CSC180: Lecture 3

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

C

Variables

- Represent storage units in a program
- Used to store/retrieve data over life of program
- Type of variable determines what can be placed in the storage unit
- *Assignment* – process of placing a particular value in a variable
- Variables must be *declared* before they are assigned
- The value of a *variable* can change; A *constant* always has the same value

Naming variables

- When a variable is *declared* it is given a name
- Good programming practices
  - Choose a name that reflects the role of the variable in a program, e.g.
    - Good: customer_name, ss_number;
    - Bad : cn, ss;
  - Don’t be afraid to have long names if it aids in readability
- Restrictions
  - Name must begin with a letter; otherwise, can contain digits or any other characters. C is CASE SENSITIVE! Use 31 or fewer characters to aid in portability

Variable Declaration

- All variables must be **declared** in a C program before the first executable statement! Examples:

```
int a, b, c;
float d;
```

C Variable Names

- Variable names in C may only consist of letters, digits, and underscores and may not begin with a digit
- Variable names in C are case sensitive
- ANSI standard requires only 31 or fewer characters. Enhances portability to follow this rule
- Should be very descriptive

Variable assignment

- After variables are declared, they must (should) be given values. This is called **assignment** and it is done with the ‘=‘ operator. Examples:

```
float a, b;
int c;
b = 2.12;
c = 200;
```

C Data Types

Basic C variable types

- There are four basic data types in C:
  - char
    - A single byte capable of holding one character in the local character set.
  - int
    - An integer of unspecified size
  - float
    - Single-precision floating point
  - double
    - Double-precision floating point

char variable type

- Represents a single *byte* (8 *bits*) of storage
- Internally char is just a number
- Numerical value is associated with character via a *character set*.
  - ASCII character set used in ANSI C

int variable type

- Represents a signed integer of typically 4 or 8 bytes (32 or 64 bits)
- Precise size is machine-dependent

float and double variable types

- Represent typically 32 and/or 64 bit real numbers
- How these are represented internally and their precise sizes depend on the architecture. We won’t obsess over this now.

Declaring variables

- All variables must always be *declared* before the first executable instruction in a C program
- Variable declarations are always:
  - var_type *var_name;*
    - int age;
    - float annual_salary;
    - double weight, height; /* multiple vars ok */
- In most cases, variables have no meaningful value at this stage. Memory is set aside for them, but they are not meaningful until assigned.

Assigning values to Variables

- Either when they are declared, or at any subsequent time, variables are assigned values using the “=“ operator.
- Examples

```
int age = 52; //joint declaration/assignment
double salary;
salary = 150000.23;
age = 53; //value may change at any time
```

Assignment, cont.

- Be careful to assign proper type – contract between declaration and assignments must be honored
  - int x=2.13 /* what is the value of x? */
  - double x = 3; /* is this ok? */
  - char c = 300; /* ??? */
- General advice
  - Don’t obsess too much over this at beginning
  - Keep it simple, stick to basic data types
  - We will be more pedantic later in the course

C Program Anatomy

```
/* description of program */
```

```
#include <stdio.h>
/* any other includes go here */
```

```
int main(){
 /* program body */
return 0;
}
```

- Let’s learn more about the structure of “program body”

Program Body - declarations

- Always begins with all variable declarations. Some examples:

```
int a, b, c;  /* declare 3 ints named a,b,c */
int d, e;     /* similar to above in two steps */
int f;
int g = 1, h, k=3;
double pi = 3.1415926;
```

