CSA2090: Systems Programming

Introduction to C

Lecture 1: Introduction

Dr. Christopher Staff

Department of Computer Science & AI

University of Malta

Introductions…

- Dr. Chris Staff
- Rm 402
- Open office hours, but prefer appointment
- cstaff@cs.um.edu.mt
- http://www.cs.um.edu.mt/~cstaff

Course details

- Forms part of Systems Programming, over two semesters
- Assessment: assignment, exam
- For C, 7 lectures + 3 lab sessions
- Joel Azzopardi takes lab sessions
- You must already know BSD UNIX/SunOS, and csh, ksh, or bash

Course details

- Reference books…
  - Love, T. ANSI C for Programmers on UNIX Systems. Cambridge University Engineering Dept. http://www-h.eng.cam.ac.uk/help/documentation/docsource/teaching_C.pdf
  - Kernighan and Ritchie. Programming in C. Prentice Hall.
  - Deitel and Deitel, C How to Program. Addison-Wesley.
  - UNIX man pages for info on C commands

Aims and Objectives

- Introduction to some syntax
- Compilation stages
- Variables and literals
- Declarations and Definitions

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Prints the average of 2 numbers

C is case sensitive

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Preprocessor commands

Header files…

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Every C program must contain one main function

Execution begins here

{}

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Statements are terminated by semi-colon

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Variables are declared after any { and before other statements

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Comments…

Also // comments till end of line

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Function that returns a value

Specify return type

Declare parameter types

Function call

If returned value is missing compiler will *not* complain

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Value left on stack is placed in answer

Our first program (basics.c)

```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

Program terminates on:

End of main

Exit

Program is interrupted

Program crashes

Exit

- exit statements can return a value
- Although numeric value can be used, and can be useful…
- … frequently better to use EXIT_SUCCESS or EXIT_FAILURE
- defined in stdlib.h as *symbolic constants*

Our first program (basics.c)

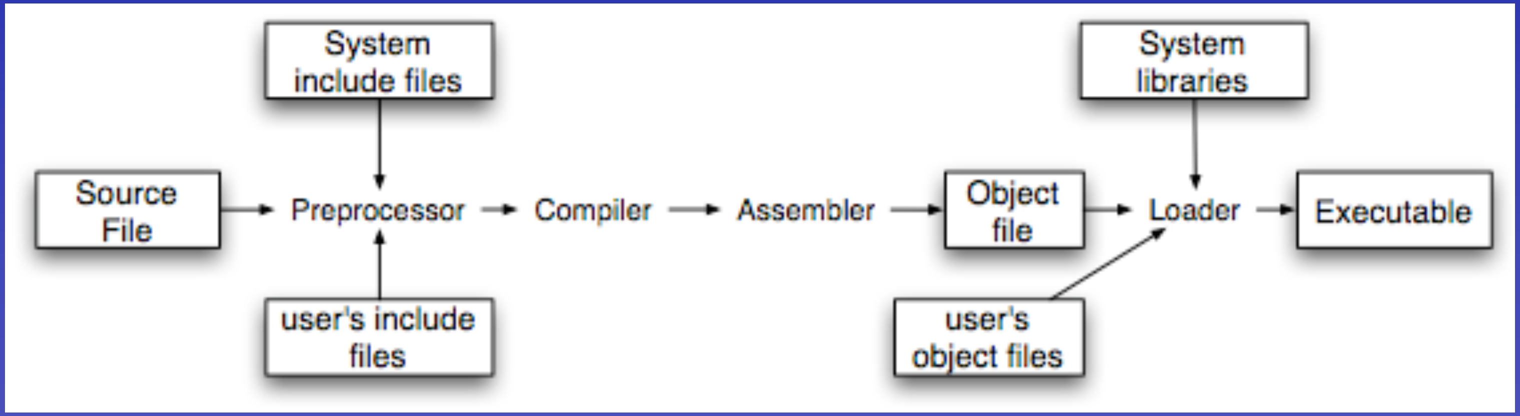
```
#include <stdio.h>
#include <stdlib.h>
int mean(int a, int b)
{
  return (a + b)/2;
}
int main()
{
  int i, j;
  int answer;
  /* comments are done like this */
  i = 7;
  j = 9;
  answer = mean(i, j);
  printf("The mean of %d and %d is %d\n", i, j,
answer);
  exit(0);
}
```

printf…

Compiling the program

- gcc basics.c -o basics

Compilation Stages



Try gcc -v basics.c -o basics to see stages in the compilation process

Variables and literals

- Variables must be declared before use
- Available scalar types are:
  - Char, short, int, long, float, double, and long double
  - Chars and integers can be signed or unsigned
- C will automatically convert between some types
  - E.g., float, double, and int

Variables and literals

```
unsigned int i;
float f = 3.14;
i = f;
f = i * 1.0;
```

- Types can be explicitly changed using *casting*:

```
i = (unsigned int) f;
```

Variables and literals

- sizeof operator will report number of bytes used by data types
  - Which can change across platforms!

Variables and literals

- Scope of variables is normally limited to the {} block in which it is declared
  - Once block is exited, variable is destroyed
- Variables can be declared outside function blocks
  - 'Global' or external variables
  - Scope is remainder of file

Variables and literals

- External variables and functions are visible from other program files, unless declared static
- Variables in functions will retain their values between subsequent calls if they are defined as static (default is automatic)
- See variables.c for examples