PHY 250

Andrew Diggs

Winter 2024

C++: Variables

All variables must have their type declared.

```
//Hello World
int my_int;
float fp;
void LegacyOfThe;
...
```

C++: Passing Variables

```
void DoSomething(float var){
printf("var = %.2f");
}
```

```
int main(){
float fp = 10.0 ;
DoSomething(fp);
}
```

```
var = 10.0
```

C++: Passing by Copy

```
void DoSomething(float var){
var = 2.0;
printf("var = %.1f",var);
}
```

```
int main(){
float fp = 10.0 ;
DoSomething(fp);
printf("fp = %.1f",fp);
return 0 ;
}
```

```
    var = 2.0
fp = 10.0
```

C++: Passing by Pointer

```
void DoSomething(float * var){
*var = 2.0;
printf("*var = %.1f",*var);
}
```

The * means we are accessing the value stored var.

```
int main(){
float fp = 10.0 ;
DoSomething(&fp);
printf("fp = %.1f",fp);
return 0 ;
}
```

The & means we are passing the address of fp.

C++: Passing by Pointer

```
        *var = 2.0
fp = 2.0
```

C++: Passing by Reference

```
void DoSomething(float &var){
var = 2.0;
printf("var = %.1f\n",var);
}
```

The & also means we are accessing the value stored at the address of var.

```
int main(){
float fp = 10.0 ;
DoSomething(fp);
printf("fp = %.1f",fp);
return 0 ;
}
```

C++: Passing by Reference

```
     var = 2.0
fp = 2.0
```

This is just like fortran!

C++: Copies, Pointers, and Reference

```
int  main(){
float fp = 3.0 ;
float & ref = fp;
printf("ref = %.1f\n");
float * ptr = &fp;
*ptr = 4.0;
printf("ref = %.1f *ptr = %.1f\n", ref, *ptr);
return 0 ;
}
```

```
    var = 3.0
var = 4.0 *ptr = 4.0
```

C++: Memory

Two types of memory

- static: allocated on the stack. Faster to access but limited in size. Memory is managed by subsystem.
- dynamic: allocated on the heap. Slower to access but but large amounts available. Memory is managed by **you**.

```
int  main(){
float fp = 10.0 ;
float arr[10 ];
return 0 ;
}
```

fp and arr are being allocated on the stack. Memory allocated on the stack is released upon exiting the scope, { }.

C++: Dynamic Memory

```
int  main(){
float * ptr = (float *)malloc(10 * sizeof(float));
return 0 ;
}
```

malloc accesses the heap and "gets" a memory block the size you request. malloc returns a void * so the (float *) typecasts. But this code is missing something?

C++: Dynamic Memory

Dynamic Memory Needs to be released by you!!!

```
int  main(){
float * ptr = (float *)malloc(10 * sizeof(float));
free(ptr);
return 0 ;
}
```

C++: Dynamic Memory

Modern C++ has an additional and safer way to allocate dynamic memory.

```
int  main(){
float * ptr = new float [10 ];
delete[] ptr;
return 0 ;
}
```

Any memory allocated using new **MUST** be released by you. This is done using the delete keyword.

C++: Dynamic Memory 2D

```
int  main(){
float ** ptr = new float *[10 ];
for(int i = 0; i < 10; i++) {
ptr[i] = new float [10 ];
}
for(int i = 0; i < 10; i++) {
delete[] ptr[i];
}
delete[] ptr;
return 0 ;
}
```

For a double pointer you need to delete[] each inner float * and then delete[] the float **.

C++: Main Point

```
float  DoSomething(float var){
float ret;
float arr[1000];
...
return ret;
}
```

This is good, arr is fast memory and will be released when the function returns.

C++: Main Point

```
float  DoSomething(float var){
float ret;
float * arr = new  float [1000];
...
return ret;
}
```

This is bad!!!

arr will not be released and every time you call DoSomething a new block of memory will be reserved and not released. This is know as a memory leak.

C++: Main Point

Sometimes you need a very large amount of memory inside a function

```
float  DoSomething(float var){
float ret;
float * arr = new  float [1000000];
...
delete [] arr;
return ret;
}
```

Great, No more leaks.

C++: Lab 1

Write a function that takes two int s and returns a float ** allocated using new . Assign the values of prt[i][j] = i*j inside the function.

Write a void function that takes a float ** and two int s with a nested for loop that prints the value of i*j and the value of your float **[i][j] and compare the answers.

You will need to release your float **. Write a third function that returns a void and deletes the memory you allocated in your first function.

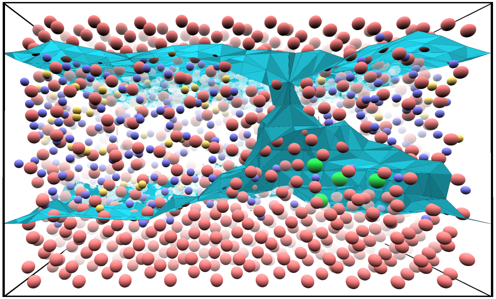
We have diverse backgrounds in this class, If you need help I encourage you to ask your neighbor. If you finish early please see if anyone needs help.

Challenge: Complete the tasks of Lab 1 using only pointer arithmetic. Your print function should be a template that takes a void *.

C++: Objects

Objects are a very powerful tool in C++ that allows us to organize our code.

C++: Objects



C++: Objects

Two main types of objects in C++, struct and class.
They are identical. The reason there are two is for further organization.
We will just focus on class for now.

```
class MyClass {
private:

public:

};
```

The keywords private, protected, and public are access qualifiers.
We won't be using protected.

C++: Objects

A great way to think about objects is a car. Let's make an engine class.
We need to start our engine.

```cpp
class Engine {
private:

public:
    Engine();
};
```

In C++ this called a constructor.

C++: Objects

We need to turn off our engine.

```cpp
class Engine {
private:

public:
    Engine();
    ~Engine();
};
```

In C++ this called a destructor.

C++: Objects

We need to accelerate our engine.

```cpp
class Engine {
private:

public:
    Engine();
    ~Engine();

    void Accelerate(float HowMuch);
};
```

We have added a class member function to accelerate.

C++: Objects

Accelerating increases how fast the engine going through a cycle so we need to keep track of the RPMs. But this is a quantity that is internal to the engine so let's make it a private member.

```
class Engine {
private:
    float RPM;
public:
    Engine();
    ~Engine();

    void Accelerate(float HowMuch);
};
```

Private members can not be accessed outside of the class.

C++: Objects

To actually move the car we need the transmission to know what the engine RPMs are, but we made it a private member. We can use a getter.

```cpp
class Engine {
private:
    float RPM;
public:
    Engine();
    ~Engine();

    void Accelerate(float HowMuch);
    float Get_RPM();
};
```

Getters and setters are a common practice to modify and read private members.
The Accelerate function we wrote earlier is actually a setter

C++: Objects

Here is our engine class.

```
class Engine {
private:
    float RPM;
public:
    Engine();
    ~Engine();

    void Set_RPM(float HowMuch);
    float Get_RPM();
};
```

C++: Lab 2

In your new myclass.hpp file define a class called fArray.
We want this class to contain an array of N floats, where N is a variable.
It will need a default constructor and a destructor.
What else should fArray have if we want to be able do things with it?

Work with your neighbors and develop your fArray class. We will use this class throughout the remainder of this course.