

INDEX

Symbols

- ! 114
- 114
- 114
- != 115, 121
- #define 13, 27, 207
- #endif 13, 27, 207
- #ifndef 13, 27, 207
- #include 13, 27, 103
- % 114
- %= 115
- & 114, 115, 122
- & operator 172, 174
- && 115
- &= 115
- () 114
- (type) 114
- * 114
- * operator 172, 174
- *= 115
- + 114
- ++ 114
- += 115
- . 114
- . * 114
- / 114
- /= 115
- < 114
- <> 103
- << 114
- <<= 115
- <= 115
- = 115
- = 106, 115
- == 115, 121
- > 114
- > 115
- > operator 256
- >* 114
- >= 115
- >> 114
- >>= 115
- ? : 115
- [] 114
- ^ 115, 122
- ^= 115
- | 115, 122
- |= 115
- ~ 114

Numerics

- 64 bit processor 104

A

- abstract base class 266, 420, 434
- abstract base classes 328, 416, 417
- abstract class 343
 - definition of 343
- abstract classes 343
- abstract data types 167
- abstract machine
 - targeting 83
- abstract thinking 7, 47
- abstract thought 84
- abstraction 83, 416
 - problem 7
- abstractions
 - getting them right 416
 - selecting the right kinds of 501
- access
 - member
 - horizontal 274
- access specifier
 - private 260, 274
 - protected 274
 - public 259, 274
- access specifiers 274, 328, 332, 362
 - private 289
 - protected 289
 - public 289
- accessing member elements via pointers 256
- accessor 374
- accessor functions 287, 331
- ad hoc polymorphism 372
- Ada 36
- adapters 406
- addition operator 117
- additive operators 117
- aggregation 302, 322
 - relationship to object lifetime 302
 - simple 305
 - simple and composite 302
 - simple vs. composite 302
 - two forms 322
- alert 109
- algorithm
 - growth rate 93
- algorithms 80, 91, 412
- American National Standards Institute (ANSI) 104
- analysis 50
- application
 - graceful recovery 48
- architecture
 - extension via inheritance 328
 - improving with three design principles 482
- arguments
 - passing to a function by reference 219
 - passing to a function by value 216
- array
 - accessing elements 186
 - addressing formula 185
 - arrays of pointers 188
 - automatic initialization of multi-dimensional 193
 - brace map 194
 - combining declaration and definition 187
 - danger of uninitialized elements 187
 - declaration
 - four-dimensional array 192
 - two-dimensional array 190
 - declaring 185
 - declaring two-dimensional 189
 - definition of 184
 - dimensional grouping at declaration 191
 - dynamic array declaration 196
 - dynamically allocated multi-dimensional 197
 - element addressing 184
 - locating array elements 185
 - memory representation of two-dimensional array 191
 - multi-dimensional 189
 - of function pointers 236
 - passing as argument to function

