Compiling your code

Unlike scripts, C programs must be converted into a binary format before they can be run.

This process is called compiling.

For this course we are going to us the GNU project's C compiler: gcc

You can use the ***man*** pages to find out more about it.

Compiling your code

By default, the compiler creates a binary image of your code called a.out.

This is not very descriptive of your code and we should be creating better names to make the purpose of the executable more obvious.

Also, if all we use is a.out, how do you keep more than one binary in any one directory?

The –o option of gcc allows you to specify the name of the binary that is to be created.

Compiling your code

NOTE: be careful naming your binary.

If you accidentally call it by the same name as your source file, the compiler will **OBLITERATE** your source and overwrite it with the binary.

You will then be back at step 1.

Compiling your code

The second compiler option you may want to invoke is the WARNINGS flag.

You can add –Wall (Warnings all) to the compile line to get a more thorough listing of what might wrong with your code.

Compiling your code

On the system used for this course, you will be using the gcc compiler.

Usage: `gcc -Wall -o executable_output_name sourcecode_filename.c`

You may need to add some additional parameters depending on what your code call. For many math heavy applications, you may need to add `-lm` to the compile line.

Running your code

*NIX uses the **PATH** environment variable to search the system for the different commands you enter.

Your work directory is almost certainly NOT in your PATH. The system will not find your executable.

You must therefore prefix your executable name with a directory path

e.g.:

Using a full path:

```
jacques@UBU64vm> /home/jacques/lab3/read_directory
```

or, using a path relative to the current directory:

```
jacques@UBU64vm> ./read_directory
```

DEBUGGING !

My program won't compile!

```
fred.c:7:3: warning: incompatible implicit declaration of
   built-in function 'printf' [enabled by default]
    printf("Hello World !!!\n);
```

You forgot to include the library function required to use the printf function

Solution: man 3 printf (or whatever function is giving you a hard time)

Debugging!

Syntax errors (a.k.a TYPOS):

```
fred.c:7:10: warning: missing terminating " character
   [enabled by default]
   printf("Hello World !!!\n);
          ^
fred.c:7:3: error: missing terminating " character
   printf("Hello World !!!\n);
   ^
fred.c:9:1: error: expected expression before ‘}’ token
 }
 ^
```

Fix: Find the missing character and add it. Maybe you forgot the semi-colon at the end of the line? Forgot a curly brace?

Ooohhh It Compiled !

The compile checks for SYNTAX errors only.

It doesn't check for LOGIC errors

To find these you need to trace the running of your code to find out WHERE in the code it is messing up.

If you don't try to find it on your own, I will ask you to try. It would be too time consuming in a lab environment to do a logic trace on C code for 110 students.

Ooohhh It Compiled !

Tracing execution.

Easiest: put printf() calls in strategic locations in your code:

e.g.

printf("Before the while statement\n");

printf("calling function duplicate\n")

```
int    duplicate(int value)
{
    printf("Inside duplicate: received %d\n",value);
    …
    printf("Leaving duplicate\n");
}
```

Ooohhh It Compiled !

Method 2:

```
#define TRUE 1
#define FALSE 0
…
int main(int argc, char *argv[])
{
int DEBUG;
…
DEBUG = TRUE;

if ( DEBUG )
 {
    printf("Before calling duplicate. Parameter=%d\n",value);
    printf("\tweight=%6.2f\tvolume=%7.2\n",weight,volume);
}
…
}
```

Ooohhh It Compiled !

You get the dreaded “Segmentation error”

Fix: you have tried to access memory that doesn’t belong to you:

- Check if you run off the end of an array
- You stuff something too big into something too small. (strncpy())
- You are using an uninitialized pointer.

Ooohhh It Compiled !

There's garbage showing up in your variable

- Could happen with a char array

C requires a NULL character to define the end of a sequence of char. If you don't have a NULL in the string you are copying, the previous content of the array are still present.

Unless you remove them:

```
bzero(student_name,sizeof(student_name));
```

Unless you add a NULL to the string you are copying in.

```
strncpy(student_name,"Emilio Segre\0",13);
```

Ooohhh It Compiled !

- It is IMPERATIVE that you research the compile time errors. Even with errors, the compiler might give you an executable. The quality/validity of that image should WORRY you.
- Run time errors are LOGIC errors. You know your code best. Having me or someone else look at your code for 20 minutes to find a small error isn’t fair to everyone else.
- Having said that, if after 15 minutes of REAL effort you haven’t found it, I do love a puzzle!