . . 286
8.1.5 * wide characters (wchar_t, wint_t) . . . 287
8.1.6 – wchar_t . . . 287
8.1.7 — wint_t . . . 288
8.1.8 literal character . . . 289
8.1.9 C++ character . . . 289
8.1.10 * C-backward compatible . . . 289
8.1.11 C++03: char . . . 290
8.1.12 * C++0x, C++11: char16_t and char32_t . . . 290
8.1.13 Escape character . . . 290
8.1.14 Utilities . . . 290
8.1.15 – char . . . 290
8.1.16 – wchar_t, wint_t . . . 291
8.1.17 in Windows: CHAR, TCHAR, _TCHAR, _TSCHAR, _TUCHAR, _TXCHAR . . . 292
8.1.18 in .NET: Char . . . 293
8.2 String in C . . . 293
8.2.1 narrow-character string (char *) . . . 293
8.2.2 – char* vs. char[] . . . 295
8.2.3 – char const* vs. char* const vs. const char* . . . 295
8.2.4 wide-character string . . . 296
8.2.5 string manipulation . . . 297
8.2.6 * string copy . . . 297
8.2.7 * concatenate: 2 strings . . . 298
8.2.8 * concatenate: >2 strings with sprintf(), snprintf(), asprintf(), string_format, (C++11) snprintf . . . 299
8.2.9 * remove trailing blank characters . . . 302
8.2.10 * search substring . . . 303
8.2.11 * split the string into tokens: single character delimiter 303
8.2.12 * split the string into tokens: string delimiter . . . 306
8.2.13 * compare 2 strings . . . 308
8.2.14 * work on character and a string . . . 308
8.2.15 * use a string (buffer) to transfer data . . . 308
8.2.16 * convert string and number . . . 309
8.2.17 *: Convert multibyte string from/to wide-char string 309
8.2.18 * length of a C string . . . 310
8.2.19 * loop through very characters in a C string . . . 310
8.2.20 format a C string . . . 311

DRAFT

8.2.21 NULL character (string terminator) 311
8.2.22 * check NULL-terminated string? 311
8.2.23 String I/O 311
8.2.24 Working with Unicode 312
8.3 String in C++ 312
8.3.1 C-style string 312
8.3.2 C++ strings: std::string, std::wstring 313
8.3.3 – std::string (std::basic_string) 313
8.3.4 – std::wstring 315
8.3.5 – QString (unicode-aware) 315
8.3.6 – C++11: u16string, u32string 315
8.3.7 string manipulation 316
8.3.8 * concatenate: 2 strings 319
8.3.9 * concatenate: > 2 string 319
8.3.10 * check character in string 319
8.3.11 * create an array of strings 320
8.3.12 * replace a substring in a string with a different content 320
8.3.13 * remove trailing blank characters 320
8.3.14 * split a string into an array of tokens 324
8.3.15 * compare 2 strings 325
8.3.16 * convert string to number 326
8.3.17 * convert a std::string to char* or std::vector<char> 326
8.3.18 copy a string 327
8.3.19 format a C++ string 328
8.3.20 * fastformat library 328
8.3.21 * stringstream: ostringstream, istringstream 328
8.3.22 ** reset a stringstream 329
8.3.23 * Boost::format() 329
8.3.24 * std::snprintf (C++11) 330
8.3.25 Loop through very characters in the C++ string . . 330
8.3.26 Intrinsic C vs. C++ strings 331
8.3.27 * length of a C++ string 333
8.3.28 * length of a string in Windows 333
8.4 Literal string (string literal) 334
8.4.1 raw form vs. cooked form 334
8.4.2 pointer assignment vs. string initialization 334
8.4.3 C++03: regular string literal 336
8.4.4 – Literal string wide-character (C++03) 337
8.4.5 C++11: regular string literal 338
8.4.6 – Raw string literal (C++11) 338

DRAFT

8.4.7 in Windows . . . 338
8.5 String in Windows C++ . . . 340
8.5.1 in Windows . . . 340
8.5.2 UNICODE_STRING . . . 346
8.6 String in .NET . . . 346
8.7 Covert string between C++ and C-style . . . 347
8.7.1 * const char* from std::string . . . 347
8.7.2 * std::string from const char* . . . 349
8.7.3 *: char* (TCHAR*) vs. CString . . . 349
8.7.4 * exception-safe approach . . . 350
8.8 <string.h>, <cstring>, and <string> . . . 351
8.9 Convert string from/to number . . . 352
8.9.1 from integer or floating-point . . . 353
8.9.2 to integer . . . 355
8.9.3 to floating-point number . . . 356
8.9.4 in Windows . . . 357
8.10 Special values in std::string . . . 359
8.10.1 npos . . . 359