- 222
 - passing multi-dimensional to function 223
 - pointer arithmetic 187
 - representation in memory 184
 - single dimension
 - declaration of 185
 - strings 200
 - subscript method of element accessing 186
 - three and more dimensions 191
 - two-dimensional 189
 - types they can contain 185
 - use of braces 194
 - visualizing multi-dimensional 189
 - What is an array? 184
 - array elements 63
 - array processing 48
 - arrays
 - intro to 184
 - using constants to set array size 186
 - assembler 83
 - assembly 83
 - assembly language 83, 458
 - adding object module to C++ project 469
 - assembling 467
 - creating source file 467
 - inline assembly in Macintosh environment 469
 - inline in C++ function 465
 - issues to consider before using 465
 - linking object file 467
 - linking object modules with C++ 465
 - steps to create object file 467
 - to machine code translation 83
 - assignment operator 378, 394
 - assignment operators 124
 - associating an identifier with a memory location 168
 - asterisk
 - overloaded use of 173
 - placement in pointer declaration 172
 - attribute candidates 49
 - attributes 47, 48
 - auto increment and decrement operators 125
 - auxiliary storage devices 81
 - avoiding break and continue 158
- B**
- backslash 109
 - backspace 109
 - bad programming habits
 - avoiding 100
 - bad software architecture
 - characteristics of 500
 - base class 266, 328, 331, 362
 - base class attributes
 - initialized from derived class 337
 - base class constructor 328
 - base class constructor call 362
 - base class constructors
 - calling from derived classes 335
 - base class pointer 420
 - base class pointers 416
 - base classes
 - preventing multiple instances of 360
 - behavior 328
 - defining common in base classes 328
 - predictable object 446
 - Bertrand Meyer 482, 497
 - Bertrand Meyer's Design by Contract (DbC) 483
 - Bertrand Meyer's Design by Contract programming 482
 - binary 26
 - binary instructions 26
 - bit 89
 - bitwise AND operator 121
 - bitwise exclusive OR operator 122
 - bitwise inclusive OR operator 122
 - block statement 143
 - bool 104
 - boolean literals 111
 - branch link register 101
 - break 157
 - buses 81
 - byte 90
 - byte addressable memory 90
- C**
- C 36
 - C library function header 459
 - C library function implementation file 459
 - C library routines
 - adding files to CodeWarrior project 460
 - CodeWarrior project target settings 461
 - compiling CodeWarrior project 462
 - using 462
 - C vs. C++
 - important difference 458
 - C++ 36
 - power and danger of 484
 - C++ & Java 470
 - C++ class construct 274
 - C++ implementation dependencies 465
 - C++ Man 167
 - C++ preprocessor 27
 - C++ program
 - smallest 100
 - C++ standard 19
 - C++ Standard Template Library 412
 - C.A.R. Hoare 233
 - cache 83
 - cache memory 88
 - callback function 237
 - callback functions 237
 - called routine 27
 - calling conventions 27
 - calling routine 27, 101
 - carriage return 109
 - case label 150
 - cfloat.h 104
 - challenges 4
 - char 104
 - character 104
 - character literals 108
 - cin 134
 - CISC 83
 - CIWS class 345
 - class 266, 274
 - constants 275
 - declaration 330
 - declaring new data types 274
 - class attributes 278
 - class behavior 322
 - class constants
 - initializing via initializer list 280
 - class declaration
 - body 275
 - don't forget the semicolon! 275
 - how to name classes 274
 - minimum 275
 - opening and closing braces 275
 - parts of 274
 - placement of public functions 275
 - class declarations
 - viewed as behavior specifications 484
 - class diagram 274, 276, 506
 - compartments 276
 - indicating private class members 276
 - indicating public class members 276
 - class diagrams
 - adding attributes in Describe™ 517
 - adding operations in Describe™ 517
 - creating in Describe™ 515
 - Describe™ properties editor window 518
 - class invariant 484, 486
 - defined 484

- class invariants 484
 - class member functions 276
 - class template 398
 - class type 362
 - classes 263
 - difference between classes and structs 265
 - introduction to 263
 - polymorphic 417
 - static relationship modeling 506
 - limits.h 104
 - code reuse 328
 - CodeBuilder 36
 - CodeWarrior
 - disassembly tool 101
 - COFF library file format 467
 - cohesion 19
 - comma operator 125
 - comment clutter 17
 - comments 103
 - bad use of 17
 - c++-style 17
 - creating in source code 103
 - C-style 103
 - c-style 16, 17
 - self commenting code 17
 - source code 16
 - communication devices 81
 - compiler 26, 27
 - compiler errors
 - caused by missing class declaration semicolon 275
 - fixing 11
 - complex application behavior 302
 - complexity
 - conceptual 416
 - managing physical 302
 - physical
 - on larger projects 309
 - complexity management 291
 - composite aggregate
 - definition 302
 - composite aggregation 322
 - example 303
 - compound assignment operators 125
 - compound statements 143
 - computer
 - definition of 80
 - general purpose machine 80
 - vs. computer system 80
 - computer architecture 83
 - computer program 250
 - computer system 80
 - bus 81
 - definition of 80
 - interface components 81
 - main logic board 81
 - main logic board block diagram 81
 - system unit 81
 - typical 80
 - computers 80
 - conceptual complexity 416
 - concrete class 343
 - concrete classes 434
 - conditional operator 123
 - constant 49, 100
 - constants 103, 128
 - declaration 100
 - naming convention 18
 - constructor 275, 278, 331
 - initializer list 279
 - purpose 279
 - constructors 274
 - containers 406, 412
 - continue 158
 - control flow 142
 - control statement usage guide 160
 - copy
 - deep 379
 - deep vs. shallow 379
 - shallow 379
 - copy assignment operator 282
 - purpose 283
 - purpose of 449
 - copy constructor 281
 - example code 281
 - purpose 281
 - purpose of 449
 - coupling 19
 - cout 135
 - Cplusplusified C code 482
 - creating objects 167
 - creating objects via definition 168
 - C-style comment 103
 - current position 50
 - CVS 507
- ## D
- dangling reference 229
 - data 250
 - data encapsulation 267, 274, 289
 - data modeling 250
 - data structure 49
 - data structures 48
 - data types
 - determining size with sizeof 106
 - DbC 483
 - debugger 27
 - decimal 107
 - decimal integer literal 106
 - declaration statement 142
 - declaring
 - overloaded functions 231
 - decode 88
 - deep copy 379
 - default constructor 278, 279
 - definition 279
 - purpose of 449
 - delete 114
 - delete operator 176
 - dependency inversion principle 482, 500
 - dependency inversion principle (DIP) 328
 - derived class 266, 328, 362
 - derived class initializer list 328
 - derived class object 362
 - derived class objects 416
 - Describe™ 276, 506
 - feature set 507
 - Design by Contract 483
 - destructor 275, 283, 331
 - purpose 284
 - purpose of 449
 - virtual 330
 - destructors 274
 - effects of non-virtual destructors
 - 341
 - virtual 340, 341
 - development cycle
 - applying 45
 - code 45
 - deploying 45
 - factor 45
 - integrate 45
 - plan 45
 - test 45
 - using 45
 - direction 49
 - disassemble 101
 - disassembled code listings 101
 - disassembly 101
 - division operator 116
 - DLL 293
 - creating Mac OSX DLL with GNU C++ compiler 476
 - do statement 155
 - document-as-you-go 511
 - doing something forever 152
 - DOORS™ 507
 - dot operator 255
 - double 105
 - double quote 109
 - Dr. Barbara Liskov 483
 - Dr. Bertrand Meyer 483
 - Droning Professor 166
 - dumb sort 92
 - dynamic linked library 293
 - dynamic object creation 174, 420
 - dynamic polymorphic behavior 362
 - dynamic polymorphism 416
 - complex example 422
 - defined 417
 - example 419
 - dynamically allocated resources 379

