CMSC 104 - Lecture 4
John Y. Park, adapted by C Grasso

Introduction to C

Introduction to C

Topics

- Brief History of Programming Languages & C
- The Anatomy of a C Program
- Compilation
- Using the gcc Compiler
- 104 C Programming Standards and Indentation Styles

History of Programming Languages & C

- Machine code (aka "binary")
  - Raw sequence of binary patterns

    1011010111001011

    1011010110101010
- Assembly "language"
  - Gave human-friendly syntax to machine code:

```
MOV     1200, R0
SUB     1202, R0
MOV     R0, 1200
```

History of Programming Languages & C

- Early high-level languages
  - COBOL
    - SUBTRACT B FROM A GIVING C
    - MULTIPLY C BY 2 GIVING D
  - FORTRAN

    C = A - B

    D = C * 2

    H = SQRT((S1 * S1) + (S2 * S2))

History of Programming Languages & C

- Another early high-level language
  - LISP
    - (lambda (a)
      (mapcar (func '+)
      (cons (car (car a)) (car (cadr a)))))

History of C

- Derived from… "B"!
- Design goals were for C to be:
  - **Efficient**
    - Fast
  - **Close to the machine**
    - I.e., it could directly manipulate the CPU's memory to control hardware-level functions
  - **Structured**
    - A true high-level language with sophisticated control flow, data structures

History of C

- UNIX was recoded from Assembler to C
  - Most operating systems were written in Assembler
- C is written in C!
  - Of course, first versions were written in Assembler
  - Ritchie had great inspiration for a Trojan horse

Does Programming Language Choice Matter?

- Short answer: "Yes, but…"
- C:
  - ```
    main( ) {
      printf("hello, world");
    }
    ```
- COBOL:
  - ```
    MAIN SECTION
    DISPLAY "hello, world"
    STOP RUN.
    ```
- Fortran77:
  - ```
    PROGRAM HELLO
    PRINT*, 'hello, world'
    END
    ```
- Lisp:
  - ```
    (defun helloworld ()
      (print "hello, world") )
    ```
- English:
  - Hello, world.
- Spanish:
  - Hola mundo
- French:
  - Salut le Monde
- Greek:
  - Γεια σου κόσμε

Writing C Programs

- A programmer uses a **text editor (not the same as a word processor!)** to create or modify files containing C code.
- Program code is also known as **source code**.
- A file containing source code is called a **source file**.

A Simple C Program

```
#include <stdio.h>

int main ( )

{

    printf (“Hello, World”) ;

}
```

- Create a file in CMSC104 / hw3 called hello.c and type this program into it.

Computers don't understand letters

- Computers can only "see" numbers
  - It doesn't know what a letter is
- Each letter is represented as an 8-bit number
  - Several different codes are in use
  - Most well-known is ASCII code
    - http://www.asciitable.com/
- To display character codes in vi editor
  - :%!xxd

A Simple C Program… to a Computer

m	a	i	n	(	)	{	\n	\t	p	r	i	n	f	(	"
h	e	l	l	o	,		w	o	r	l	d	"	)	;	\n
}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

- Just a stream of characters that is meaningless to the computer.

- So, after a C source file has been created, the programmer must **invoke the C compiler** and **linker** before the program can be **executed** (**run**).

3 Stages of Compilation

Stage 1: **Preprocessing**

- Performed by a program called the **preprocessor**
- Main purposes:
  - Performs extra processing before compiling
  - Creates a new version of the source code in memory containing the modified version of the code
    - Your source code as stored on disk is not modified.

