CMPT 102
Introduction to Scientific Computer Programming

Components of a C program

© Janice Regan, CMPT 102, Sept. 2006

Introduction

- ***Computer program****:* an ordered sequence of instructions whose objective is to accomplish a task.
- ***Programming:*** process of planning and creating a program
- ***Programming Language****:* a set of symbols, special words (semantics), and rules for using those symbols and special words (syntax) to create statements to implement the ordered sequence of instructions to accomplish a task

© Janice Regan, CMPT 102, Sept. 2006

Programs and Libraries

- Programs are divided into smaller modules called functions. Functions can
  - be separately tested and reused by many programs
  - Call other functions
  - Become part of libraries
- Every program has a main function.
  - The main function controls the highest level of abstraction of the program.
  - The main function may call other functions to complete lower level tasks (like evaluating an exponential)
  - Many commonly needed methods are supplied in libraries as part of the C language, some less commonly needed methods are available as additional add on libraries

© Janice Regan, CMPT 102, Sept. 2006

The Components of a C Program

- Each C program includes a series of functions or program units.
  - Every C application program includes a main function, sometimes called a main program
  - There may be other functions used by the main program.
    - Functions from C libraries
    - Functions written by the user
- For your first programs we will illustrate only a main function

© Janice Regan, CMPT 102, Sept. 2006

Hello World: your first program

```
/* My first C program */
/* make the computer print the string “Hello world” */

/* connect to any necessary libraries */
#include <stdio.h>

/* the main function, implements the algorithm to solve the problem*/
/* the main function may use functions from included libraries */
    int main ( )
    {
        printf("Hello, world!\n");
    }
```


How to create a main function

```
// main method, implements the algorithm to solve the problem
// main method may use functions from included libraries
    int main ( )
    {
        Body of the main function
    }
```

- The start of the main method is indicated by the statement
  int main ( )
- The body of the main method (the code that accomplishes the task of the main method) is enclosed in {}.
- The convention used in this course in to put each of the {} on a separate line.

Libraries

- A main function, or any other function may wish to use functions in the built in libraries of the C language or in user developed or application libraries
- To use such functions our program must indicate that the libraries are being used and provide a way to access the functions in the various libraries at link time.

© Janice Regan, CMPT 102, Sept. 2006

Hello World: your first program

```
/* My first C program */
/* make the computer print the string “Hello world” */

/* connect to any necessary libraries */
#include <stdio.h>

/* the main function, implements the algorithm to solve the problem*/
/* the main function may use functions from included libraries */
    int main ( )
    {
        printf("Hello, world!\n");
     }
```

© Janice Regan, CMPT 102, Sept. 2006

Using Libraries in C

- A C program that wants to use a library starts with one or more include directives
  - #include <*Library_Reference*>
  - #include "*Library_Reference*"
- Here *Library_Reference* indicates the name of a file containing definition of all the functions in the library.
  - A function definition tells us about the function but does not include the body of the function
  - For the main function we looked at the definition would be *int main();*

Syntax of include directive

#include <*Library_Reference*>

#include "*Library_Reference*"

- " " indicates that the files location should be completely specified by the programmer. If only a filename is given that file must be in the same directory as the code being compiled.
- < > indicates that the file is part of the C++ language itself and can be located by the preprocessor, compiler and linker without programmer intervention. The location may be controlled using the library path.

Example: Standard Libraries in C

- *#include <stdio.h>*
  - The # at the beginning of the line indicates that the command is for the C preprocessor (***a preprocessor directive***)
  - The C preprocessor puts the contents of the file stdio.h into your program instead of the line of code above
    - The file stdio.h contains definitions of the functions in the input/output library (not the bodies of the functions)
    - These functions in the input output library can be successfully called by your program only if the contents of the file stdio.h is in your compiled program (object file)

Example: User Libraries in C

- ***#include "mylib.h"***
  - The # at the beginning of the line indicates that this is a C preprocessor directive
  - The " " indicate that
    - the file mylib.h is not part of the standard C language (It is usually part of a user or application defined library)
    - The location of the file mylib.h is specified by the programmer
    - The location is given relative to the directory in which the file containing the code in which the include statement is embedded

Comments

- Comments: explanations of the purpose of series of C expressions (lines of C code)
  - Comments begin /* and end */
    - /* This is a comment */
  - Comments can appear on their own line
  - Comments can appear on the same line as an expression
    - A = b + c; /* add b to c and store the result in A */
- Comments are essential to clearly explain the purpose of your code and the methods of implementation used
  - When you use another programmer's code, another programmer tries to use your code, or you try to use your own code after months or years it is much easier to figure out how things work.

