CS24 - Problem Solving with Computers II

Data types, classes, pointers and memory (Review?)

The very basics...

Control structures:

if/else if/ else: if(condition) { DoStuff(); } else { DoOtherStuff(); }

```
switch(variable){    case (1): cout << "Variable is 1" << endl;  break;
                     case ("Bob"): cout << "Variable is Bob" << endl;  break;}
bool greaterThan = (a > b) ? true : false;
```

While loop: while(condition) { DoStuff(); }

For loop: for (int i=0; i<5; i++){ cout << "Loop number: "+i << endl; }

```
string[] names = ["Bob", "Nancy", "Drew"];  // could loop through each i'th position of the array
for(string name: names){  cout << name << endl; } // easier to loop through each name
```

The very basics...

Data Types:

char = character (‘a’, ‘b’, ‘+’, ‘3’, etc)

string = a string of characters (“Bob”, “Nancy”, etc)

bool = boolean value (true/false, 1/0)

int = integer values (-3, -1, 0, 2, 4, etc)

float = floating point decimal, up to seven digits of precision, 32bit (1.234567×10^4)

double = double precision number, much higher range (34595858.235325)

Generic data type (template)

“template” can be used if a function is built for multiple types of data

```
template <class ClassToCompare>
ClassToCompare getGreater(ClassToCompare first, ClassToCompare second){
    return (first > second) ? first : second;
}

int main(){
    int zero=0, one=1;
    char a='a', b='b';
    cout << getGreater(zero, one) << endl; // prints "1"
    cout << getGreater(a, b) << endl; // prints "b"
}
```

Classes (and Structs)

Classes are the backbone of C++'s "Object Oriented Programming" designation

Function to bundle together variables/functions that work together (encapsulation)

Classes are composed of "inner variables" known as "fields" and class specific functions known as "methods"

Fields and methods can be private or public (the default for class is private, struct is public)

Declaring a class is creating its template. Defining a class is describing how to fill the template

When you define a specific object, you create an instance of that class

```cpp
using namespace std;

class Vehicle{
private:
    int numberWheels=0;  //valid but should only be used if instances all start with this value
    int numberDoors=0;

public:
    Vehicle(int, int); // Only declares that this option is available
    int getWheels() { return numberWheels;} // Declares and defines
    int getDoors() { return numberDoors;}
};

Vehicle::Vehicle(int wheels, int doors){ numberWheels=wheels; numberDoors=doors;}

int main(){
    Vehicle motorcycle( wheels: 2, doors: 0);  // Creates an instance of the object
    cout << motorcycle.getWheels() << endl; // Calls a public function to get number of wheels
}
```

```cpp
#include <iostream>
using namespace std;

struct Vehicle{
    int numberWheels;
    int numberDoors;
    Vehicle(int wheels=0, int doors=0); // This is the proper way to set defaults outside of a constructor
};

Vehicle::Vehicle(int wheels, int doors){ numberWheels=wheels; numberDoors=doors;}

int main(){
    Vehicle motorcycle;
    motorcycle.numberWheels = 2;
    cout << motorcycle.numberWheels << endl;
}
```

Classes (and Structs)

Private variables cannot be accessed from outside the scope of the class

Modifying/accessing private variables is performed with “get” and “set” operations

Important when controlling who has access to variables and what they can be set to

Prevents setting variables to something that could break the program (ie. infinite loops or divide by 0)

Classes (and Structs)

Important terminology:

- Parameterized constructor (with default values)
- Default constructor
- Initializer lists
- Destructor

```cpp
#include <iostream>
using namespace std;

class Vehicle{
private:
    int numberWheels;
    int numberDoors;

public:
    Vehicle(){numberWheels=0; numberDoors=0;}; // Default Constructor
    Vehicle(int wheels, int doors): numberWheels(wheels), numberDoors(doors){} // Initializer List
    int getWheels() { return numberWheels;} // Declares and defines
    int getDoors() { return numberDoors;}
};

int main(){
    Vehicle defaultVehicle;
    Vehicle motorcycle( wheels: 2, doors: 0);  // Creates an instance of the object
    cout << motorcycle.getWheels() << endl; // Calls a public function to get number of wheels
    motorcycle.~Vehicle(); // Destructor, not normally explicitly called
}
```

Memory

In Java/Python, memory is managed for you

In C++ only some of the memory is managed

Memory for a program consists of four types of memory

- “Code memory” - stores the binary files for the executable file
- “Static memory” - lasts for the duration of the program (global variables)
- “Stack memory” - lasts for the duration of the “call stack” (duration of a function)
- “Heap memory” - dynamic memory, persists after function close -- needs allocated manually (leaks)

Memory

Each memory location has an address

In C++, you can “point” to a location in memory

By default, this address will print out in hexadecimal

Location zero is reserved for a “nullptr”, a pointer to nothing

A pointer (to memory location) is defined with an *

A reference to a certain location in memory is defined with an & (more on the difference later)

Memory

```
#define arraySize 10
int main(){
    int *arr = new int[arraySize];

    for(int i=0; i< arraySize; i++){
        cout << &arr[i] << endl; //prints in Hexadecimal
        //printf("%ju\n", (uintmax_t)&arr[i]); //converts to decimal, don't worry about this syntax
    }
}
```