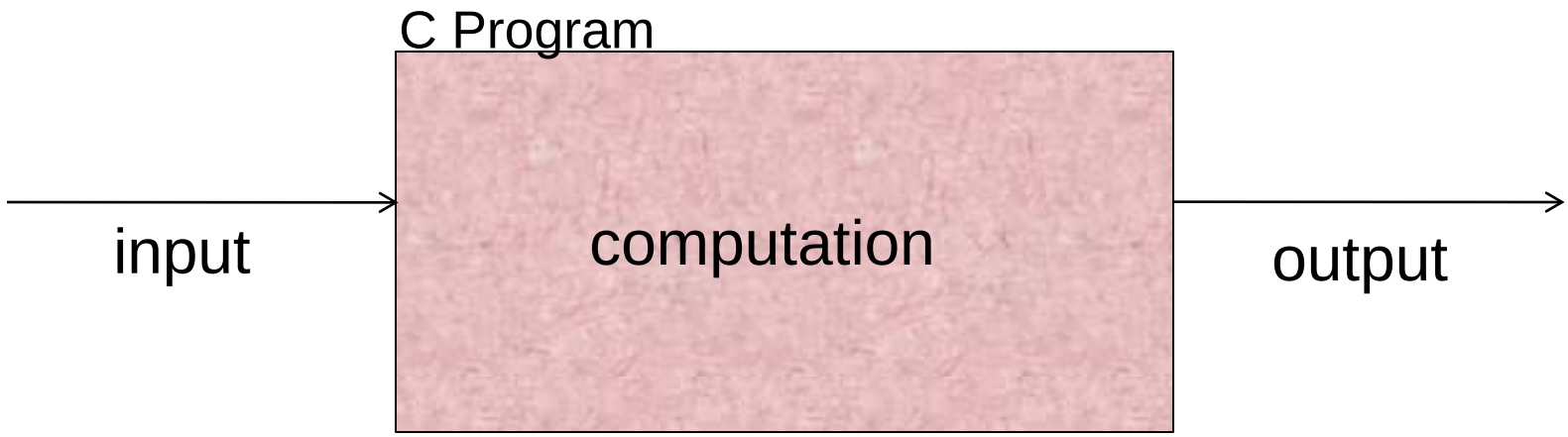
Statements

- Note: all *statements* end with a semicolon!
  - Statements can (with a few exceptions) be broken across lines or ganged on a single line
- Commas separate multiple declarations
- Blank lines have no effect
- Extra spaces between *elements of a statement* has no effect.
- *Comments* are ignored by the compiler

Program Body – Executable Statements

- *Executable statements* always follow variable declarations/initializations
- *Executable statements* include any valid C code that is not a declaration, ie valid C code to do things like:
  - *"multiply the value of a by 10 and store the result in b"*
  - *"add 1 to the value of j and test whether it is greater than the value of k"*
  - *"store 5.2 in the variable x"* (ie assignment)
  - *"print the value of x,y, and z, each on a separate line"*

Interactive Programs



printf()

- Sends output to *standard out*, which for now we can think of as the terminal screen.
- General form

  printf(format descriptor, var1, var2, …);
- format descriptor is composed of
  - Ordinary characters
    - copied directly to output
  - Conversion specification
    - Causes conversion and printing of next *argument* to printf
    - Each conversion specification begins with %

Printf() examples

- Easiest to start with some examples
  - printf(“%s\n”, “hello world”);
    - Translated: “print hello world as a *string* followed by a newline character”
  - printf(“%d\t%d\n”, j, k);
    - Translated: “print the value of the variable j as an integer followed by a tab followed by the value of the variable k as an integer followed by a new line.”
  - printf(“%f : %f : %f\n”, x, y, z);
    - English: “print the value of the floating point variable x, followed by a space, then a colon, then a space, etc.

More on format specifier

- The format specifier in its simplest form is one of:
  - %s
    - sequence of characters known as a ***String***
    - **Not a fundamental datatype in C (really an *array* of char)**
  - **%d**
    - **Decimal integer (ie base ten)**
  - **%f**
    - **Floating point**
- **Note that there are many other options. These are the most common, though, and are more than enough to get started.**

What do program instructions look like?

- A simple program has at least these three main parts
  - variable declaration
  - variable initialization
  - main body

First C Program

A Simple C Program

```
2    A first program in C */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "Welcome to C!\n" );
8
9     return 0;
10 }
```

```
Welcome to C!
```

- Comments
  - Text surrounded by `/*` and `*/` is ignored by computer
  - Used to describe program
- **`#include <stdio.h>`**
  - Preprocessor directive
    - Tells computer to load contents of a certain file/library
  - **`<stdio.h>`** allows standard input/output operations

A Simple C Program, Cont.

- **`int main()`**
  - C programs contain one or more functions, exactly one of which must be **`main`**
  - Parenthesis used to indicate a function
  - **`int`** means that **`main`** "returns" an integer value
  - Braces (**`{`** and **`}`**) indicate a block
    - The bodies of all functions must be contained in braces

A Simple C Program: Printing a Line of Text

- **`Return 0;`**
  - A way to exit a function
  - **`Return 0`**, in this case, means that the program terminated normally

Second C Program

variables: value vs. address

```
#include <stdio.h>

int main()
{

  int X;

  X = 20;
  printf("The value of X is %d. The address of X is %d\n“, X, &X);

  return 0;

}
```