© Janice Regan, CMPT 102, Sept. 2006

Hello World: your first program

```
/* My first C program */
/* make the computer print the string "Hello world" */

/* connect to any necessary libraries */
#include <stdio.h>

/* the main function, implements the algorithm to solve the problem*/
/* the main function may use functions from included libraries */
    int main ( )
    {
        printf("Hello, world!\n"); /*Prints Hello world to the screen*/
     }
```

© Janice Regan, CMPT 102, Sept. 2006

Reserved Words and Identifiers

- Token: smallest individual unit of a program
- Types of tokens include
  - ***Reserved words*** called *keywords,* names for things inherent to the high level language *(C)*
  - ***Identifiers*** or names for things defined by the user
    - names of variables (x or y, in expression y = x;)
    - Names of functions (main, sin, tan, exp …)
    - Names of constants, string literals, structures, enumerations …

© Janice Regan, CMPT 102, Sept. 2006

Reserved words in C

- A ***reserved word*** is a part of the computer language we are using. In is a special word that gives the compiler a specific instruction it can understand
  - auto, break, case, char, const, continue
  - default, do, double, else, enum, extern, float
  - for, goto, if, int, long, main, register, return
  - short, signed, sizeof, static, struct, switch
  - typedef, union, unsigned, void, volatile, while
- A ***reserved word*** cannot be used as an user specified identifier

© Janice Regan, CMPT 102, Sept. 2006

Hello World: your first program

```
/* My first C program */
/* make the computer print the string “Hello world” */

/* connect to any necessary libraries */
#include <stdio.h>

/* the main function, implements the algorithm to solve the problem*/
/* the main function may use functions from included libraries */
    int main ( )
    {
        printf("Hello, world!\n");
     }
```


C Identifiers

- Names of program components: variables, constants, structures, functions, enumerations
- Must contain only:
  - Letters (upper and lower case)
  - Digits (0-9)
  - The underscore character (_)
- Identifiers must begin with a letter or an _
- Identifiers beginning with an underscore are used for specific purposes and should be avoided in general use
- C is case sensitive. The identifier Toy is not the same as the identifier toy

More About C Identifiers

- The number of significant characters in an identifier is determined by the operating system and compiler used.
- An identifier must not be a reserved word
- An identifier should be descriptive. It makes your code much easier to understand and maintain
  - sideLength is much clearer than sl
  - But keep your names a reasonable length for code clarity

Yet More About C Identifiers

- Can divide identifiers into two groups
  - Standard Identifiers:
    - Identifiers defined by the compiler that are not parts of the C language definition
    - This include identifiers used as names of functions in the standard C libraries
  - User (programmer) defined Identifiers:
    - Names the programmer chooses for quantities, functions, etc used in their own programs

Program - sum two numbers

```
/*  This program computes the sum two numbers                    */

#include <stdio.h>
int main(void)
{
   /*  Declare and initialize variables.  */
   double number1 = 473.9,  number2 = 45.7,  sum;

   /*  Calculate sum.  */
   sum = number1 + number2;

   /*  Print the sum.  */
   printf("The sum is %5.2f \n", sum);

   /*  Exit program.  */
   return 0;
}
```

Standard identifier

Illegal Identifiers

- Prog#1 The # is not a letter digit or _
- 1stprogram cannot begin with a digit
- auto cannot be a keyword

Which Are Legal C Identifiers?

