LECTURE 01

Input and Output

Outcomes

1. To give an introduction of computer/C programming executed under Code::Blocks IDE with GCC compiler.
2. To execute simple C program involving input and output.

Contents

1. Introduction to Computer Programming
2. Introduction to C programming
3. Compile and Execute Complete C programming
4. Basic Structure of C Programming
5. Writing `printf` and `scanf` Statements

1. Introduction to Computer Programming

What is Computer Programming?

Today, most people don't need to know how a computer works. Most people can simply turn on a computer or a mobile phone and point at some little graphical object on the display, click a button or swipe a finger or two, and the computer does something. So how all of these things work? The answer is computer programming. Having a computer device (e.g. laptop, personal computer, smart phone) without computer programming is like having a car without petrol! In formal, computer programming is defined as the process of developing and implementing various sets of instructions to enable a computer to do a certain task. These instructions are considered computer programs and help the computer to operate smoothly.

Programming Languages

A program is a set of instructions that tells a computer what to do in order to come up with a solution to a particular problem. Programs are written using a programming language. A programming language is a formal language designed to communicate instructions to a computer. There are two major types of programming languages: **low-level languages** and **high-level languages**.

(i) Low-Level Languages

Low-level languages are referred to as 'low' because they are very close to how different hardware elements of a computer actually communicate with each other. Low-level languages are machine oriented and require extensive knowledge of computer hardware and its configuration. There are two categories of low-level languages: machine language and assembly language.

Machine language, or machine code, is the only language that is directly understood by the computer, and it does not need to be translated. All instructions use binary notation and are written as a string of 1s and 0s. A program instruction in machine language may look like something like this:

```
10010101100101001111101010011011100101
```

Technically speaking, this is the only language computer hardware understands. However, binary notation is very difficult for humans to understand. This is where assembly languages come in.

Assembly language is the first step to improve programming structure and make machine language more readable by humans. An assembly language consists of a set of symbols and letters. A translator is required to translate the assembly language to machine language. This translator program is called the 'Assembler.' It can be called the second generation language since it no longer uses 1s and 0s to write instructions, but terms like `MOVE`, `ADD`, `SUB` and `END`. Many of the earliest computer programs were written in assembly languages. Most programmers today don't use assembly languages very

often, but they are still used for applications like operating systems of electronic devices and technical applications, which use very precise timing or optimization of computer resources. While easier than machine code, assembly languages are still pretty difficult to understand. This is why high-level languages have been developed.

(ii) High-Level Language

A high-level language is a programming language that uses English and mathematical symbols, like `+`, `-`, `%` and many others, in its instructions. When using the term 'programming languages,' most people are actually referring to high-level languages. High-level languages are the languages most often used by programmers to write programs. Examples of high-level languages are C++, FORTRAN, Java and Python. To get a flavor of what a high-level language actually looks like, consider an ATM machine where someone wants to make a withdrawal of $100. This amount needs to be compared to the account balance to make sure there are enough funds. The instruction in a high-level computer language would look something like this:

```
x = 100
if balance  x:
 print 'Insufficient balance'
else:
 print 'Please take your money'
```

This is not exactly how real people communicate, but it is much easier to follow than a series of 1s and 0s in binary code. There are a number of advantages to high-level languages.

The first advantage is that high-level languages are much closer to the logic of a human language. A high-level language uses a set of rules that dictate how words and symbols can be put together to form a program. Learning a high-level language is not unlike learning another human language - you need to learn vocabulary and grammar so you can make sentences. To learn a programming language, you need to learn commands, syntax and logic, which correspond closely to vocabulary and grammar.

The second advantage is that the code of most high-level languages is portable and the same code can run on different hardware. Both machine code and assembly languages are hardware specific and not portable. This means that the machine code used to run a program on one specific computer needs to be modified to run on another computer. Portable code in a high-level language can run on multiple computer systems without modification. However, modifications to code in high-level languages may be necessary because of the operating system. For example, programs written for Windows typically don't run on a Mac. A high-level language cannot be understood directly by a computer, and it needs to be translated into machine code. There are two ways to do this, and they are related to how the program is executed: a high-level language can be compiled or interpreted.

