Scope

PHYS2G03

McMaster University

Scope

- **Scope** means where something is known to the code/compiler
- Every piece of code that uses something you made, like a variable, struct or a function must know about it
- If you define it once inside a brace { } the rest of the code in that block knows about it
- If you define it at the top of a file, all code in the file knows.

somefile.cpp

```
int  VariableX;
float MyFunction( float VariableY ) {
{
     VariableX = 10;    VariableY = sin(1.0)+20.;
     for (;;) {
          int VariableZ;
          VariableZ = VariableZ+1;
     }
}
float OtherFunction( int VariableA, int VariableB ) {
     VariableX = VariableA+VariableB;
}
```

```
int  VariableX;
float MyFunction( float VariableY ) {
{
     VariableX = 10;    VariableY = sin(1.0)+20.;
     for (;;) {
          int VariableZ;
          VariableZ = VariableZ+1;
     }
}
float OtherFunction( int VariableA, int VariableB ) {
     VariableX = VariableA+VariableB;
}
```

Scope of VariableX
a global variable

```
int  VariableX;
float MyFunction( float VariableY ) {
{
     VariableX = 10;    VariableY = sin(1.0)+20.;
     for (;;) {
          int VariableZ;
          VariableZ = VariableZ+1;
     }
}
float OtherFunction( int VariableA, int VariableB ) {
     VariableX = VariableA+VariableB;
}
```

Scope of VariableY MyFunction

VariableY is an argument to MyFunction

```
int  VariableX;
float MyFunction( float VariableY ) {
{
     VariableX = 10;    VariableY = sin(1.0)+20.;
     for (;;) {
          int VariableZ;
          VariableZ = VariableZ+1;
     }
}
float OtherFunction( int VariableA, int VariableB ) {
     VariableX = VariableA+VariableB;
}
```

Scope of VariableZ for (;;)

VariableZ is a local variable

Variables

- You can define variables as **global** (for a whole file). This is usually considered bad, lazy programming
- Global variables provide a sneaky backdoor way to pass around data to other files and functions
- Good programming practice is to use explicit arguments to send data

Variables

- If the use is very limited (like one loop) you might define a variable just for that loop
- The variable literally no longer exists outside the loop
- In general variables no longer exists outside their scope. The memory they used is recycled and their data is lost.

Scope of Count
for (;;)

```
float MyFunction( float VariableY )
{
     VariableY = sin(1.0)+20.;
     for (int Count=0;Count<10;Count++) {
          int VariableZ;
         VariableZ = VariableZ+Count;
     }
    VariableY += Count;   //ERROR,  Count is dead
}
```

Variables

- Typically Variables are defined inside functions. This way the definition is easy to find – it is at the top of the function
- In old C defining at the top is required
- In modern C/C++ you can mix code and definitions inside a function. In this case, the variable only exists from the definition to the end of the block

Scope of VariableZZZ
Last part of MyFunction

```
float MyFunction( float VariableY ) {
{
    VariableY = sin(1.0);

    float VariableZZZ = 5.0;

     VariableY = VariableZZZ+20.;
}
```

VariableZZZ is a local variable

funkyfile.cpp

```
int  FunctionX( int, int );

float MyFunction( float VariableY )
{
     VariableY = sin(1.0)+20.;
}
float OtherFunction( int A, int B ) {
    int VariableC;
    VariableC = FunctionX( A, B );
}
```

Scope of Functions

- C/C++ treats functions a lot like variables
- Standard practice is to define all the functions you intend to use at the top of the file with declarations (also called prototypes)
- The actual code for the function can be in a different file. As long as the compiler sees a prototype it is happy to have you use the function.

```
int  FunctionX( int, int );


float MyFunction( float VariableY
{
     VariableY = sin(1.0)+20.;
}
float OtherFunction( int A, int B ) {
    int VariableC;
    VariableC = FunctionX( A, B );
}
```

Scope of FunctionX is the whole file. This is why headers go at the top

```
float MyFunction( float VariableY ) {
{
     VariableY = sin(1.0)+20.;
}
float OtherFunction( int A, int B ) {
    int  FunctionX( int, int );
    int VariableC;
    VariableC = FunctionX( A, B );
}
```