3 Stages of Compilation (con't)

Stage 2: **Compilation**

- Performed by a program called the **compiler**
- Translates the preprocessor-modified source code into **object code (machine code)**
  - Each **.c** file will be compiled & saved into a **.o** file
  - For example, **hello.c** will be compiled into **hello.o**

3 Stages of Compilation (con't)

Stage 2: **Compilation**

- Checks for **syntax errors** and **warnings**
  - If any compiler errors are received, no object code file will be generated.
  - The compiler may issue warnings, but will still generate the object code if there are no errors

```
gcc   -Wall  –c  hello.c
```

3 Stages of Compilation (con't)

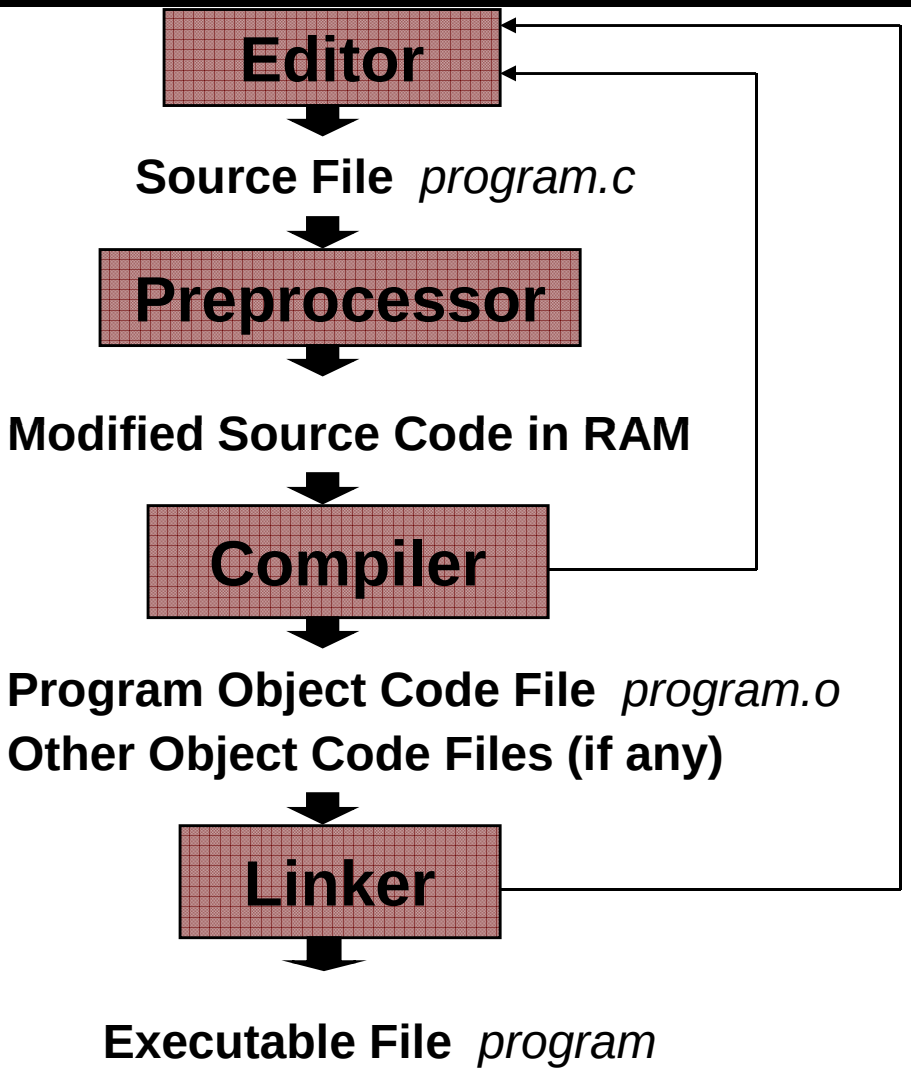
Stage 3: **Linking**

- Combines the program object code with other object code to produce the executable file.
- The other object code can come from the **Run-Time Library**, other libraries, or object files that you have created.
- Saves the machine executable code to another file
  - If any linker errors are received, no executable file will be generated.

```
gcc   -Wall  -o hello  –c  hello.c
```