E

- Eiffel 483
- electronic pathways 81
- Emacs 37
- Embarcadero Technologies 507
- empty project
 - creating in CodeWarrior 459
- encapsulation 267
- enum 250, 252, 254
 - default state values 252
 - state name conflicts 253
- enumerated data types 252
- enumerated types 250, 252
- enums 252
- equal To operator 121
- equality operators 121
- error checking 48, 70
- error handling
 - trapping bad input 134
- errors
 - compiler 11
- escape sequence table 109
- escape sequences 109
 - using in strings 111
- essentiality 416
- ethernet 81
- evolving software architecture 458
- Example 392
- executable image 26
- executable program 84
- execute 88
- EXIT_SUCCESS 133
- explicit template specialization 451
- expression forms
 - additive operators 112
 - assignment operators 113
 - bitwise AND operator 113
 - bitwise exclusive OR operator 113
 - bitwise inclusive OR operator 113
 - comma operator 113
 - conditional operator 113
 - constant expressions 113
 - equality operators 113
 - explicit type conversion 112
 - logical AND operator 113
 - logical OR operator 113
 - multiplicative operators 112
 - pointer-to-member operators 112
 - postfix 112
 - class member access 112
 - const cast 112
 - decrement 112
 - dynamic cast 112
 - explicit type conversion 112
 - function call 112
 - increment 112
 - pseudo destructor call 112
 - reinterpret cast 112
 - static cast 112
 - subscripting 112
 - type identification 112

- primary 112
- relational operators 113
- shift operators 113
- unary 112
 - decrement 112
 - delete 112
 - increment 112
 - new 112
 - sizeof 112

expressions 103, 111

- composition 111
- table of expression forms 112

expression-statements 142

extended mnemonic 101

extern “C” language linkage specification 478

extern keyword 458

F

- false 111
- fetch 87, 88
- file scope 130
- firewire 81
- fleet simulation 343
- float 105
- floating point 104
- floating point arithmetic 83
- floating point literals 110
- floor 50
- flow 9
 - achieving 9
 - concept of 9
 - stages 9
- for statement 155
- force multipliers
 - abstraction and inheritance 416
- form feed 109
- Fortran 36
- frustrations 4
- function
 - array of pointers to 236
 - callback via function pointer 237
 - called function responsibilities 216
 - calling 209
 - calling function responsibilities 216
 - characteristics of a well written 208
 - code module 206
 - declaration 209
 - declaring 208
 - default argument values 227
 - defining 208
 - definition 209
 - definition of 206