- cycle
- A!star
- int
- Score
- Y-Z
- Trial#2
- This_One
- $MyProgram
- FOR
- 3constant_values
- “MaxScores”
- _External_Prog
- Mary’s
- StarChart

© Janice Regan, CMPT 102, Sept. 2006

Components of a C Program

- More types of tokens (smallest individual units of a program) include:
  - ***String Literal Constants*** or constant labels used in expressions
    - Bob in expression Name = ” Bob”;
  - ***Literal Constants*** or fixed numbers used in expressions
    - 3 in expression y = x + 3;
  - ***Variables and Constants***

© Janice Regan, CMPT 102, Sept. 2006

Constants and Variables

- Each constant or variable represents a specific item in the problem the program solves and is associated with
  - a memory cell used to hold its value
  - a unique identifier or name
  - a data type
- The value of a constant does not change during execution of the program
- The value of a variable may be changed during the execution of a program.

Types of Constants and Variables

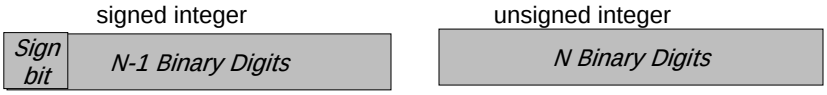
- A Data Type is
  - *A set of values plus a set of operations on those values*
  - A crucial concept on modern programming
- The data type of a variable or constant also determines how the variable's value will be represented in memory
- Variables of several types can have numerical values
- Variables of other types have values that are characters, or logical values.

Types with Numerical Values

- Two classes of numerical values
  - Integer and floating point
- Integer values are whole numbers
  - No fractional part
  - 1, 12354, 68, -123
  - C Types: int, unsigned int, long int, short int, long unsigned int, …
- Floating point values have a fractional part
  - 987.4, -0.332, 0.123, 3.14159
  - C types: float, double, long double
- Integers and floating point numbers are represented differently inside the computer

Representations of Integers

- Integers are represented by a string of binary digits.



- Binary digits are 0's and 1's
- An unsigned integer has values between 0 and 2^N-1.
  - 1 1 1 1 1 1 1 1 8 digits N=8
  - $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

    128 + 64 + 32 + 16+ 8 + 4 + 2 + 1 = 255
- A signed integer has values between -2^{N-1} and 2^{N-1}-1
- Converting from a unsigned integer with M bits to a signed integer may require a signed integer of M+1 bits
- long int variables represent numbers with twice as many digits as int variables

Floating point numbers

- A floating point number is represented by a mantissa and an exponent
- For example $1.23456 * 10^{12}$

Mantissa = 0.123456	*Exponent = 13*

- The mantissa will be represented by N binary digits. It can take the 2^N-1 values. These values will be at equal intervals between 0 and 0.99999999. These 2^N-1 values are the representable values of the floating point representation
- For a long double there will be more representable values than for a double because more bits are allocated to the mantissa. (the long double will have N approximately 2X larger)

Declaring the variables you need

- The first step in building the body of your main function is the declaration of the variables you will need to execute your algorithm.
- To declare a variable we need to
  - choose an identifier as the name of the variable.
  - choose what the type of the variable is.
- Values of all declared variables and constants should be declared at the start of the function
  - Variables and constants can also be given values, (be initialized within declarations

Declarations of variables in C

- TYPE var_identifier;
  - declare names for constants and variables (memory cells used in program)
  - TYPE may be int, float, double, char….
- TYPE var_identifier1, var_identifier2, …;
  - multiple variables of the same type may be declared on the same line
- TYPE may be also use modifiers, short, long …
  - long int or long, short int or short, long double
  - unsigned int signed int unsigned long int

Program - sum two numbers

```
/*  This program computes the sum two numbers                    */
#include <stdio.h>
int main(void)
{
   /*  Declare variables.  */
   double number1,  number2:
   double sum;

   /*Initialize variables */
   number2 = 45.7;
   number1 = 473.9

   /*  Calculate sum.  */
   sum = number1 + number2;
  /*  Print the sum.  */
   printf("The sum is %5.2f \n", sum);
  /*  Exit program.  */
  return 0;
}
```