2. Introduction to C Programming

Brief History of C

C is a highly structured oriented programming language that was originally developed by Dennis Ritchie for the UNIX operating system. It was first implemented in between 1969 to 1973 at Bell Laboratories. C was originated from two programming language, namely BCPL (Basic Combined Programming Language) and B language. BCPL and B were developed in intended as a language to develop operating systems and compilers. BCPL and B are “type less” languages whereas C provides a variety of data types. Therefore, BCPL and B were then replaced by C language.

Why Use C?

C was initially used for system development work, in particular the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Thus it is important for scientists and engineers to have a good understanding of the language to assist them in their computing needs. The C programming language contains only 32 keywords. This makes it a small language when compared to others. It is also very stable, fast, and has been so widely used that the syntax is the basis for many other languages. C has now become a widely used professional language for various reasons:

- One of the early programming languages.
- Easy, simple and reliable to learn.
- C is structured language.
- The standard library concept.
- It produces efficient programs.
- It can handle low-level activities.
- Modern programming concepts are based on C.
- It can be compiled on a variety of computer platforms.
- Ready access to the hardware when needed.

C is often called a “Middle-Level” programming language. This is not a reflection on its lack of programming power but more a reflection on its capability to access the system's low-level functions. Most high-level languages (e.g. FORTRAN) provide everything the programmer might want to do already build into the language. A low-level language (e.g. assembler) provides nothing other than access to the machines basic instruction set. A middle-level language, such as C, probably doesn’t supply all the constructs found in high-level languages but it provides with all the building blocks that the programmer need to produce the results that they want.

C has been around for 30 years, and there is a ton of source code available. This means there's a lot to learn from, and a lot to use. Moreover, many of the issues with the language have been clearly

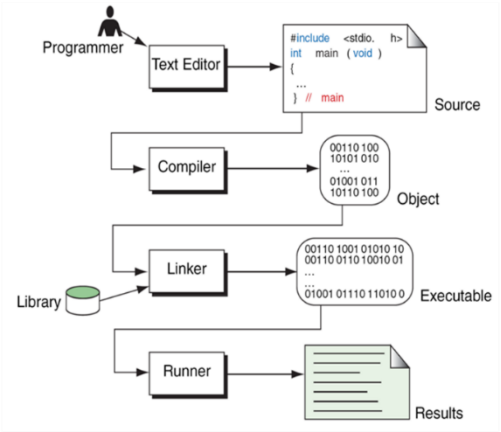
elucidated. It's well understood, and you can find a lot of tutorials available. Plus, with C, you get lots of strong opinions mixed with insights that you can understand.
As a result of its age and its use as the language of system programming for UNIX, C has become something of the lingua franca of programming. C is a great language for expressing common ideas in programming in a way that most people are comfortable with. Moreover, a lot of the principles used in C for instance, loop constructs and variable types will show up in a lot of other languages you learn so you'll be able to talk to people even if they don't know C in a way that's common to both of you.

C is reasonably close to the machine. When you're working with pointers, bytes, and individual bits, things like optimization techniques start to make a lot more sense. There's also utility in knowing exactly how something works underneath the hood. This helps a great deal when something you're trying to do in a higher level language seems way slower than expected, or just doesn't work at all. You also tend to get a better picture of advanced topics like exactly how networking works. A higher level language will make it a little bit simpler, but it'll be harder to understand what's going on, and when things stop working, it's much better to know exactly what's going on so you can fix it. Additionally, if you like computer science as a discipline, or just like knowing how things work learning the details of the system is great fun.