- definitions 207
- description of 206
- function parameter scope 214
- grouping related statements into 206
- hiding 337
- hiding global variables 212
- implementation files 207
- interface vs. implementation 207
- local variable scoping 212
- member function 206
- naming 208
- overloading 231, 328, 337
- overriding 328, 420
- passing arguments by reference 219
- passing arguments to 215
- passing arrays as arguments 222
- passing multi-dimensional arrays to 223
- passing pointers as arguments 220
- passing references 221
- placing declarations in header files 207
- pointer
 - syntax 235
- pointers 234
- purpose of 206
- recursion 232
- recursive calling 232
- return keyword usage 226
- return values 225
- returning objects 226
- returning pointers 227
- returning reference 229
- returning void 225
- signature 231
- static variables 213
- use of scoping blocks 212
- using return values 225

function access rights

- as preconditions 496

function argument types

- as preconditions 494

function declaration in header file 207

function libraries 207

function library 239

- steps to creating 239

function overloading 328

function overriding 416

function parameter scope 214

function parameters 212

function pointer

- array of 236
- assigning the address of a function to 235
- calling function via 236

function pointers 234, 235, 237

function return types

- as postconditions 496
- function return values 225
- function scope 130
- function signature 231, 340, 362
- function signatures 290
- function stubbing 11, 56
- function template 398
- functions 48, 230
 - accessor 274
 - class members 276
 - declaring pure virtual 342
 - declaring virtual 340
 - introduction to 206
 - main() 100
 - member functions 206
 - multiple with same name 231
 - mutator 274
 - naming convention 18
 - overloading 231, 289, 372
 - protected 275
 - pure virtual 342
 - purpose of pure virtual 342
 - virtual 340
- fundamental data types 250
- fundamental language features 48
- fundamental types 104
 - determining value ranges 105
 - range of values 104
 - value range table 105

G

- G4 82
- G4 7400 82
- generalized component behavior 434
- generic class definition 398
- generic code 398
- generic function 398
- global variable
 - hiding with local variables 212
- global variables 131
- GNU C++ compiler 476
- GNU C++ compiler command 38
- good software architecture
 - characteristics of 501
- goto statement 159
- greater than operator 120
- greater than or equal to operator 121
- growth rate 93

H

- hardest thing about learning to program 4
- hardware 83
- header file 14, 207
 - structure of 207
 - typical 207
- header files
 - contents of 14

- heap 175
- heat sink 82
- heterogeneous objects 184
- hexadecimal 107
- hexadecimal escape sequences 110
- hiding outer block variables 213
- high-level program modules 52
- homogeneous objects 184
- horizontal access 330
 - concept 289
 - controlling via access specifiers 288
- horizontal member access 274
- horizontal tab 109
- HP 36
- hungarian notation 126

I

- IComponent class 422
- IDE 27
 - creating new project 28
 - creating project with Codewarrior 28
 - project 28
 - project stationary 28
 - tracking project changes 35
- identifier
 - naming conventions 126
- identifiers 126
- if statement 143
- if statements and compound statements 144
- if-else statement 145
- IMDE 506
- implementation file
 - contents of 15
- incremental code evolution 361
- inheritance 266, 331, 332, 362, 416
 - benefits 328
 - effects of using different types 332
 - multiple 353
 - private 333
 - protected 333
 - public 333
 - purpose and use 328
 - uses of 361
 - virtual 357
- inheritance hierarchy 422
- inheriting 328
- initializer list 274
 - definition 279
 - example code 279
- initializer lists 274
 - line wrapping for readability 280
- input error checking 134
- input/output 83
- insertion operator 375
- instance attributes 278

- instruction 83
- instruction decode 87
- instruction execution 87
- instruction execution stages 82
- instructions 26, 250
- int 105
- integer 104
- integer literals 107
- Integrated Development Environment 27
- Integrated Modeling and Development Environment 506
- integration
 - robot rat project 60
- interface 266, 311, 434
 - benefits of separating 12
 - separating from implementation 302
 - separation from implementation 290
 - separation of 12
- interface components 81
- interface vs. implementation 207
- internal hard disk drives 81
- invoking a method on an object 267
- iostream 54, 103, 374
- iostream.h 103
- iteration statements 150
- iterators 412

