

Simple Data Types

Integral: char, short, int, long
 Floating-point: float, double

String Type

literals: examples: "1984" "I am a house."
 access to class: #include <string>

Constants

declaration: const data-type identifier = value;

Variable Declaration

declaration: data-type identifier[, identifier, ...];

Arithmetic Operators and Expressions

basic operators: + - * / %
 order of operations:
 highest: * / %
 lowest: + -
 parenthesis: () changes the order of operations

Type Conversion

simple: (result-type) value
 static: static_cast<result-type>value

Assignment

operator: =
 syntax: variable = expression

Block or Compound Statement

purpose: combines multiple statements into one statement; defines scope of identifiers
 syntax: {
 other statements
 }

String Manipulation

concatenation operator: +

Unary Increment Operator

++

Unary Decrement Operator

--

Output Stream Insertion Operator

<<

Standard Device Output

library access: #include <iostream>
 standard device var: cout
 some syntax of use: cout << expression[<< expression];

File Output

class: ofstream
 class access: #include <fstream>
 variable declaration: ofstream identifier;
 creating output stream: identifier.open(c-string-path);
 output to file: identifier << expression [<< expression];
 closing output stream: identifier.close();

Input Stream Extraction Operator

>>

Standard Device Input

library declaration: #include <iostream>
 standard device var: cin
 some syntax of use: cin >> variable[>> variable];

File Input

class: ifstream
 class access: #include <fstream>
 variable declaration: ifstream identifier;
 creating input stream: identifier.open(c-string-path);
 input from file: identifier >> expression [>> expression];
 closing input stream: identifier.close();

Function getline

syntax: getline(input-variable, string-variable);
 function: reads an entire line, including white-space
 example: string name;
 cout << "Enter your full name: ";
 getline(cin, name);

Formating Output

library access: #include <iomanip>
 maipulators: fixed - disables E notion
 showpoint - forces output of .
 functions: setprecision(fraction-size)
 setw(minimum-number-spaces)

Relational and Lgical Operators

== != < > <= >=
 & && | || !

If Selection

syntax: if (boolean-expression)
 statement
 if (boolean-expression)
 statement
 else
 statement
 if (boolean-expression)
 statement
 else [if (boolean-expression)
 statement]

Switch Selection

typical syntax: switch(expression)
 {
 case value1:
 statement1
 break;
 case value2:
 statement2
 break;
 ...
 case valueN:
 statementn
 break;
 default: statement
 }

Counters

purpose: to allow a variable to have its value changed based on its original value
 syntax: variable = variable operator expression;

While Loop

syntax: while (boolean-expression)
 statement

Do-While Loop

syntax: do
 statement
 while(boolean-expression);

For-Loop

syntax: for(initial-statement; loop-condition; update-statement)
 statement
 order of execution: initial-statement
 loop-condition
 statement
 update-statement
 return to loop condition

Break Statement

purpose: exits the surrounding control structure
 syntax: break;

Continue Statement

purpose: short circuits the execution of the statement of a loop; the loop continues
 syntax: continue;

Single Dimension Arrays

declaration syntax: type identifier[size];
 type identifier[] = { value-list};
 type identifier[size] = { value-list};
 element access syntax: identifier[index-expression]

Two Dimension Arrays

declaration syntax: type identifier[size][size];
 type identifier[][] = { value-lists};
 type identifier[size][size]
 = { value-lists};
 element access syntax: identifier[index-exp][index-exp]

Void Functions

prototype syntax: void function-identifier(parameter-list);
 function syntax: void function-identifier(parameter-list)
 {
 statements
 }
 function call syntax: function-identifier(actual-parameters);

Non-Void Functions

prototype syntax: type function-identifier(parameter-list);
 function syntax: type function-identifier(parameter-list)
 {
 statements
 at-least-one-return-statement
 }
 function call syntax: function-identifier(actual-parameters);

Return Statement

purpose: exit a function; may set a value to return from the function
 syntax: return;
 return expression;

Formal Parameter

parameter listed in a function header

Actual parameter

parameter listed in a function call

Pass-by-Value Parameter

a copy of the actual parameter is created under the name of the formal parameter; the formal parameter becomes a separate variable and changes to it cannot change the actual parameter

formal parameter syntax: nothing additional
 actual parameter syntax: nothing additional

Pass-by-Reference Parameter

the address of the actual parameter is passed to the formal parameter; the formal parameter becomes a local name for the memory address of the actual parameter; changes to the formal parameter are changes to the actual parameter

special formal parameter syntax: type & identifier
 special actual parameter syntax: nothing additional

Note: Arrays are pass by reference.

