**POINTERS**

Basic:

int k = 14;

int \*ptrk; *//points to garbage value*

ptrk = nullptr; *//points to null*

ptrk = &k; *//points to var k, stores memory location of k*

//&(var) = address of var

\*ptrk = 15; *// \* walks the arrow to the var ptr is pointing at, “dereference”*

cout<<k<<endl; *//outs 15*

cout<<ptrk<<endl; *//outs LOC of k*

cout<<\*ptrk<<endl; *//outs 15*

cout<<&ptrk<<endl; *//outs LOC of ptrk*

int\* p1, \*p2; *//need \* on both!!*

*//many pointers can point to one var*

\*p1 = 15; *//ILLEGAL: segmentation fault! because p1 is not pointing to anything rn*

Pointers & Array:

int main(){

int a[5];

cout<<sizeof(a)/sizeof(int);

*//prints out size of a*

}

*//does not work with functions, as arrays are passed as a pointer*

void printSize(int arr[])

*//same as* printSize(int \* arr)

{

*//code*

}

Pointers & Classes:

bankAccount b;

bankAccount \* ptrBankAccount = &b;

b.setName(“Pixie”);

ptrBankAccount->setName(“Muffin”);

*// -> syntax for objects, equivalent to*

(\*ptrBankAccount).setName(“Muffin”);

*//need parenthesis!*

**SAMPLE CAR CLASS(Car.cpp file)**

#include <iostream>

#include “Car.h”

using namespace std;

Car::Car() //constructor

{

m\_make = “”;

m\_model = “”;

}

//Alternate syntax: initialization list

Car::Car(): m\_make(“”),m\_model(“”){}

Car::Car(string make, string model)

{

m\_make = make;

m\_model = model;

}

//Alternate syntax: initialization list

Car::Car(string make, string model): m\_make(make), m\_model(model){}

void Car::start()

{

cout<<”start”<<endl;

}

string Car::getMake()

{

return m\_make;

}

void Car::setMake(string make)

{

m\_make = make;

}

**C-STRINGS:**

char s[10]; *//empty c-string with 10 slots*

s = “supp”; *//ILLEGAL ASSIGNMENT*

char t[5] = “Hi”;

//equivalent to s[0] = ‘H’; s[1] = ‘i’; s[2] = ‘\0’;

char u[] = “Howie”; *//automatically create char array with 6 slots including ‘\0’ at end*

char w[] = {‘a’, ‘b’, ‘c’}; *//not a cstring, no null byte*

Walk through Cstring with loop:

int index = 0;

while(array[index]!='\0'){

array[index] = 'x';

index++;

}

Library Functions:

strcpy(dest, src)

//void copy(cstring, cstring), need to make sure dest have enough space for src

strcat(dest, src)

//void add(cstring, cstring), need to have enough space for result and null

strlen(src)

//int length(cstring), doesn’t include ‘\0’

toupper(c), tolower(c) //returns int!

char c = toupper(‘a’);

//easily convert back to char

cout<< char(toupper(‘a’)); //prints A

strcmp(char s[], char t[])

//returns **0** if equal\*\*, <0 is s is less than t, >0 if s is greater than t

Inputs:

Can use cin >> , but reads only one word

cin.getline( char \*, int max)

//supports cstring data, need to make sure char\* is big enough for input, cannot exceed max

\*\*be extra careful when you combine cin and getline!! faces regular string problems

**SAMPLE CAR CLASS(main.cpp)**

#include <iostream>

#include “Car.h”

int main()

{

Car c; //calls basic constructor

Car c(); //ILLEGAL!!

c = Car(); //legal

Car myCar(“VW”, “Golf”);

//calls different constructor

c.make =“VW”; //ILLEGAL, private var

myCar.setMake(“Honda”);

myCar.setModel(“Prelude”);

cout<< myCar.getMake()<< endl;

}

**STRUCTS:**

struct Date

{

int month;

int day;

int year;

}D1, D2; //SEMI COLON!

//D1, D2 are declared as type Date

//\*\*member variables can have same name as struct!!

Date dueDate = {12, 31, 2003};

cout << dueDate.month; //outputs 12

Date birthday = {12};

//day and year will be initialized to 0

**DYANMIC VARIABLES**

Basics:

int\* p1 = new int;

*//reserves a space in heap for new int*

\*p1 = 10; *//sets a value for the space* *reserved, aliases are no longer needed*

delete p1; *//dynamic var that is created needs to be deleted*

p1 = nullptr; *//the val in memory is deleted, but p1 still exists*

with Classes:

bankAccount\* bPtr = new bankAccount(“howie”, 10.0);

bankAccount\* aPtr = new bankAccount();

delete bPtr; delete aPtr;

Dynamic Arrays:

int k;

cin >> k;

int\* ptrArray = new int[k]; *//LEGAL!*

*//reserves more space than new int;*

for(int j = 0; j<k; j++){

ptrArray[j] = j;

*//alternate syntax*

\*(ptrArray + j) = j; *//equiv to [] syntax*

*//alternate…*

int \* temp = ptrArray;