J

- Java 36
 - compiling Java source file 472
 - creating Java source file 471
 - steps to create JNI C++ program 470
 - using javah command line tool 473
- Java & C++ 470
- Java Native Interface
 - creating C++ Win32 DLL 474
 - creating header file 472
 - Mac OSX example 475
 - Win32 JNI example 471
- Java Native Interface (JNI) 458, 470
- javah command line tool 473

K

- keyboard 80, 81
- keywords 104
 - listing 104
 - reserved identifiers 104
- Knuth 233

L

- labeled statements 159
- language features 44, 50
 - strategy area check-off list 51

- language linkage specification 458
 - left shift operator 118
 - legacy application code 458
 - legacy C code 458
 - using 458
 - less than operator 120
 - less than or equal to operator 121
 - libraries 103
 - adding to programs 103
 - library
 - function
 - creating 239
 - library code written in C 458
 - library routines
 - building in C 458
 - steps to creation in C 459
 - limits.h 104
 - link register 101
 - linker 26, 27
 - Linux 36
 - Liskov Substitution Principle 482
 - relationship to Meyer Design by Contract Programming 483
 - three rules of 497
 - Liskov Substitution Principle (LSP) 328, 483
 - literal
 - hexadecimal 107
 - octal 107
 - literals 106
 - boolean 111
 - characters 108
 - decimal 107
 - floating point 110
 - dissected 110
 - parts of 110
 - integer 107
 - multiple characters 108
 - single characters 108
 - strings 111
 - using escape sequences in character literals 109
 - load immediate 101
 - local function variables 212
 - local variable 212
 - local variables 212
 - declaring 212
 - logical AND operator 123
 - logical OR operator 123
 - long double 105
 - long int 105
 - long long int 105
 - looping
 - forever 152
 - lower-level modules 52
 - LSP 483
 - LSP & DbC
 - C++ support for 483
 - common goals 483
 - designing with 483
 - lvalue vs. rvalue 124
- ## M
- machine code 26, 83
 - machine instructions 26
 - MachTen 36
 - Macintosh OSX 476
 - Macintosh™ 100, 506
 - main file
 - main() function 15
 - main logic board 81
 - main memory 81
 - main() function 100, 103, 132
 - calling functions upon exit 133
 - disassembled 100
 - EXIT_SUCCESS 133
 - exiting 133
 - purpose 132
 - returning integer value 100
 - two forms 132
 - makefile 37, 38
 - contents of 38
 - creating 38
 - dependencies 38
 - for firstprog 38
 - managing physical complexity 291
 - member access via this pointer 278
 - member function 206
 - member function access 278
 - member function overloading 274
 - member functions
 - access to class attributes 278
 - memory 81
 - addressability 90
 - alignment 90
 - arrangement 168
 - boundaries 90
 - cache 89
 - miss performance penalty 89
 - contiguous allocation for array 184
 - non-volatile 88
 - stack and heap 175
 - sub systems 89
 - volatile 88
 - memory address of objects 169
 - memory addressing 169
 - memory hierarchy 89
 - memory organization 88
 - memory region
 - size occupied by an object 168
 - menu 49
 - Mercury Test Director™ 507
 - methods rule 497
 - Metrowerks CodeWarrior 28, 401, 459
 - adding files to projects 30
 - Groups 29
 - projects 29
 - running projects 31
 - setting project search paths 29
 - Sources group 29
 - microphones 81
 - microprocessor 82
 - Microsoft Visual C++ 32, 507
 - creating new workspace 32
 - StdAfx.h 32
 - Workspace 32
 - millions of instructions per second 82
 - minimal C++ program 100
 - minimal development environment 27
 - MIPS 82
 - mnemonic
 - extended 101
 - model 47
 - modeling 47
 - modem 81
 - modulus operator 117
 - monitor 80
 - mouse 80, 81
 - multi-file projects 12
 - multi-file variable usage 131
 - multiple character literals 108
 - multiple file project 48
 - multiple file projects 48
 - multiple inheritance 353
 - multiplication operator 116
 - multiplicative operators 116
 - mutator 374
 - mutator functions 287
- ## N
- name mangling 458
 - name spaces 253
 - name table 101
 - naming
 - functions 208
 - naming convention
 - sticking with 19
 - naming conventions
 - adopting 19
 - concepts of 17
 - constants 18
 - functions 18
 - hungarian notation 126
 - variables 18
 - nested while loop 153
 - nesting for statements 157
 - nesting if-else statements 146
 - nesting while statements 152
 - new 114
 - new class types 328
 - new line 109
 - new operator 175
 - NeXT 36
 - non-static class attribute 278