Unusual to see: Scope of FunctionX limited to just OtherFunction

Declaration vs. actual function

- A Declaration is just a Prototype, a short summary of the function for the compiler.
- It only shows what it returns and the kinds of arguments it expects
- This is enough to compile.
- The Declaration can appear in many files.
- To actually link and make a program, actual code for the function must be included in one (and only one) of the files

```
int  FunctionX( int, int );

int FunctionX(int a, int b)
{
    int c;
    c = a*a + b*b;

    return c;
}
```

Prototype of FunctionX

Actual Code for FunctionX

```
int  FunctionX( int a, int b);

int FunctionX(int a, int b)
{
    int c;
    c = a*a + b*b;

    return c;
}
```

Prototype of FunctionX
Can include the names but not necessary, all the compiler needs is the types (int, float etc…)

Functions and return values

When a function is used, the compiler replaces the function name with its return value in that expression

e.g. For a function defined as follows

```
int myfunction( int x ) {
    return x*x;
}
```

Then when we use it,

```
y = myfunction(x)+2;
```

is equivalent to

```
y = x*x + 2;
```

(sometimes literally, the compiler may decide this is more efficient than having separate function code)

Functions and return values

When you define a function as:

int myfunction( )

You are promising it will return an integer

float otherFunction( )

promises to return a float (real number)

void LameFunction( )

promises to return nothing. In this case you don't put a return in that function code

Void function (no return)

```
void printfunction ( string greet ) {
    std::cout << greet << “\n”;
}

int main()
{
        printfunction( “Hello World!”);
        return 0;
}
```

Functions and arguments

When you define a function as:

int myfunction(int a, float x)

It expects to be given an integer and float

e.g. y = myfunction(1, 2.0);

void otherfunction(string b, int c)

Expects a string and an integer

e.g. otherfunction(“Hello”, 42);

Objects: structs, classes

- User defined types like a struct or class are usually defined once per file, at the top
- It is important that struct definitions are identical every where or the compiler doesn't think they are the same thing
- **It is good practice to include these in every file to ensure every bit of code agrees**

Struct

somefile.cpp

```
struct usertype1 { float x,y,z; };
float myfunction ( usertype1  V ) {
    float mag;
     mag=sqrt(V.x*V.x+V.y*V.y+V.z*V.z);
     return mag;
}
float otherfunction( int a ) {
    struct usertype2 { int a; float x };
    usertype1  VariableX;
    usertype2  VariableY;
   VariableX.x = VariableY.x;
}
```

Struct

```
struct usertype1 { float x,y,z; };
float myfunction ( usertype1  V ) {
    float mag;
     mag=sqrt(V.x*V.x+V.y*V.y+V.z*V.z);
     return mag;
}
float otherfunction( int a ) {
    struct usertype2 { int a; float x };
    usertype1  VariableX;
    usertype2  VariableY;
   VariableX.x = VariableY.x;
}
```

Scope of usertype1 Any function in the file can use it. Also good to put at the top (e.g. include it in a header .h file)

Struct

```
struct usertype1 { float x,y,z; };
float myfunction ( usertype1  V ) {
    float mag;
     mag=sqrt(V.x*V.x+V.y*V.y+V.z*V.z);
     return mag;
}
float otherfunction( int a ) {
    struct usertype2 { int a; float x };
    usertype1  VariableX;
    usertype2  VariableY;
   VariableX.x = VariableY.x;
}
```

Scope of usertype2 Because it was defined in otherfunction it can only be used there

What about stuff in more than one file?: Struct or class

- Often the actual code in multiple functions wants to use a user defined type:

```
struct vector { float x,y,z };
```

- Every file with code that uses it needs the definition of vector at the top so the compiler knows what a vector is
- You can type it in by hand once at the top of every file or you can put it in a header file, “vector.h” and include it automatically

```
struct vector { float x,y,z; };
```

vector.h
defines vector type