In fact, a lot of fun programming is done in C for instance, if you want to be able to do more than write a simple web app, C is a great language. If you want to write a great, fast game, C is again a great choice. You can write an entire OS in C. It'll be much harder to do so in Java, and nearly impossible in a scripting language. And the language, being succinct as C is, will probably make your fun program more elegant looking to boot.

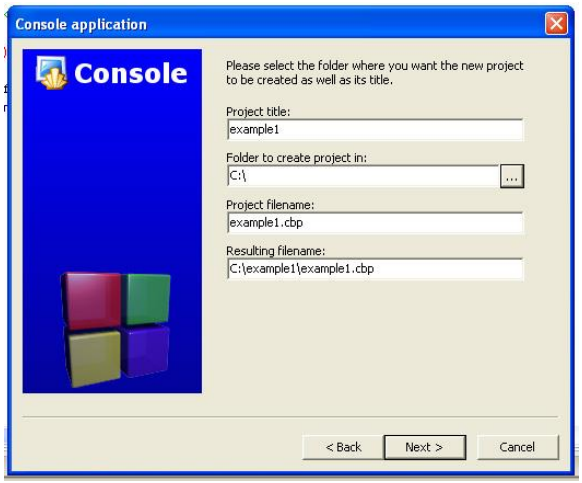
3. Compile and Execute Complete C Programming

In order to run a C program, you need a compiler. Compiler change source code (code written by programmer) to object code (code that computer understands) and creates executable file. Then, the linker link the program with the required library modules. The steps of creating and running C program are illustrated in below figure.

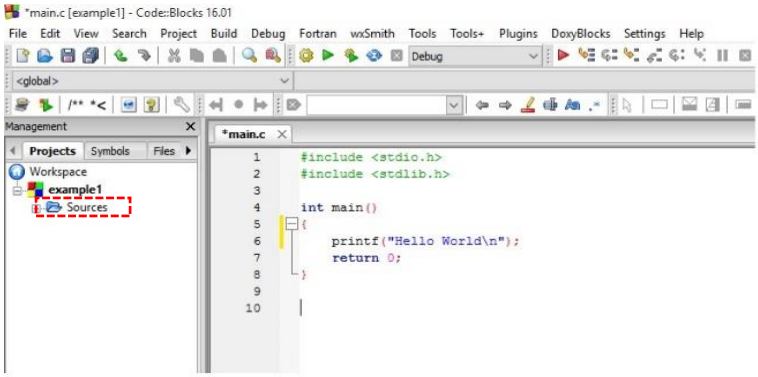


There are many free and professional compilers available. You can refer to https://en.wikipedia.org/wiki/List_of_compilers#C_compilers for list of compilers for C. For this course, GCC compiler is used using Code::Blocks IDE. The main reason of using this compiler because it is free and also can be used in different computer platforms. The steps for installing Code::Blocks to your computer and compile and execute complete C programming are as follow:

1. Open link (http://www.codeblocks.org/downloads/26). Choose your computer operating system. Choose your OS with mingw_setup if you don't have any GCC compiler in your computer. Download the setup file.
2. Install the setup file. Open CodeBlocks program.
3. Click File → New → Project. Select Console application then select C language. The term console means we are using the keyboard (for input) and monitor (for output) for this implementation.
4. Create Folder. Folder is used to store your entire working file. For example in below figure, name your project title as example1. Put your folder path (Folder to create project in) in C:\ drive.



5. Accessing C code (main.c) in example1 project is by clicking Sources folder tab



6. Execute and compile. Click Build → Build and Run or Click F9 button.
7. In order to open the save project file, click File → Open → go to save folder → double click example1.cbp in that directory.

4. Basic Structure of C Programming

Below are few commands and syntax used in C programming to write a simple C program.

```
/*
C Program to demonstrate the simple C programming
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdio.h>
#include <stdlib.h>
const int a;
void display();
int main()
{
    int b;
    printf("Hello World from main function\n");
    display();
    return 0;
}
void display()
{
    printf("Hello World from display function\n");
}
```

```
Hello World from main function
Hello World from display function
```

Lets see all the sections of a simple C program line by line in the following table.