- non-virtual inheritance 360
 - not an lvalue... 124
 - not equal to operator 121
 - noun 49
 - nouns 48, 49
 - mapping to data structures 49
 - null statements 142
 - null termination of strings 111
 - numeric_limits 105, 106
 - how to use 105
 - numeric_limits<> 106
- O**
- object 266, 274
 - accessing via pointer 173
 - addressing 169
 - behavior 446
 - definition 167
 - dynamic creation using new operator 174
 - state and behavior 276
 - three ways to create 167
 - object behavior 276
 - object code 26
 - object creation 167
 - object interaction 84
 - Object Management Group 506
 - object modules 27
 - object state 276
 - Objective-C 36
 - object-oriented analysis & design 267
 - object-oriented architecture
 - extending 482
 - preferred characteristics 482
 - reasoning about 482
 - understanding 482
 - object-oriented design 7
 - reasoning about 361
 - object-oriented programming 416
 - C++ language feature support for 417
 - defined 417
 - object-oriented thinking 266, 274
 - ObjectPlant™ 276, 506
 - objects 331
 - accessing via address 169
 - assignment 447
 - copying 447
 - creation 446
 - destruction 447
 - lifetime control in composite aggregates 322
 - other design usage contexts 447
 - usage contexts 446
 - observable behavior 84
 - definition 84
 - OCCF 448
 - OCCF class functions 449
- OCP 498
 - defined 498
 - example 498
 - octal 107
 - octal escape sequences 110
 - ofstream 374
 - overloading 374
 - OMG 506
 - open-closed principle 482, 497
 - achieving 498
 - conventions 498
 - relationship to the LSP and DbC 498
 - open-closed principle (OCP) 328
 - operands 101
 - operating system 80, 100
 - operator 100, 176, 177, 184
 - insertion 375
 - operator * 186
 - operator overloading
 - cool things C++ programmers can do 372
 - goal 372
 - in the context of your design 372
 - operator precedence 114
 - operators 113
 - > 256
 - addition 117
 - additive 117
 - assignment 124
 - division 116
 - equal to 121
 - equality 121
 - greater than 120
 - greater than or equal to 121
 - left shift 118
 - less than 120
 - less than or equal to 121
 - modulus 117
 - multiplication 116
 - multiplicative 116
 - not equal to 121
 - overloading 372, 373
 - precedence table 114
 - right shift 119
 - shift 118
 - shorthand member access 256
 - subtraction 118
 - optimized routine 458
 - organizational complexity 54
 - orthodox canonical class form 448
 - extending 451
 - four required functions 448
 - ostream 374
 - overloadable operators 372
 - overloaded function
 - defined 231
 - return types 231
 - overloaded operator 394
- overloaded operators
 - leading to cleaner code 372
 - parameter list 373
 - overloading
 - arithmetic operators 383
 - assignment operator 378
 - comma operator 392
 - compound assignment operator 386
 - constructors 279
 - decrement operator 387
 - equality operator 381
 - function operator 392
 - increment operator 387
 - iostream operators 374
 - member operator 392
 - relational operators 380
 - subscript operator 384
 - virtual overloaded operators 392
 - overloading functions 289
 - overloading iostream operators 374
- P**
- parentheses 115
 - using to control operator precedence 115
 - part class 302
 - pass by value 216
 - pen 49
 - Perplexed One 166
 - pipeline 82
 - pipelined instructions 82
 - placeholders
 - in templates 398
 - placement of the asterisk 172
 - Plant class 343
 - pointer
 - accessing object via 173
 - array of 188
 - callback functions 237
 - declaration
 - use of asterisk 172
 - declaration of 172
 - definition of 171
 - dereferencing 173
 - how not to return from a function 229
 - using to access array elements 187
 - vs. reference 177
 - pointer arithmetic 187
 - pointer dereferencing 173
 - pointer vs. reference 177
 - pointers 166, 328, 340
 - base class to derived class objects 340
 - to functions 234
 - uses of 166
 - what are they good for? 166