Program Development Using gcc

`gcc –o program -c program.c`



A Simple C Program

```
/* Filename:     hello.c
 * Author:       Brian Kernighan & Dennis Ritchie
 * Date written:1978
 * Description:  This program prints the greeting
                 "Hello, World!"
 */

#include  <stdio.h>

int main ( )
{
    printf ("Hello, World!\n") ;
    return 0 ;
}
```

Anatomy of a C Program

program header comment

preprocessor directives (if any)

```
int main ( )
{
    statement(s)
    return 0 ;
}
```

Program Header Comment

- A **comment** is descriptive text used to help a *reader* of the program understand its content.
- All comments must begin with the characters **/*** and end with the characters ***/**
- These are called **comment delimiters**
- The program header comment always comes first.

Preprocessor Directives

- Lines that begin with a # in column 1 are called **preprocessor directives** (**commands**).

- **#include <stdio.h>**
  - Copies the contents of the file **stdio.h** at this point in the code.
  - This header file was included because it contains information about the printf ( ) function that is used in this program.

int main ( )

- Every program must have a **function** called **main**. This is where program execution begins.
- main() is placed as the first function in the source code file for readability.
- The **reserved word** "int" indicates that main() **returns** an integer value.
- The parentheses following "main" indicate that it is a function.

The Function Body

- Every function takes the form:

```
type name(arguments)
{
   statements
}
```

- A minimal function is:

```
dummy() { }
```

The Function Body

```
int main ( )
{
    printf (“Hello, World! \n”) ;
    return 0 ;
}
```

printf ("Hello, World! \n") ;

- This line is a C **statement**.
  - Notice that this line ends with a semicolon.
  - All statements in C end with a semicolon.

- It is a **call** to the function **printf ( )** with a single **argument - ** namely the **string** "Hello, World! ".
  - Even though a string may contain many characters, the string itself should be thought of as a single quantity. It is everything between the double quotes

return 0 ;

- **int main ( ) …**
  - indicates that the function **main()** returns an integer value back to whoever called it
- **return 0 ;**
  - tells it to return a value of **0**
  - in main(), a value of **0** indicates that the program ran successfully.

Another C Program

```
1. /**************************************************
2. ** File: message.c
3. ** Author: Joe Student
4. ** Date: 9/15/06
5. ** Section: 0105
6. ** E-mail: jstudent22@umbc.edu
7. **
8. ** This program prints a cool message to the user.
9. **************************************************/
```

Another C Program

```
#include <stdio.h>
int main()
{
   printf("Programming in CMSC104 is fun. \n") ;
   printf("C is a really cool language! \n") ;
   return 0 ;
}
```

What will the output be?

What does the \n do ?

Using the C Compiler at UMBC

- Invoking the compiler is system dependent.
  - At UMBC, we have two C compilers available, **cc** and **gcc**.
  - For this class, we will use the **gcc** compiler as it is the compiler available on the Linux system.

Invoking the gcc Compiler

At the Linux prompt, type

`gcc –Wall -c program.c`

- ***`program.c`*** is the source file to compile
- **-Wall** is an option to turn on all compiler **warnings** (best for new programmers).
- output will be **program.o**

The Result : An Executable

`gcc -Wall –o` *`program`* `-c` *`program.c`*

- If there are no programming errors in ***`program`*`.c`**, this command produces an executable file name ***`program`***
  - If you do not specify **`–o hello,`** **`t`**he compiler will produce an executable file name **a.out**

- To execute the program, at the prompt, type

  ./program

Good Programming Practices

- C programming CMSC104 Coding Standards and CMSC104 Indentation Styles are available on the 104 course Web page on the Homework tab.
- You are expected to conform to these standards for all programming projects in this class and in CMSC 201.
  - This will be part of your grade for each project!
- The program just shown conforms to these standards, but is uncommented.
- Subsequent lectures will include more "Good Programming Practices" slides.