C codes	Explanation

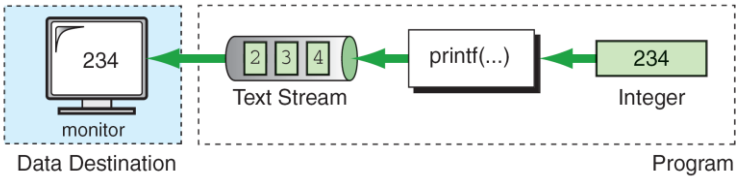
`/*`
`C Program to demonstrate the simple C programming`
`Written by: AFAN, FKP, UMP`
`Date: September 2016`
`*/`
	Text surrounded by (`/*` and `*/`) are comments and it is ignored by computer. It is useful to describe coding.
	(`//`) this is for single line comment.
	(`/* */`) this is a block comment that can covers many lines.

`#include <stdio.h>` `#include <stdlib.h>`	This is a preprocessor command that includes standard input output header file (`stdio.h`) and standard library header file (`stdlib.h`) from the C library before compiling a C program
`const int a;`	Global declarations
`void display();`	Declarations for `display` function
`int main()`	This is the `main` function from where execution of any C program begins. The `main` function must be only one (1)
`{`	This indicates the beginning of the `main` function

C codes	Explanation
`int b;`	Local declarations
`printf("Hello World from main function\n");`	`printf` command prints the output onto the screen
`display();`	Calling display function
`return 0;`	This command terminates C program (`main` function) and `returns 0` (nothing)
`}`	This indicates the end of the `main` function
`void display()`	This is the `display` function (other function) – void means no return value to `main` function. The other function can be more than one function
`{`	This indicates the beginning of the `display` function
`printf("Hello World from display function\n");`	`printf` command prints the output onto the screen
`}`	This indicates the end of the `display` function

5. Writing `printf` and `scanf` Statements

The output formatting function is `printf`. The `printf` function takes a set of data values, converts them to a text stream, and sends the resulting text stream to the standard output (monitor). For example, an integer 234 stored in the program is converted to a text stream and then sent to the monitor. Figure below shows the concept.



General syntax for `printf` is `printf("statements");`. Other than that, we can also print control characters such as tabs (`\t`), newlines (`\n`), and alert (`\a`). Tabs are used to format the output into columns. Newlines terminate the current line and continue formatting on the next line. Alerts sound an audio signal to alert, usually to alert the user to a condition that needs attention. Example:

```
/*C Program to demonstrate the printf
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Hello World from first line\n");
    printf("Hello World from second line\n");
    return 0;
}
```

```
Hello World from first line
Hello World form second line
```

Instead of using two lines of `printf` code, you can also write in a single line like this `printf("Hello World from first line\nHello World from second line\n");`. Other example is:

```
/*C Program to demonstrate the printf
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
```

```
    int b;
    printf("Name:\t\tAli\n");
    printf("Address:\tPekan, Pahang\n");
    return 0;
}
```

```
Name:       Ali
Address:    Pekan, Pahang
```

The use of tab is to make the output organize well into columns. Lets say the tab is not inserted to the above code, what will happened? Here is the output:

```
Name:Ali
Address:Pekan, Pahang
```