- polymorphic class 340
 - polymorphic classes 417
 - polymorphism
 - ad hoc 372
 - defined 417
 - dynamic 416, 436
 - static 398, 402, 412
 - post condition
 - example 485
 - postcondition 486
 - defined 484
 - postconditions 484
 - changing in derived class functions 493
 - power supply 81
 - PowerDesigner™ 506
 - PowerPC 100
 - PowerPC G4 82
 - PowerPC mnemonic 101
 - PPC Std C++ Console Settings 101
 - precondition 486, 487
 - defined 484
 - example 485
 - preconditions 484
 - changing preconditions of derived class functions 488
 - weakening 489
 - predictable object behavior 446
 - preprocessor directive 103
 - preprocessor directives 13, 27
 - behavior of 14
 - previously developed assembly modules 458
 - private 328, 332, 334
 - private access specifier 260
 - problem abstraction 7, 84
 - problem domain 6, 44, 48
 - procedural design 7
 - procedural programming paradigm 49
 - processing cycle 87
 - processor 81, 82
 - program
 - definition of 83
 - two aspects 83
 - program creation process 26
 - program termination
 - indicating with result code 100
 - program transformation process 85
 - programming 4
 - programming as art 4
 - programming cycle 10
 - code 10
 - factor 10
 - integrate 10
 - plan 10
 - repeating 11
 - summarized 11
 - test 10
 - programming skills required 4
 - programs 80
 - project approach strategy 6
 - design 7
 - in a nutshell 7
 - language features 7
 - problem domain 6
 - requirements 6
 - strategy areas 6
 - using 44
 - project complexity
 - managing 11
 - project file format 14
 - project objectives 47
 - project requirements 6
 - project specification 49
 - project stationary 28
 - projects
 - multi-file 12
 - properties rule 497
 - protected 328, 332, 334
 - protected function 331
 - protected functions 275
 - protocol 101
 - pseudocode 69, 70
 - public 328, 332, 334
 - public access specifier 259
 - public interface functions 329
 - pure abstraction 416
 - pure virtual function 342, 420
 - pure virtual functions 328, 434
 - pure virtual member functions 416
 - purpose of a constructor 279
- Q**
- question mark 109
 - quicksort 233
- R**
- RAM 88
 - random access memory 88
 - Rational Rose™ 506
 - recursion 232
 - creating recursive functions 232
 - stopping 232
 - recursive function calls 232
 - recursive functions 232
 - Reduced Instruction Set Computer (RISC) 82
 - reference 177
 - declaration and use of 178
 - returning from function 229
 - region of storage 167
 - register usage 83
 - relational operators 120
 - reliable object-oriented software
 - creating 483
 - requirements 6, 44
 - gaining insight through pictures 49
 - requirements gathering 6
 - resolving C function library names 458
 - result code 100
 - result store 87
 - return 133
 - return keyword 226
 - reverse engineering
 - merging two systems after 528
 - using Describe™ 527
 - right shift operator 119
 - RISC 82
 - Robot Rat 507
 - robot rat
 - version two design issues 513
 - version two project specification 508
 - robot rat project specification 46
 - analyzing 47
- S**
- scope 128
 - creating scope blocks with {} 129
 - file 130
 - function 130
 - scope blocks 129
 - scope resolution operator 278
 - scoping blocks 212
 - scoping rules 212
 - Sedgewick 233
 - selection statements 142, 143
 - if 142
 - if-else 142
 - switch 142
 - semicolon
 - placement of 275
 - semicolon as sequence point 142
 - sending a message to an object 267
 - sentinel values 151
 - separating interface from implementation 290
 - sequence diagram 308
 - sequence diagrams
 - adding messages to 523
 - creating in Describe™ 522
 - sequence point 142
 - shallow copy 379
 - shift operators 118
 - short int 105
 - shorthand member access 255
 - shorthand member access operator 256
 - side 142
 - signature rule 497
 - signed char 104
 - signed int 105
 - signed long int 105
 - signed long long int 105
 - signed short int 105
 - simple aggregate

- definition 302
- simple aggregation 322
- simple escape sequences 109
- simplifying assumptions 309
- single quote 109
- sizeof 106, 114
- smallest C++ program 100
- software 83
- software development roles 5
 - analyst 5
 - architect 5
 - programmer 6
- software modules 458
- software systems 458
- source code
 - generating in Describe™ 525
- source code control systems 507
- speakers 80, 81
- special member functions 274
 - default behavior 286
 - four types 278
- special register 101
- stable application architecture 328
- stack 175
- stack and heap memory 175
- stack frame 212
- stages of execution 82
- standard C++ console application 54
- standard libraries 83
- Standard Template Library 405
- state transition diagram 65, 66
- statement 100
 - declaration statement 142
 - goto 159
 - termination with semicolon 142
- statements 103, 142
 - block statement 143
 - break 157
 - compound statement 143
 - continue 158
 - do 155
 - difference from while 155
 - nesting 155
 - examples of 142
 - for 155
 - nesting 157
 - written as a while statement 156
 - goto
 - usage advice 159
 - if 143
 - if-else 145
 - nesting 146
 - iteration 150
 - labeled 159
 - looping forever 152
 - null statement 142
 - selection statements 143
 - side effects of 142

