General:

Begins with function main()

Has two types of comments, // and /\*...\*/

Variables of type int are not always the same size (2 bytes or 4 bytes)

\ is an escape character

Character is enclosed in single quotes

String is enclosed in double quotes

Character and ASCII are nearly the same to the compiler

Variables don’t have to be declared just after function declaration (C only)

C is a subset of C++

C++ variable names must start with an underscore or a letter

Statements evaluate to true for any non-zero value

Some storage classes: auto (default), static, external, register

A void function should have a name that conveys action

One should write functions that are logically coherent

C++ has no bound checking with array subscripts

Arrays and strings are constant pointers that always point to subscript 0

Arrays only hold one type of data

The name of the file in #include [SOMETHING] is not case sensitive by itself

/ performs div (how many times b goes into a, given a/b) when used on two ints

/ performs actual division when a or b is not an int, given a/b

Attempting to store a float into an int truncates the decimal

In a non-void function, you must return something

When declaring an array of size 20, array subscripts range from 0 to 19

Can initialize an array to 0 using: int Array[20]={0};

& is the reference operator used to get the memory address of a variable

\* is the dereference operator used to get the value of what a pointer variable is pointing at; can also be used to declare a pointer

3 types of parameters: Pass by value (C/C++), pass by reference (C++), pass by value with pointer (C/C++)

Pointer math: P + (bytesize of datatype)

C++ allows up to 12 dimensional arrays int Array[A][B][…][L]

When using enum {X, Y, Z};, if no values specified, X = 0 and values are assigned in order. If X = 2, then Y = 3 and Z = 4

cin.good() returns a 1 or non-zero if the last input was successful

Pointer variables can only hold the address to another variable of the same type

The routines in conio.h are **NOT** redirectionable

The new operator is used to allocate memory during program execution

Use typedef to redefine a data type:

typedef DataTypeName NewDataTypeName

typedef int integer;

typedef int\* IntPtr;

Data Files:

When working with data files, must #include <fstream>

ofstream is used to create output files

ifstream is used to read input files

Using data files in 6 easy steps:

1) #include <fstream>

2) declare file variable (ifstream FileIn; ofstream FileOut;)

3) open file for input or output

FileIn.Open(“input.dat”);

FileOut.Open(“output.dat”);

If a file does not exist for output, it will be created, if it already exists, it will be overwritten

4) Check for opening errors

if (FileIn.fail()){cerr<<”File Error”;return;}

5) Use the data file

while (!FileIn.eof()){ ... }

6) Close the file

FileIn.close();

FileOut.close();

Modules:

1) Specification file (\*.h): Abstract, function prototypes and constants, function descriptions, public, considered the abstract that the client can see

2) Implementation file (\*.cpp): Detail of the function, source code -> object code

A global variable declared as static in one file cannot be accessed from another file

Command Line Arguments:

int main(int argc, char \*argv[]){ ... }

argc: number of command line arguments

argv: array of char pointers where each one points to a different argument

argv[0]: full path and name of executable

argv[1]: argument 1

argv[2]: argument 2

...

argv[n]: argument n

Structs and Classes:

Structs are public by default and are C/C++

Classes are private by default and are C++ only

In Classes and Structs, data members are accessed by ObjectName.MemberName

In C++, structs and classes can be passed both by reference and value

The built-in assignment operator performs a shallow copy

Shallow copy + pointers = bad things

A friend function has access to the private data members

The keyword friend is only needed in the function prototype

Definitions:

Member function: Part of a class definition used to manipulate class data

Constructor: Used to initialize data members with passed in data or different data type then class

Object: A variable of user defined data type

public: Data members and functions that anyone has access to

Copy Constructor: Used to initialize data members with object of same class definition

Friend Function: Not a member of the class, but yet has access to private members

private: Members that are only accessible through member functions or friend functions

Default Constructor: Used to initialize data members with a preset value

class: User defined data type that has both data and functions associated with it

Sample Class Declaration:

class MyClass{

private:

int DataMember1;

char Datamember2;

public:

MyClass();

const MyClass& operator=(const MyClass &Source);

friend ostream& operator<<(ostream &MCout, const MyClass &Source);

}; <- **Don’t forget the trailing semi-colon!**

Operator Overloads:

Assignment Operators:

const ClassName& operator[TYPE](const ClassName &Source);

Comparator Operators:

const bool operator[TYPE](const ClassName &RightHandSide) const;

Arithmetic Operators:

const ClassName operator[TYPE](const ClassName &RightHandSide) const;