Memory

```
0xf71770
0xf71774
0xf71778
0xf7177c
0xf71780
0xf71784
0xf71788
0xf7178c
0xf71790
0xf71794
```

```cpp
#define arraySize 10
int main(){
    int *arr = new int[arraySize];

    for(int i=0; i< arraySize; i++){
        cout << &arr[i] << endl; //prints in Hexadecimal
        //printf("%ju\n", (uintmax_t)&arr[i]); //converts to decimal, don't worry about this syntax
    }
}
```

Memory

```
#define arraySize 10
int main(){
    int *arr = (int*) malloc( _Size: arraySize * sizeof(int)); //allocates memory with malloc

    for(int i=0; i< arraySize; i++){
        //cout << &arr[i] << endl; //prints in Hexadecimal
        printf( format: "%ju\n", (uintmax_t)&arr[i]); //converts to decimal, don't worry about this syntax
    }
}
```

Memory

```
595824
595828
595832
595836
595840
595844
595848
595852
595856
595860
```

```
#define arraySize 10
int main(){
    int *arr = (int*) malloc( _Size: arraySize * sizeof(int)); //allocates memory with malloc

    for(int i=0; i< arraySize; i++){
        //cout << &arr[i] << endl; //prints in Hexadecimal
        printf( format: "%ju\n", (uintmax_t)&arr[i]); //converts to decimal, don't worry about this syntax
    }
}
```

Memory

Each int takes 4 bytes (32 bits)

When memory is allocated, a block of memory is found that can hold the entire array

What happens if you declare a new string?

Memory

Each int takes 4 bytes (32 bits)

When memory is allocated, a block of memory is found that can hold the entire array

What happens if you declare a new string?

Try it and post to Piazza!

Pointers and References

Pointers vs References (what's the difference?)

- Internally a reference is a pointer
- A reference should be used when passing in an object to be manipulated by a function or when returning an object
- This represents passing the object itself into or out of a function
- Use pointers for everything else

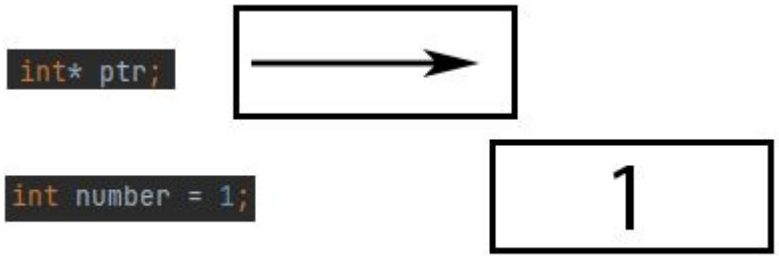
```
class Vehicle{
private:
    int numberWheels;
    int numberDoors;

public:
    Vehicle(){numberWheels=0; numberDoors=0;}; // Default Constructor
    Vehicle(int wheels, int doors): numberWheels(wheels), numberDoors(doors){} // Initializer List
    int getWheels() { return numberWheels;} // Declares and defines
    int getDoors() { return numberDoors;}
    void setDoors(int newDoors){numberDoors = newDoors;}
    void copyNumberOfDoors(Vehicle &copyToThisVehicle){copyToThisVehicle.setDoors(numberDoors);};
};


int main(){
    Vehicle copyFrom( wheels: 2, doors: 4);
    Vehicle copyTo( wheels: 3, doors: 6);
    cout << copyTo.getDoors() << endl; // Prints 6
    copyFrom.copyNumberOfDoors( &: copyTo); // Changes referenced Vehicle's number of doors to 4
    cout << copyTo.getDoors() << endl; // Prints 4
}
```

Pointers and References

How to use pointers:

```
int main(){
    int* ptr;  // Creates a location in memory that will store the address of something else
    int number = 1; // Creates a location in memory that stores a "1"
    ptr = &number;  // Sets "ptr" to point to the location of memory that stores "number"
    *ptr = *ptr + 1; // "Dereferences", allows you to work with the value stored at the location
    cout << ptr << endl; // Prints the location being pointed at
    cout << *ptr << endl; // Prints the value at the location (Will print a 2);
}
```

Pointers and References

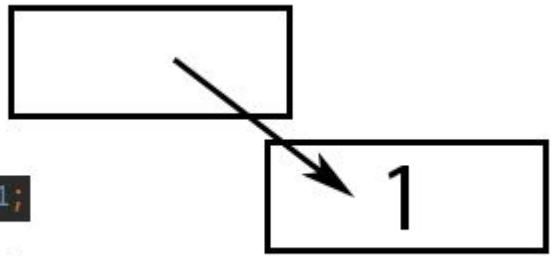


Pointers and References

```
int* ptr;
```

```
int number = 1;
```

```
ptr = &number;
```



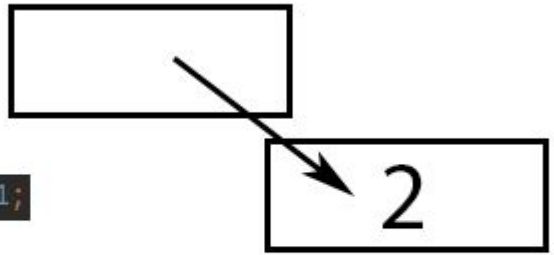
Pointers and References

```
int* ptr;
```

```
int number = 1;
```

```
ptr = &number;
```

```
*ptr = *ptr + 1;
```



What you should know for Quiz 1 (Friday July 2nd)

How to define/declare classes and the difference between public and private

What scope is and how to access members of a class from outside the declaration scope

How a class is initialized with the different constructors

What pointers and references are

The different types of memory and how it’s allocated

Up next...

Linked Lists!