ICS103 Programming in C

Lecture 2: Introduction to C (1)

Outline

- Overview of C
  - History & Philosophy
  - Why C?
  - What's Missing?
- General form of a C program
- C Language Elements
  - Preprocessor Directives
  - Comments
  - The "main" function
  - Variable Declarations and Data Types
  - Executable Statements
  - Reserved Words
  - Identifiers

History & Philosophy

- C is developed in 1972 by Dennis Ritchie at the AT&T Bell Laboratories for use with the Unix.
- The most commonly used programming language for writing system software.
- Machine independent: by minimal change in source code, can be compiled in a wide variety of platform and operating system.

Why C?

- Many, many companies/research projects do all their programming in C.
- Looks good on your resume.
- Small, compact code.
- Produces optimized programs that run faster.
- Low-level access to computer memory via machine addresses and pointers.
- Low level (BitWise) programming readily available.
- Can be compiled on a variety of computers.

What’s Missing?

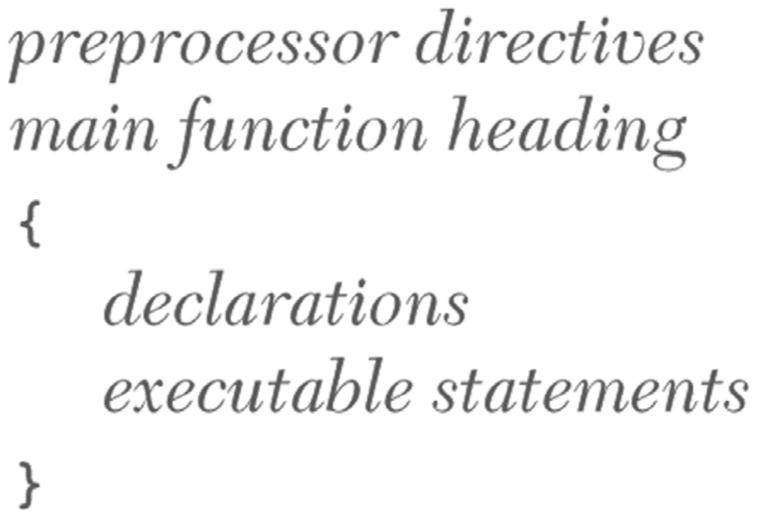
- Poor error detection which can make it difficult to use for the beginner
  - No automatic garbage collection.
  - No bounds checking of arrays and allocated memory segments.
  - No exception handling.
- No native support for multithreading and networking, though these facilities are provided by popular libraries
- No standard libraries for graphics and several other application programming needs

A Simple, Example C Program

```
/* helloworld.c */
#include <stdio.h>
int main(void) {
  printf("Hello World!\n");
  return(0);
}
```

- Every C program has a `main` function.
- `printf` is also the name of a function
- This program can use the `printf` function, because of the line `#include <stdio.h>` in the source code.

General Form of a C program



```
preprocessor directives
main function heading
{
    declarations
    executable statements
}
```

- Preprocessor directives are instructions to C Preprocessor to modify The text of a C program before compilation.
- Every variable has to be declared first.

- Executable statements are translated into machine language and eventually executed.
- Executable statements perform computations on the declared variables or input/output operations.

Preprocessor Directives

```
/* Converts distances from miles to kilometers */
#include <stdio.h>                        /* printf, scanf definitions */
#define KMS_PER_MILE 1.609                /* conversion constant */
int main(void)
{
    double miles,      //distance in miles
           kms;        //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Preprocessor Directives

- Preprocessor directives are commands that give instructions to the C preprocessor.
- Preprocessor is a system program that modifies a C program prior to its compilation.
- Preprocessor directives begins with a #
  - Example. #include or #define

#include

- `#include` is used to include other source files into your source file.
- The `#include` directive gives a program access to a library.
- **Libraries** are useful functions and symbols that are predefined by the C language (standard libraries).
  - Example: You must include `stdio.h` if you want to use the `printf` and `scanf` library functions.
  - `# include<stdio.h>` insert their definitions to your program before compilation.

#define

- The `#define` directive instructs the preprocessor to replace each occurrence of a text by a particular constant value before compilation.
- `#define` replaces all occurrences of the text you specify with value you specify
  - Example:

```
#define KMS_PER_MILES 1.60
#define PI 3.14159
```

Comments

```
/* Converts distances from miles to kilometers */

#include <stdio.h>                      /* printf, scanf definitions */
#define KMS_PER_MILE 1.609              /* conversion constant */

int main(void)
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Comments

- Comments provide supplementary information making it easier for us to understand the program, but are ignored by the C compiler.
- Two forms of comments:
  - /* */ - anything between them with be considered a comment, even if they span multiple lines.
  - // - anything after this and before the end of the line is considered a comment.
- Comments are used to create **Program Documentation**
  - Information that help others read and understand the program.
- The start of the program should consist of a comment that includes programmer’s name, date of the current version, and a brief description of what the program does.
- **Always Comment your Code!**

The “main” Function

```
/* Converts distances from miles to kilometers */

#include <stdio.h>                      /* printf, scanf definitions */
#define KMS_PER_MILE 1.609              /* conversion constant */

int main(void)
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

The “main” Function

- The heading `int main(void)` marks the beginning of the `main` function where program execution begins.
- Every C program has a `main` function.
- Braces ({,}) mark the beginning and end of the body of function main.
- A function body has two parts:
  - **declarations** - tell the compiler what memory cells are needed in the function
  - **executable statements** - (derived from the algorithm) are translated into machine language and later executed by the compiler.

Variables and Data Types

```
/* Converts distances from miles to kilometers */

#include <stdio.h>                        /* printf, scanf definitions */
#define KMS_PER_MILE 1.609                /* conversion constant */

int main(void)
{
    double miles,     //distance in miles
           kms;       //equivalent distance in kilometers

    //Get the distance in miles
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

Variables Declarations

- **Variable** – The memory cell used for storing a program's data and its computational results
  - Variable's value can change.
  - Example: `miles`, `kms`
- **Variable declarations** –Statements that communicates to the compiler the names of variables in the program and the kind of information they can store.
  - Example: `double miles`
    - Tells the compiler to create space for a variable of type `double` in memory with the name `miles`.
  - C requires you to declare every variable used in the program.

Data Types

- **Data Types**: a set of values and a set of operations that can be performed on those values
  - **int**: Stores integer values – whole numbers
    - 65, -12345
  - **double**: Stores real numbers – numbers that use a decimal point.
    - 3.14159 or 1.23e5 (which equals 123000.0)
  - **char**: An individual character value.
    - Each char value is enclosed in single quotes. E.g. ‘A’, ‘*’.
    - Can be a letter, a digit, or a special symbol
  - Arithmetic operations (+, -, *, /) and compare can be performed in case of **int** and **double**. Compare can be performed in **char** data.