9 C Pointers 361
9.1 Introduction . . . 361
9.1.1 Variable . . . 361
9.1.2 Pointers . . . 362
9.1.3 Pointers vs. Arrays . . . 363
9.1.4 'far', 'near' and 'huge' pointer . . . 363
9.2 Generic pointer (void *) . . . 364
9.3 NULL pointer . . . 365
9.4 Pointer constant vs. Constant pointer . . . 366
9.5 typedef's of pointers . . . 366
9.6 'restrict' pointer . . . 367
9.7 Pointers as function arguments . . . 367
9.8 Pointer vs. Array . . . 368
9.9 Pointer in 1D arrays . . . 370
9.10 Pointer in 2D arrays . . . 370
9.10.1 C . . . 371
9.10.2 C++ . . . 372
9.10.3 C++11 . . . 373
9.10.4 Jagged array . . . 373
9.11 Pointers in strings . . . 374
9.12 Pointer-to-pointer (C language) vs. Reference-to-pointer (C++) 374

DRAFT

9.12.1 Pointer to Pointers (C language) 375
9.12.2 Reference to pointer (C++) 376
9.13 Dynamic memory allocation 376
9.13.1 Boost::Pool (C++) 377
9.13.2 Understanding malloc() 377
9.13.3 C: malloc() or calloc() 378
9.13.4 C: realloc() 380
9.13.5 C: xalloc() 381
9.13.6 C: alloca() on stack 381
9.13.7 C: free() and dangling pointer 381
9.13.8 C++: notice 382
9.13.9 new operator (C++) 382
9.13.10 delete vs. delete[] 384
9.14 Type cast 384
9.14.1 In C 385
9.14.2 – (T)somevariable 385
9.14.3 In C++ 386
9.14.4 – T(somevariable) 387
9.14.5 – static_cast 388
9.14.6 – const_cast 389
9.14.7 – dynamic_cast 389
9.14.8 – reinterpret_cast 390
9.14.9 In C++/CLI 390
9.14.10 Tips 390
9.15 Print address 390
9.16 Increment a pointer 391
9.17 Size of an array (sizeof) 391
9.18 Potential mistakes 392
9.18.1 Reset array 392
9.18.2 Array assignment vs. Array copy 392
9.19 Copy memory using pointers 393
9.19.1 mem* function (memcpy, memmove) vs. std::copy, std::fill, std::equal 394
9.19.2 memcpy() 394
9.19.3 memmove 394
9.19.4 std::copy_n 395
9.19.5 std::copy 395
9.19.6 get/set 396
9.19.7 std::fill, std::fill_n 396
9.19.8 memset 396

DRAFT

9.19.9 String copy . . . 397

10 C++ Pointers 399
10.1 boost::shared_ptr (Boost library) . . . 399
10.2 vtkSmartPointer . . . 399
10.3 Smart pointer (C++11) . . . 399
10.3.1 auto_ptr (deprecated) . . . 399
10.3.2 unique_ptr<type> template (move-only objects) . . 403
10.3.3 shared_ptr (using a counter) . . . 405

11 Function and Procedures 407
11.1 Function prototype . . . 407
11.2 Optional parameters . . . 407
11.3 Overloaded functions . . . 408
11.3.1 C . . . 408
11.3.2 C1x . . . 411
11.3.3 C++ . . . 411
11.4 Restrictions on function declaration . . . 412
11.5 Pass by reference vs. Pass by pointer vs. Pass by value 412
11.5.1 TIPS: Passing pointer (* or **) with 'const' non-modified purpose . . . 413
11.6 Passing an array (i.e. a pointer) . . . 415
11.6.1 array with minimum length . . . 415
11.6.2 passing an array that cannot be changed . . . 416
11.6.3 1D-array . . . 416
11.6.4 2D-array . . . 416
11.6.5 3D-array . . . 418
11.7 Passing a string as argument to a function . . . 418
11.8 Return by reference or const reference . . . 419
11.9 Return an array or a pointer . . . 419
11.10Nested function . . . 422
11.11Function pointer . . . 423
11.11.1 a pointer to a function . . . 423
11.11.2 typedef a function pointer . . . 424
11.12Applications of function pointer . . . 425
11.12.1 modify function's body . . . 426
11.12.2 signal() function in C . . . 426
11.12.3 callback . . . 428
11.13How to end a program . . . 429
11.13.1 Failure/Success portable . . . 429

DRAFT

11.13.2 exit(), return . . . 430
11.13.3 abnormal termination . . . 430
11.13.4 Register some functions to be called on exit() . . . 431
11.13.5 normal termination . . . 431

