Introduction to C/C++
Lecture 1 - Introduction to Programming

Rohit Sehgal
Nishit Majithia

Association of Computing Activities,
Indian Institute of Technology, Kanpur

rsehgal@cse.iitk.ac.in
nishitm@cse.iitk.ac.in

May 18, 2016
Content Coverage of the Course:

- Elements of C/C++ programming languages
- Data types
- Sequential and conditional execution, Loops
- Arrays, pointers
- Strings and their Manipulation
- Functions
- User Defined Data Types in C
- Migrating to C++, User defined Data Types Classes
- Simplifying Coding Using STL
Course Scheme

- Course Website
  cse.iitk.ac.in/users/aca/sumschool2016/lectures.html
Course Scheme

- Course Website
  cse.iitk.ac.in/users/aca/sumschool2016/lectures.html

- Evaluation
  - One Quiz After first week
  - Two programming Assignment
  - Final Exam at the end of the course
  - Practice problems (Ungraded)
Course Scheme

- **Course Website**
  cse.iitk.ac.in/users/aca/sumschool2016/lectures.html

- **Evaluation**
  - One Quiz After first week
  - Two programming Assignment
  - Final Exam at the end of the course
  - Practice problems(Ungraded)

- **Course Timings**
  -
Suggestions?
Questions?
Computer being an electronic device understand ON/OFF Signals.
Computer being an electronic device understand ON/OFF Signals.

Series of ON/OFF Instructions analogous to 1/0(binary Instructions).
Computer being an electronic device understand ON/OFF Signals.
Series of ON/OFF Instructions analogous to 1/0(binary Instructions).
Binary codes provides way to interact to Computers, Machine Language
Computer being an electronic device understand ON/OFF Signals.

Series of ON/OFF Instructions analogous to 1/0(binary Instructions).

Binary codes provides way to interact to Computers, **Machine Language**

Computer can do intended task, if provided with meaningful string of 0,1(s) as these ON/OFF signals controls different components of computers viz ALU, CU etc.
Computer being an electronic device understand ON/OFF Signals.

Series of ON/OFF Instructions analogous to 1/0 (binary Instructions).

Binary codes provide a way to interact with Computers, **Machine Language**

Computer can do intended tasks if provided with meaningful string of 0,1(s) as these ON/OFF signals control different components of computers viz. ALU, CU etc.

Cumbersome to write a binary string even to add two nos.
Computer being an electronic device understand ON/OFF Signals.

Series of ON/OFF Instructions analogous to 1/0(binary Instructions).

Binary codes provides way to interact to Computers, **Machine Language**

Computer can do intended task, if provided with meaningful string of 0,1(s) as these ON/OFF signals controls different components of computers viz ALU, CU etc.

Cumbersome to write a binary string even to add two nos.

Solution **Assembly language**?
Assembly Language provides Mnemonics.
Assembly Language provides Mnemonics.

E.g ADD, LOAD, STORE, need not to remember binary codes(s) for these Operations.
- **Assembly Language** provides **Mnemonics**.
- E.g ADD, LOAD, STORE, need not to remember binary codes(s) for these Operations.
- **Assembler**, converts Assembly Language to Machine.
- **Assembly Language** provides **Mnemonics**.
  
  E.g ADD, LOAD, STORE, need not to remember binary codes(s) for these Operations.

- **Assembler**, converts Assembly Language to Machine.

- Programming in Assembly, **Why not something similar** to Natural Languages like English.
Languages Above Machine Language

- **Assembly Language** provides **Mnemonics**.
- E.g ADD, LOAD, STORE, need not to remember binary codes(s) for these Operations.
- **Assembler**, converts Assembly Language to Machine.
- Programming in Assembly, Why not something **similar** to Natural Languages like English.
- High level Languages **C, C++, Java, python**
**Compiler** Convert HLL to Assembly.

Programming Computers is now easier.
C/C++

- **Compiler** Convert HLL to Assembly.
- Programming Computers is now easier.

**Example (Compilers for C/C++)**

```bash
$ gcc <filename.c> -o <outputfilename>
$ g++ <filename.cpp> -o <outputfilename>
```

- .c,.cpp are the file extensions for C,C++ source file respectively.
Compiling and Running

- **Linux users** gcc, g++
  
  
  $ sudo apt-get install gcc g++

- **Windows users:**
  
  - Install Virtual Machine with Linux system.
  - OR, Install Mingw,
    
    http://www.mingw.org/

- **We will be working with Linux System**
Our First Program

Example (C)
```c
#include<stdio.h>

int main()
{
    printf("Hello World");
    return 0;
}
```

Example (C++)
```cpp
#include<iostream>
using namespace std;

int main()
{
    cout<<"Hello World";
    return 0;
}
```

- Now onwards we will follow C’s syntax.
# pre-processor Directives

- Replaced before compilation starts.
  
  \$ gcc -E demo.c > preprocessed_file.dat

- `#include <>`, `#include """, includes the content of file in source file.
  
  - these file stores **declarations** of **functions**
  - functions are part of program that does some computations *details in functions lecture*

- others `#define`, `#ifdef`, `#ifndef` etc.
  
  - `#define` - text replacement **blindly**.
  - e.g
The Main function

- Entry point of code execution as program may contain several functions.
- body of any function is enclosed with \{ \}
- return 0 ??, exit status, we will see later.
- writes to standard output (monitor).
- first argument is always a **format specifier**—how the output is printed to screen.
Comments

- Compiler do not read comments.
- // this is a single line comment.
- /* this comment spans multiple lines */
#include <stdio.h>

#define N 10

/* Block
 * comment */

int main()
{
    int i;

    // Line comment.
    printf("Hello World %d", N); // prints to std output
    return 0;
}
Assignments
1. Write a C program that prints: This is Quotation mark- ”
2. Write a C program that prints: Printing % is not that easy
3. Write a C program to find square of 23 using Macro function.
4. Write a C program to find the product of two numbers, and do check whether it do runs correctly for the following input-product(10+20,10+20).
5. Preprocessor directives are replaced before compilation try the gcc compiler -E option with different directives and observe the output file.
The End