\*(temp + j) = j;

ptrArray = temp; *//ILLEGAL*

}

delete [] ptrArray; *//delete with []*

**SAMPLE Car CLASS(Car.h file)**

#ifndef CAR\_H

//if not defined then, continue till #endif, prevent .h file from being created multiple times

#define CAR\_H

//C++ will set CAR\_H to value 1

#include <string>

using namespace std;

class Car{

public:

Car(); //constructor

Car(string, string);

//can overload constructors

void start(); //methods

void stop();

string getMake();

//accessors: allow public to access private data

void setMake(string make); //mutators: allow public to change private data

private:

string m\_make;

string m\_model;

}; //SEMICOLON

#endif

**ENUMS**

enum day = {Sunday, Monday};

//equivalent to const int Sunday = 0;

const int Monday = 1;

enum SUIT{SPADES = 100, CLUBS = 100, HEARTS = 200, DIAMONDS = 200, GREY};

//legal to have overlapping values, grey will be 201 by default

Enums in Classes:

class Ticket{

public:

enum KIND {ATHLETIC\_GAME, CONCERT, MOVIE, OTHER};

};

int main(){

Ticket::KIND k = Ticket::KIND::MOVIE;

if(k == Ticket::KIND::MOVIE)

cout << “It’s a movie!” <<endl;

Ticket::KIND m = TICKET::KIND::MOVIE+1;

//ILLEGAL, enum don’t support + - \* /

}

**Reverse Cstring!**

**void** csReverse(**char** c[])

{

**int** len = 0;

**while**(c[len]!='\0') len++;

**for**(**int** k=0; k<len/2; k++){

**char** tmp = c[k];

c[k] = c[len-1-k];

c[len-1-k] = tmp;

}

}

**Sample Class with Dynamic Arrays**

Class Airplane{

Public:

Airplane(int size = 100);

~Airplane(); //Destructor! called when dynamic variable falls out of scope

Private:

Passenger \* myArrayOfPassenger; //dynamic array, no set size

int amount;

}

Airplane::Airplane(int size): amount(size)

{

myArrayOfPassenger = new Passenger[amount];

//calls default constructor of passenger, all the passengers are called from same constructor

}

Airplane::~Airplane()

{

delete []myArrayOfPassengers;

}

int main()

{

cout<<”plane size?”<<endl;

int size = 0;

cin >> size;

Airplane \* plane = new Airplane(size); //need \* !!

delete(plane); //destructor is called

}

}

}

**SAMPLE THROWING ERROR**

#include <iostream>

#include <string>

#include <stdexcept>

using namespace std;

void throws\_error()

{

throw logic\_error("discussion section error");

}

int main()

{

try {

throws\_error();

}catch(logic\_error e){

cout << e.what() << endl;

}

}

**THINGS TO BE CAREFUL ABOUT:**

-Default constructors leave primitive member variables uninitialized and calls default constructor for class members(objects)

-If member variables of an object doesn’t have a default constructor, it must be initialized through the initializer list!! when we instantiate an object, member variables gets initialized first, then constructor is called. Member variables are initialized by first consulting the initializer list

-Default, parameterless constructor is not supported by std::logic\_error

-When you pass in an object as const into a function, that object should only call functions marked as const

**Cstring: Remove non-alpha**

**void** removeNonAlpha(**char** m[]){

**for**(**int** k=0; m[k]!='\0';k++){

**if**(!isalpha(m[k])){

**for**(**int** i = k; m[i]!='\0'; i++){

m[i] = m[i+1];

}

k--;

}

}

}

**SHIFT RIGHT FUNCTION PROJECT 4**

int shiftRight( std::string array[ ], int size, int amount, std::string placeholder )

{

int result( 0 );

if (size < 0 || amount < 0)

{

result = -1;

}

else

{

// loop as many times as requested

for( int i = 1; i <= amount; i++ )

{

// push the data element down one

for (int j = size - 1; j > 0; j--)

{

array[ j ] = array[ j-1 ];

}

// fill in the placeholder value

array[ 0 ] = placeholder;

result = result + 1;

}

}

return( result );

}

**Using THIS in constructors:**

class Person{

Public:

Person(){}

Person(int uid, double gpa){

this -> uid = uid;

*//this is a pointer to current object, eq. to:*

(\*this).uid = uid;

this -> gpa = gpa;

}

int uid;

double gpa;

}

**EQUIVALENT to DOT guy for OBJECTs:**

int main(){

Person \*persons = new Person[10];

for(int k = 0; k < 10; k++){

cout<<persons[k].uid; *//DOT GUY cuz persons[k] is already dereferenced*

cout<<\*(persons+k).uid;

cout<<(persons+k) -> uid;

}

**Special Case when printing CString**

char\* c = new char[4];

c[0]= ‘H’; c[1]= ‘I’; c[2]=’!’; c[3] = ‘\0’;

cout << c <<endl; //does not print loc of c, but HI!

//because << is overloaded for c strings

cout<< c+1 <<endl; //outs I!

cout<< (void\*) c; //casts it back to print loc of c

**Passing Pointers as Parameters:**

void foo(const int a[]); //a is a pointer to a const int

void foo(int const \* b); //b is a pointer to a const int

void foo(int \* const e); //e is a const pointer to an int

void foo(const int \* const f); //f is a const pointer to a const int