12 Program Constructs 433
12.1 Loops . . . 433
12.1.1 finite loop . . . 433
12.1.2 ++ii or ii++ ? . . . 434
12.1.3 infinite loop . . . 434
12.1.4 ** while loop . . . 434
12.1.5 ** for_each loop . . . 435
12.2 Conditional statements . . . 436
12.2.1 'if' statements . . . 436
12.2.2 Short-hand conditions . . . 437
12.2.3 switch. . . case . . . 438
12.3 Logical operators . . . 439
12.3.1 Bitwise operator . . . 439
12.3.2 Condition for string . . . 439
12.3.3 Condition for function call . . . 440

II Object-Oriented 441

13 Programming Concepts 443
13.1 Exception handling . . . 443
13.2 Higher-order function (functor) . . . 444
13.2.1 mapping (transform) function . . . 446
13.2.2 zip (convolution) function . . . 446
13.3 Kalman Filter . . . 446
13.4 Event-based programming . . . 448

14 Object-Oriented (OO) Concepts 451
14.1 Class/Object/Object-oriented programming languages . . . 451
14.2 Inheritance . . . 451
14.2.1 Polymorphism . . . 451
14.3 Root class (object/Object) . . . 452
14.4 Interface vs. abstract class . . . 452
14.4.1 What can be in an Interface . . . 452
14.4.2 What can be in an abstract class . . . 453

DRAFT

14.5 Virtual constructor 453
14.6 Java annotation vs. C# Attributes 453
14.7 Mutable vs. Immutable 453
14.8 Trivial vs. Non-trivial special member function 454
14.9 V-table (virtual table) 455
14.10Naming style 455
14.10.1 data member 455
14.10.2 arguments 455
14.11registry 456
14.11.1 Implement registry in C++ 456
14.11.2 Implement registry in Python 456
14.11.3 Implement registry in Java 456
14.12Class method: overloading vs. overriding 456

15 Procedural programming 457

16 Functional paradigm 459

17 Object-Oriented C++ 461
17.1 namespace 461
17.2 Class 461
17.2.1 Friend of a class 462
17.2.2 * Friend function of a class 462
17.2.3 * Friend class of a class 462
17.2.4 std::cout friendly class 462
17.2.5 Nested class 463
17.3 Constructor and Destructor 464
17.3.1 constructor 464
17.3.2 * default constructor 464
17.3.3 * protected constructor 465
17.3.4 * virtual constructor??? 465
17.3.5 * 'explicit' constructor (with 1 parameter) 466
17.3.6 copy constructor 467
17.3.7 copy assignment (=) 467
17.3.8 move constructor (C++11) 467
17.3.9 destructor 468
17.3.10 * 'virtual' destructor 468
17.4 Using initialization list in constructor 469
17.4.1 * What data members can be initialized with initialization list? 470

DRAFT

17.4.2 * When to use initialization list . . . 470
17.4.3 * When NOT to use initialization list . . . 474
17.5 Methods (member functions) . . . 475
17.5.1 (non-virtual) member functions . . . 475
17.5.2 singlet method ??? . . . 476
17.5.3 protected data/method: 'protected' . . . 476
17.5.4 private data/method: 'private' . . . 476
17.5.5 static method: 'static' . . . 477
17.5.6 return by reference (T&) or const reference (const T&) . . . 478
17.5.7 const T& function-name(arguments) . . . 478
17.5.8 T& <function-name>(arguments) const . . . 479
17.5.9 const T& <function-name>(arguments) const . . . 479
17.6 Move assignment operator (C++11) . . . 479
17.7 Data members . . . 479
17.7.1 order of data members??? . . . 479
17.7.2 'static' data member . . . 480
17.7.3 Using pointer for data member . . . 481
17.7.4 Forward declaration for data member of an external type . . . 482
17.7.5 A reference data member . . . 482
17.8 Class Inheritance . . . 483
17.8.1 Base classes . . . 485
17.8.2 abstract class (data type) . . . 486
17.8.3 Public inheritance (default) . . . 487
17.8.4 Private inheritance . . . 488
17.8.5 Protected inheritance . . . 488
17.8.6 Object slicing . . . 488
17.8.7 Multiple inheritance . . . 490
17.8.8 Virtual base classes. Virtual inheritance . . . 492
17.9 Methods for class inheritance . . . 493
17.9.1 'virtual' member function . . . 493
17.9.2 – in base class . . . 494
17.9.3 – in derived class . . . 496
17.9.4 * override . . . 496
17.9.5 * augment . . . 497
17.9.6 * guarantee virtual method: override (C++11) . . . 497
17.9.7 pure virtual function . . . 498
17.9.8 'friend' class/method . . . 499
17.9.9 * friend class . . . 500

DRAFT