- switch 147
 - break and the default case 150
 - case labels 147
 - default label 147
 - use of break keyword 148
- using compound statements 143
- while 150
 - exiting with break 152
 - exiting with exit() 153
 - nesting 152
 - using a switch statement 153
 - using sentinel value 151
- static function variables 213
- STL 405, 412
 - adapters 406
 - algorithms 408
 - containers 406
 - iterators 407
 - list 411
 - vector 408
- STL components 398
- store 88
- strategy
 - project approach 6
- stream operators
 - overloaded 376
- strengthening preconditions 491
- string
 - array 111
 - literals 111
 - null termination 111
- string literals 111
 - assigning to an array 111
- strings 200
- struct 254
- structs
 - accessing members via pointer 255
- structure
 - member definition format 262
 - template 398
- structures 254
 - accessing elements within 255
 - C++ Style 259
- stubbing 11, 56
- subclass 266, 328
- subtraction operator 118
- SUN 36
- superclass 266, 328
- superscalar 82, 83
- supertypes & subtypes
 - reasoning about 483
- switch statement 147
- switch statements 252
- system bus 81
- system software 80
- system unit 80, 81

T

- table of contents 101
- template
 - class 398
 - complex class example 404
 - definition of 398
 - function 398
 - member function definition 403
 - placeholders 398
- template declaration
 - grouping declaration and implementation 400
- templates
 - declaring class templates 403
 - how they work 398
 - special template definition syntax 402
 - structure 398
 - type placeholder 399
 - using multiple type placeholders 400
- Tenon Intersystems
 - MachTen CodeBuilder 36
- testing
 - robot rat project 55
- text books, references, & quick reference guides
 - difference between 19
- text editor 27
- the art of programming 4, 8
 - inspiration 8
 - money but no time 8
 - mood setting 9
 - time but no money 8
 - where not to start 8
 - your computer 8
- this pointer 262, 278
- tight spiral development 45
- TOC 101
- top-down functional decomposition 52
- trapping bad input 134
- trigraphs 85
- true 111
- type definition 250
- type synonym 250
 - reason for creating 250
- typedef 250, 254
 - example usage 250
- typedef keyword 250

U

- UML 302, 307, 506
 - class diagram 276, 322, 328
 - diagram object
 - linking in Describe™ 529
 - diagrams 507
 - linking diagrams in Describe™ 511

- modeling tool
 - purpose of 506
 - two important roles played by 506
 - sequence diagram 308, 322
 - web reports
 - creating with Describe™ 530
 - UML modeling tools 506
 - Unified Modeling Language 274, 506
 - UNIX 36
 - creating makefile 37
 - make utility 37
 - makefile 37
 - unsigned char 105
 - unsigned int 105
 - unsigned long int 105
 - unsigned long long int 105
 - unsigned short int 105
 - usage for two dimensions 190
 - use case diagrams
 - adding documentation to 511
 - creating in Describe™ 509
 - programmer perspective 512
 - robot rat use case diagram 510
 - using directive 103
 - utility software 80
- V**
- variable 49, 100
 - scoping 100
 - variable naming 17
 - variables 103, 128
 - declaring 128
 - function scope 128
 - global 19
 - limiting variable scope 131
 - local scope 128
 - naming 128
 - naming convention 18
 - restricting global 19
 - scope 128
 - sharing file scope variables across multiple files 131
 - verb phrases 48
 - verbs 48, 49
 - vertical tab 109
 - Vessel class 343
 - virtual destructor 420
 - virtual destructors 420
 - virtual function 267, 422
 - virtual functions 340, 362
 - virtual inheritance 360
 - virtual keyword 328
 - vocabulary
 - object-oriented 266
- W**
- wchar_t 105
 - Weapon class 343
 - well-behaved object
 - definition 446
 - What are pointers good for? 166
 - What is a function? 206
 - What is a pointer? 171
 - What is an object? 167
 - while statement 150
 - whole class 302
 - Windows™ 506
 - wireless local area network card 81
 - word 90
 - wrapping up a project 72
 - writing generic code 398