Parallel Computing with GPUs

An Introduction to C Part 2 – Functions and Scoping

The University Of Sheffield.

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

- Functions and scoping
  - Differentiate a declaration from a definition
  - Recognise the implications of variable scoping
  - Identify appropriate usage of the keywords `extern` and `static`

Functions

- Function definition

```
return-type function-name(optional-const argument-type argument-name, …)
{
    definitions;

    statements;

    return return-value or expression;
}
```

- Arguments are always passed by value
- No return type implies void (return can be omitted)

Declaration vs Defintion

- *A **declaration** introduces an identifier and describes its type, be it a type, or function. A **declaration** is what the compiler needs to accept references to (i.e. uses of) that identifier. E.g.*

```
extern int a;
int sum(int a, int b);
extern int sum(int a, int b);
```

- A ***definition*** actually instantiates/implements this identifier (and allocates memory for it). It's ***what the linker needs*** in order to link references to those entities. These are definitions corresponding to the above declarations:

```
int a;
int a = 1;
int sum(int a, int b){ return a + b; }
extern int sum(int a, int b) { return a + b; }
extern int a = 1;
```

More Examples: https://www.geeksforgeeks.org/understanding-extern-keyword-in-c/

Scoping

```c
#include <stdio.h>

int square(int a)
{
    return a*a;
}


int main()
{
    int result;
    result = square(a);

    printf("Square of 4 is %i", result);
    return 0;
}

int a = 4;
```

- Scoping lasts from where a variable or function is declared

- What is wrong with the following?

Scoping

```
#include <stdio.h>

int square(int a)
{
    return a*a;
}


int main()
{
    int result;
    result = square(a);   //ERROR

    printf("Square of 4 is %i", result);
    return 0;
}

int a = 4; //DECLARATION AND DEFINITION
```

- Scoping lasts from where a variable or function is **declared**

- What is wrong with the following?

error C2065: 'a' : undeclared identifier

Function Scoping

```c
/* Hello World program */

#include <stdio.h>

int main()
{
    int result, a;
    a = 4;
    result = square(a); //ERROR

    printf("Square of 4 is %i", result);
    return 0;
}

int square(int a)
{
    return a*a;
}
```

- Another example with a function

error C2065: 'square' : undeclared identifier

Function Scoping

```c
/* Hello World program */

#include <stdio.h>

int square(int a)
{
    return a*a;
}

int main()
{
    int result, a;
    a = 4;
    result = square(a);

    printf("Square of 4 is %i", result);
    return 0;
}
```

This works but not always practical

Function Declarations

```c
/* Hello World program */

#include <stdio.h>

int square(int);

int main()
{
    int result, a;
    a = 4;
    result = square(a);

    printf("Square of 4 is %i", result);
    return 0;
}

int square(int a)
{
    return a*a;
}
```

- *A function declaration can be used to forward declare functions*
  - *Sometimes Referred to as a prototype*
  - *Argument names not necessary*
  - *Always considered extern*

A declaration is different to the definition

Variable Declarations

```c
#include <stdio.h>

int square(int);//function declaration
extern int a;   //DECLARATION

int main()
{
    int result;
    result = square(a);

    printf("Square of 4 is %i", result);
    return 0;
}

int a = 4; //DEFINITION

int square(int a)
{
    return a*a;
}
```

- Declarations are not just for functions.
- **`extern`** can be used to declare a variable or function
  - That is defined **elsewhere**
  - **BUT** only defined once

extern

main.c

```c
#include <stdio.h>

//DECLARATIONS
extern int square(int);
extern int a;

int main()
{
    int result;
    result = square(a);

    printf("Square of 4 is %i", result);
    return 0;
}
```

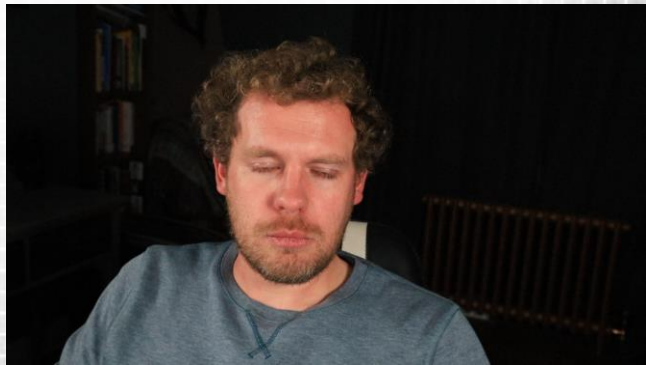
my_maths.c

```c
//DEFINITIONS

int a = 4;

int square(int a)
{
    return a*a;
}
```

- `extern` can declare variables and functions defined in other source modules
  - Resolved by linker



headers

my_maths.h

```
//DECLARATIONS
extern int square(int);
extern int a;
```

my_maths.c

```
//DEFINITIONS
#include "my_maths.h"

int a = 4;

int square(int a)
{
    return a*a;
}
```

main.c

```
#include <stdio.h>
//include
#include "my_maths.h"

int main()
{
    int result;
    result = square(a);

    printf("Square of 4 is %i", result);
    return 0;
}
```

other.c

```
//include
#include "my_maths.h"

int add_a_b_squares(int b)
{
    return square(a) + square(b);
}
```

- Headers can be used to share common declarations

Static

- What is a `static` variable?
  - A static **global** variable or function is visible only in the compilation unit it is defined
    - i.e. No use of `extern` in other source modules
  - A static **local** variable (inside a function) keeps its values between invocations
    - It is defined only once but is declared for lifetime of program

```
void static_test()
{
  int a = 10;
  static int b = 10;
  a += 5;
  b += 5;
  printf("a = %d, sa = %d\n", a, b);
}


int main()
{
  int i;
  for (i = 0; i < 5; ++i)
    static_test();
}
```

```
a = 15, b = ??
a = 15, b = ??
a = 15, b = ??
a = 15, b = ??
a = 15, b = ??
```

What are the values of `b`?

Static

- What is a `static` variable?
  - A static **global** variable or function is visible only in the compilation unit it is defined
    - i.e. No use of `extern` in other source modules
  - A static **local** variable (inside a function) keeps its values between invocations
    - It is defined only once but is declared for lifetime of program

```
void static_test()
{
  int a = 10;
  static int b = 10;
  a += 5;
  b += 5;
  printf("a = %d, sa = %d\n", a, b);
}


int main()
{
  int i;
  for (i = 0; i < 5; ++i)
    static_test();
}
```

```
a = 15, b = 15
a = 15, b = 20
a = 15, b = 25
a = 15, b = 30
a = 15, b = 35
```

What are the values of `b`?

Summary

- Functions and scoping
  - Differentiate a declaration from a definition
  - Recognise the implications of variable scoping
  - Identify appropriate usage of the keywords `extern` and `static`

- Next lecture: Arrays Strings and IO