IOStream Operators:

friend ostream& operator<<(ostream &ClassNameOut, const ClassName &Source);

friend istream& operator>>(istream &ClassNameIn, ClassName &Target);

Linked Lists:

List -> Dummy Head Node [ X | ] -> [ ‘a’ | ] -> [ ‘b’ | ] -> [ ‘c’ | ] -> |||

Easiest to use -> to navigate the links

To create linked list:

1) Create new node

2) Fill new node with data

3) Insert node in desired location in the list

Sample Linked List Node:

struct Node {

char Ch;

Node \*Link; };

Things to remember:

1) Always check for enough memory

if (List == NULL)

2) Draw pictures

List [ ] -> [ | ] -> |||

3) Doesn’t matter where arrow points in the picture as long as it’s accurate

4) Always remember to advance to the next link

Last = Last->Link;

5) Never leave dangling pointers!

Bubble Sort Linked List Code:

void SortList(NodePtr L){

NodePtr Current, Next;

int NotSorted = 1, SwappedChar;

if (L->Link == NULL || L->Link->Link == NULL)

return;

while (NotSorted){

SwappedChar = 0;

Current = Next = L->Link;

while (Next->Link != NULL){

Next = Next->Link;

if (Current->Ch = Next->Ch){

char Temp = Current->Ch;

Current->Ch = Next->Ch;

Next->Ch = Temp;

SwappedChar = 1; }

Current = Current->Link; }

NotSorted = SwappedChar ? 1 : 0; }

}

Dynamic Data:

The new operator is only a request, not a guarantee

Given int \*DynData; DynData = new int[...],

Check using if (DynData == NULL) return;

Reallocate memory when done:

Use delete DynData; if DynData is **NOT** an array

Use delete[] DynData; if DynData **IS** an array

Default Arguments:

void Display (int x = 100, int y = 10){

if (x > y)

cout << x;

else

cout << y; }

void main(){

int M = 10, N = 50;

Display(M,N); //50

Display(40, 10); //40

Display(15); //15

Display(); } //100

Templates:

Very useful when you have the same exact code doing the same thing for multiple variable types

Used to define a sort of generic variable for the data type

template <class AnyType>

void swap(AnyType &x, AnyType &y){

AnyType temp = x;

X = y;

Y = temp; }

Can have multiple variable data types

template <class Type1, class Type2>

Also works for classes:

template <class Type>

class MyClass{

private:

Type DataMember1;

public:

... };

Derived Classes and Inheritance:

Classes can have multiple parents as well as multiple levels of parents

Class A <- Parent Class

Class B <- Derived (Child) Class

Class C <- Derived (Child) Class

Class D <- Parent Class Number 1

Class F <- Derived (Child) Class

Class E <- Parent Class Number 2

Class F <- Derived (Child) Class

Class G <- Parent Class Level 1

Class H <- Parent Class Level 2

Class I <- Derived (Child) Class

Syntax:

class DerivedClassName:classaccess BaseClassName{

//class definition

};

classaccess can be public, private, or protected

Example:

class Employee{

private:

char Name[2];

float Salary;

public:

Employee();

void ReadName();

void DisplayName();

float Salary();

};

class HourlyEmployee:public Employee{

private:

float Wage;

public:

HourlyEmployee();

float Usage();

void ChangeWage();

};

Searching Arrays:

Two kinds of searches

1) Linear Search

Works on ordered or non-ordered arrays

Starts at beginning and searches until end

Can be slow (Average # of comparisons on a non-ordered list to find item is ArraySize/2 if the item is there, ArraySize if not there; Average # of comparisons on an ordered list regardless of whether the item is there is ArraySize/2)

2) Binary Search

ONLY works on ordered arrays

Scales well with large data sets

Cuts the list in half every time

Will take log2(ArraySize)-1 number of comparisons (E.g., for an array of 1000 elements, it will take 10 comparisons at most)

General Bubble Sort Algorithm:

BubbleSort(int List[], int ArraySize)

declare bool SwapsMade and set to TRUE

declare int NumberUnsorted and set to ArraySize - 1

while (SwapsMade AND NumberUnsorted >= 1)

predecrement NumberUnsorted

set SwapsMade to FALSE

for (set i to 0, while i > NumberUnsorted, ++ i)

if (List[i] greaterthan List[i + 1])

swap(List[i], List[i + 1])

set SwapsMade to TRUE

end if

end for

end while

Extra Credit:

Number of brothers and sisters: 7

One of previous years’ Halloween costumes: Zorro

Days it takes Mercury to orbit the sun: 88

How many feet in a mile: 5280

Favorite beer: Guinness