Declaration of Variables

- Define names of memory locations which can hold values that may change (variables)
- Can declare one or more variables and constants of the same type in one statement.
- Programming style for this course: Declare one variable or constant per statement (line of code)
- Examples:
  - int myintegervalue;
  - char myinitial;
  - unsigned int counter;
  - double scalefactor;
  - float temperature;
  - unsigned long int secondcounter;

© Janice Regan, CMPT 102, Sept. 2006

Literal Data

- Literals
  - Examples of literal constants:
    - 2 /*Literal constant int*/
    - 5.75 /* Literal constant double*/
  - Examples of string literal constants
    - "Z" /* String literal constant char*/
    - "Hello World" /* String literal constant string*/
- Cannot change values during execution
- Called "literals" because you "literally typed" them in your program!
- Literals do not have identifiers (names), they are used directly in the C code

© Janice Regan, CMPT 102, Sept. 2006

Program - sum two numbers

```
/*  This program computes the sum two numbers                    */

#include <stdio.h>
int main(void)
{
   /*  Declare and initialize variables.  */
   double number1 = 473.9,  number2 = 45.7,  sum;

   /*  Calculate sum.  */
   sum = number1 + number2;

  /*  Print the sum.  */
   printf("The sum is %5.2f \n", sum);

  /*  Exit program.  */
  return 0;
}
```

literal constants

String literal constants

Literal Constants

- Sometimes we use numbers in the equations we wish to code in the C language
  - A=C-4; 4 is an int constant
  - C=B*3.6; 3.6 is a double constant
  - X=Y/3.6E+03 3.6E+03 is a float constant (3600)
  - A=B+5U; 5U is an unsigned integer constant
  - C=X+33L; 33L is a long integer constant
- Integer literal constants are considered to be of type int unless otherwise indicated
- Floating point constants are considered to be of type double unless otherwise indicated.

Named Constants

- Naming your constants
  - Literal constants are "OK", but provide little meaning
    - e.g., seeing 45.7 in a program, tells nothing about what it represents
- Use named constants instead
  - Meaningful name to represent data
    const double FIRST_NUMBER = 45.7;
    - Called a "declared constant" or "named constant"
    - Now use it's name wherever needed in program
    - Added benefit: changes to value result in one fix

Initialization of Variables

- Memory locations associated with variables should have their values defined before the start of program execution.
- Using initialized variables produces unpredictable results
- Variables can be initialized
  - when they are declared
  - later (at any time) in the execution of the program
- It is good programming practice to initialize variables at the start of your program
  - The best practice is to initialize variable at the start of your executable code, that is after all variables have been declared.

Program - sum two numbers

```
/*  This program computes the sum two numbers                    */
#include <stdio.h>
int main(void)
{
   /*  Declare variables.  */
   double number1,  number2:
   double sum;
   /*Initialize variables */
   number2 = 45.7;
   number1 = 473.9

   /*  Calculate sum.  */
   sum = number1 + number2;
  /*  Print the sum.  */
   printf("The sum is %5.2f \n", sum);
  /*  Exit program.  */
  return 0;
}
```

```
/*  Declare and initialize variables.  */
    double number1 = 473.9;
     double number2 = 45.7,  sum;
```


Initialization of Variables

- Examples of initialization at time of declaration of variables:
  - long int myintegervalue = -12L;
  - char myinitial = ′a′;
  - unsigned int counter = 31U;
  - double scalefactor = 17.346798574;
  - float temperature = 3.475;
  - long double Temperature = 7.75;

Components of a C Program

- More types of tokens (smallest individual units of a program) include:
  - ***Other separators*** blanks, tabs, line feed, blank lines …
  - ***Operators*** actions used to combine constants
    - = and + in the expression y = x + THREE;

C: Binary Arithmetic Operators

- A Binary Operator operates on two arguments
  - \+ addition
  - \- subtraction
  - * multiplication
  - / division
  - % modulus or remainder (only for integers, 5%2=1)
- Evaluated left to right

Unary Arithmetic Operators in C

- A Unary Operator operates on one argument
  - \+ positive
  - \- negative
  - ++ increment
  - -- decrement
  - ~ ones complement
- Evaluated right to left

© Janice Regan, CMPT 102, Sept. 2006