Const Functions

purpose: function does not change passed parameters
 syntax: function header ends in *const*

Const Parameters

purpose: function does not change passed parameter
 syntax: *const* precedes parameter type in header

Const Return Type

purpose: prevents address of returned type from being used to access the data structure in the function or class
 syntax: *const* precedes return type of function header

Parameter List

purpose: list of variables to be sent to a function in a call or to receive sent values or addresses in a function header
 actual parameter syntax: expression[, expression]
 formal parameter syntax: type [&] identifier[, type [&] identifier]

Array as actual Parameter

actual parameter: array-identifier

Array as Formal Parameter

when an array is a formal parameter, the left-most index may remain blank; arrays are always pass-by-reference, the ampersand (&) is not required

examples: int a[]
 double f[][MAX_COL]

Access Modifiers

public, protected, private

Struct

a collection of members, default access is public

syntax: struct struct-name
 {
 type identifier;
 type identifier;

 type identifier;
 };
 declaration syntax: struct-name identifier;
 member access: identifier.member

Classes

a collection of members, default access is private; members consist of member function methods and data member; member functions are implemented separately from the definition to facilitate information hiding;

definition syntax: class class-name
 {
 public:

 public-members
 protected:
 protected-members
 private:
 private-members
 };

function method implementation syntax:
 type class-name::method-name(parameter-list)
 {
 statements
 }

Constructors

members with the same name as the collection; called when an object of the class type is created; constructors are always public

definition syntax: class-name(parameter-list);
 implementation syntax:
 class-name::class-name(parameter-list)
 {
 statements
 }

call syntax: class-name variable-name;
 class-name variable-name(parameter-list);
 variable-name = class-name(parameter-list);

Destructors

members with the same name as the collection preceded by a tilde (~); called when an object of the class type is deleted or passes out of scope; destructors are always public

definition syntax: ~class-name();
 implementation syntax:
 class-name::~~class-name()
 {
 statements
 }

Inheritance

a collection may inherit the public and protected members of another class; the inherited members become members of the inheriting class; members inherited from the parent class may be redefined (overridden) in the child class

class inheritance syntax:
 class child-class-name : access-modifier parent-class-name
 {
 ...
 };

constructor header syntax:
 child-class-name(p-list) : parent-class-name(p-list)

constructor usage syntax:
 child-class-name identifier(p-list);

Typical Creation of Random Numbers

```
#include <cstdlib>
#include <ctime>

srand((unsigned)time(NULL)); // seed generator

long num = rand(); // create random integer
```

Pointers

a variables whose content is an address

declaration syntax: type * identifier;

NULL

pointer value that does not point to any memory address; any pointer can be assigned the constant NULL; any pointer can be tested against the constant NULL

Address of Operator

returns the address of a its operand

syntax: & operand

Dereferencing Operator

refers to the object indicated by the address in a pointer

syntax: * pointer-variable

Member Access Operator Arrow

refers to the member of an object indicated by the address in a pointer

syntax: collection-variable -> member

Dynamic Variables

dynamic variables are created a run time with the operator new; the memory used to create a dynamic variable must be returned when no longer needed with the operator delete; dynamic variables have no identifiers so they have no scope; only pointers may reference dynamic variables

operator new syntax: new type
 new type(parameter-list);
 new type[size]
 new type[size][size] ...;

operator delete syntax: delete pointer;
 delete [] pointer-to-array;

Relational Operator Overloading for Classes

declare public; return boolean value

header syntax: bool operatorop(const type & a) const

Assignment Operator Overloading for Classes

*declare public; return *this*

header syntax: const type & operator=(const type & a)

Output Stream Operator Overloading for Classes

declare friend in prototype only; no access modifier; return ostream identifier

prototype syntax:

friend ostream & operator<<(ostream & out, type & a);

header syntax:

ostream & operator<<(ostream & out, type & a) { ... }

Input Stream Operator Overloading for Classes

declare friend in prototype only; no access modifier; return istream identifier

prototype syntax:

friend istream & operator>>(istream & in, type & a);

header syntax:

istream & operator>>(istream & in, type & a) { ... }

Function Templates

syntax: template<class T>
 function definition

call syntax: function-name(parameter-list)
 function-name<type>(parameter-list)

Class Templates

syntax: template<class T>
 class definition

object declaration syntax:

class-name<type> identifier;
class-name<type> identifier(parameter-list);

Setting Namespace to std

using namespace std;

Typical Pre-compiler Instructions

```
#include <library>
#ifndef identifier
#define identifier
#endif
```

Common Libraries

iostream	- standard input and output; contains objects cin and cout
fstream	- file input and output; contains classes ifstream and ofstream
cstdlib	- standard library; contains srand, rand functions and many other functions
string	- class string
iomanip	- output manipulation functions setw and setprecision
ctime	- time functions; contains function time
cmath	- math functions