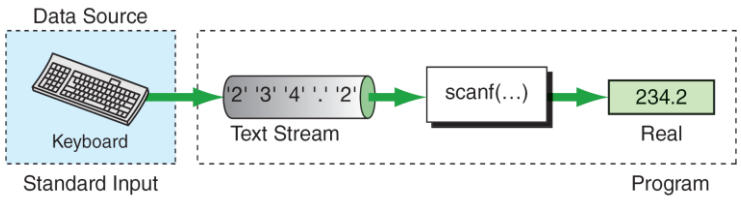
As seen on the above output, it is not being organized well. Therefore, it is important to arrange your output so that it will be easier for user to use your system. To insert data into the stream, we use a conversion specification that contain a start token (`%`). General syntax is `printf("output_format",value_list);`. For this implementation, we use (`%c`) for `char`, (`%s`) for `char string`, (`%hd`) for `short int`, (`%d`) for `int`, (`%ld`) for `long int`, (`%lld`) for `long long int`, (`%f`) for `float`, (`%f`) for `double`, and (`%Lf`) for `long double`. Here is the example:

```
/*C Program to demonstrate the printf
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char name[20] = "ahmad fakhri"; //for multiple characters(string)
    char gender = 'M';              //for single characters
    int age = 20;
    float height = 1.68;
    printf("Name:\t%s\nGender:\t%c\nAge:\t%d\nHeight:\t%.2f\n",name,
            gender,age,height);
    return 0;
}
```

```
Name:   ahmad fakhri
Gender: M
Age:    20
Height: 1.68
```

Notice that, we are using `%.2f` to print `height`. This is meaning that the `height` output will be print rounded to two decimal places. Using `printf` statement, you must aware about the amount of `output_format` (`%`) and `value_list`. Lets say you are printing 3 (three) variables, your `output_format` (`%`) and `value_list` must be three for both of them. Instead of using (`%`) in `output_format`, you can put any wording in this area including control characters such as tabs (`\t`), newlines (`\n`), and alert (`\a`) etc.

The standard input formatting function in C is `scanf` (scan formatting). This function takes a text stream from the keyboard, extracts and formats data, and then stores the data in specified program variables. The concept of this function is illustrated in below figure.



General syntax for `scanf` is `scanf("output_format",value_list);`. As we see with `printf` statement, it is all the same syntax with `scanf` except three things. **(1)** In `output_format`, you cannot insert any other wording including control characters for `scanf` except a start token (`%`). **(2)** scanf requires variable address operator (`&`) in the `value_list`. Using the address operator, if the variable name is `price`, then the `value_list` is `&price`. **(3)** There is no precision in an input conversion specification. For example, in `printf`, we use `printf("%.2f",height);`. In `scanf`, we only allow to do `scanf("%f",&height);`. The input will be taken according to the declaration of variable. Example of `scanf` usage is as follows:

```
/*C Program to demonstrate the scanf
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char name[20];
```

```
    printf("Enter name: ");
    scanf("%s",&name);
    printf("\nInserted name is: %s\n",name);
    return 0;
}
```

```
Enter name: Ahmad
Inserted name is: Ahmad
```

Another important thing to know is `scanf` reads until the maximum number of characters that have been declared or until `scanf` finds a whitespace character. If `scanf` finds a whitespace character enter by user before maximum is reached, it will stops. You have to aware about some of `scanf` rules for example, if you want to ask user to insert multiple parameters, it is recommended to you to use `scanf` line by line in order to avoid miss confusion. For instance, if the program ask user to input name, gender, age, example of miss confusion will happen is as below.

1. `scanf("%s%s%d",&name,&gender,&age);`
   Note that if there are a space between inserted inputs, it would create an error.

2. `scanf("%s %s %d",&name,&gender,&age);`
   To prevent problem no 1., put a space between each of `%`. However, this solution will not be the best solution because if user forget to put a space, it would also create a miss confusion data storage.

```
/*C Program to demonstrate the scanf
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char name[10];
    char gender[10];
    int age;
    printf("Name:\t");
    scanf("%s",name);
    printf("Gender:\t");
    scanf("%s",&gender);
    printf("Age:\t");
    scanf("%d",&age);
    printf("\nYour Inserted Info:\nName:\t%s\nGender:\t%s\nAge:\t%d\n",
          name,gender,age);
```

```
    return 0;
}
```

```
Name:   ahmad
Gender: lelaki
Age:    26

Your Inserted Info:
Name:   ahmad
Gender: lelaki
Age:    26
```