CC-112 Programming Fundamentals

Introduction to C Programming - I

Nazar Khan
Department of Computer Science
University of the Punjab

Typical Structure of a C Program

```
introductory comments
preprocessing directives
int main()
{
  statements;
}
```

Example 1: Printing a line of text

- C uses some notations that may appear strange to people who have not programmed computers.
- The following program prints a line of text.

```
1 // A first program in C.
2 #include <stdio.h>
3
4 // function main begins program execution
5 int main( void )
6 {
7    printf( "Welcome to C!\n" );
8 } // end function main
```

Comments

- For documenting programs and improving program readability.
- Begin with //
- Ignored by the C compiler – do not cause the computer to perform any action
- Usually describe the purpose of the program.
- Help *other people* read and understand your program.
- Multiline comment is enclosed with /* ... */.

`#include` Preprocessor Directive

- Directs the C preprocessor to include contents the file stdio.h.
- stdio.h is part of the C standard library.
- Contains functions for input and output operations such as printf().

The `main()` function

- Part of every C program.
- Parentheses () after `main` indicate that main is a program building block called a *function*.
- C programs contain one or more functions, one of which *must* be main.
- Execution begins at main()
- Functions can return information. Keyword `int` to the left of main() indicates that main "returns" an integer.
- Functions can receive information when they're called upon to execute. The void in parentheses here means that main does not receive any information.
- A left brace begins the body of every function.
- Corresponding right brace ends each function.

The `main()` function

- Pair of braces and statements between them is called a *block*. The block is an important program unit in C.

Good Programming Practices:

1. Every function should be preceded by a comment describing the function's purpose.
2. Add a comment to the line containing the right brace, }, that closes every function, including main.

Output Statement

- `printf` stands for print formatted.
- `printf()` instructs the computer to print on the screen the string of characters marked by the quotation marks.
- A string is sometimes called a character string, a message or a literal.
- Every statement must end with a semicolon (aka statement terminator).

Escape Sequences

Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizontal tab. Move the cursor to the next tab stop.
\a	Alert. Produces sound or visible alert without changing cursor position.
\\	Backslash. Insert a backslash character in a string.
\”	Double quote. Insert a double-quote character in a string.

Compiler errors

- Sometimes we forget to close braces, parantheses, or multiline comments.
- Sometimes we forget to put the semicolon after a statement.
- The compiler detects such errors and informs us.

The Linker and Executables

- Standard library functions like printf are *not* part of the C programming language.
- Compiler cannot find a spelling error in printf.
- Compiler provides space in the object program for a *call* to the library function.
- But the compiler does not know where the library functions are – the linker does.
- After compilation, the linker runs, locates the library functions, and inserts the proper calls to these library functions in the object program.
- Now the object program is complete and ready to be executed (called an executable).

Linker errors

- If the function name is misspelled (“print” instead of “printf”), the linker will spot the error.
- Because it will not be able to match the name “print” in the C program with the name of any known function in the libraries.

Example 2: Asking user for two integers and adding them

```
// Addition program.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int integer1; // first number to be entered by user
   int integer2; // second number to be entered by user
   printf( "Enter first integer\n" ); // prompt
   scanf( "%d", &integer1 ); // read an integer
   printf( "Enter second integer\n" ); // prompt
   scanf( "%d", &integer2 ); // read an integer
   int sum; // variable in which sum will be stored
   sum = integer1 + integer2; // assign total to sum
   printf( "Sum is %d\n", sum ); // print sum
} // end function main
```

Variables and Variable Definitions

- The lines

```
int integer1; // first number to be entered by user
int integer2; // second number to be entered by user
```

  are *definitions* of *variables*.
- A variable is a location in memory where values can be stored for use by a program.
- Every variable has
  - a name,
  - a type,
  - a size, and
  - a value.

Define Variables Before They Are Used

- All variables must be defined with a name and a data type before they can be used.
- Variable definition can be anywhere in main() but must be before the variable's first use.

Multiple definitions in one statement

- We can also define in one line as

```
int integer1, integer2;
```

- But that makes it difficult to associate comments with each of the variables.

Identifiers and Case Sensitivity

- A variable name in C can be any valid *identifier*.
- An identifier is a series of characters consisting of letters, digits and underscores that does not begin with a digit.
- C is case sensitive – uppercase and lowercase letters are different in C, so a1 and A1 are different identifiers.

Prompting Messages

- The command

```
printf( "Enter first integer\n" ); // prompt
```

  - displays the literal "Enter first integer" on standard output (usually the monitor), and
  - positions the cursor to the beginning of the next line.
- This message is called a prompt because it tells the user to take a specific action.

Getting input from the user

- The command

```
scanf( "%d", &integer1 ); // read an integer
```

  - reads from standard input (usually the keyboard),
  - converts it into integer data type, and
  - stores it in the variable called `integer1`.
- `scanf` takes two arguments
  - *format control string* `%d` indicating that input should be an integer from the decimal number system.
  - address of memory location named "integer1" indicated by the *address operator* `&`.

Assignment Statement

- The assignment statement

```
sum = integer1 + integer2; // assign total to sum
```

calculates the total of variables `integer1` and `integer2` and assigns the result to variable `sum` using the *assignment operator* =.

Printing with a Format Control String

- The command

```
printf( "Sum is %d\n", sum ); // print sum
```

  prints the character string Sum followed by the value of variable sum on the standard output.

- Alternatve command

```
printf( "Sum is %d\n", integer1 + integer2 ); // print su
```

  will also work. Variable sum will not be required then.