

# CS101: INTRODUCTION TO COMPUTING

Instructor: Samrat Mondal

# COURSE STRUCTURE

- Two components-
  - Theory: CS101 (3-0-0-6)
  - Lab: CS110 (0-0-3-3)
- Separate grading for each of the above components
- CS101 will be evaluated based on followings-
  - Class Tests: 20%
  - MidSem: 30%
  - EndSem: 50%
- Consult lab instructor for CS110 evaluation policy

## COURSE INFORMATION:

### ○ Books:

- A. Kelley and I. Pohl, *A Book on C*, 4<sup>th</sup> Ed.
- P.J.Deitel & H.M.Deitel , *C How To Program*, 5<sup>th</sup> Ed
- Kernighan & Ritchie, *The C Programming Language*, 2<sup>nd</sup> Ed.
- B. Gottfried, *Programming with C*, Schaum's Outlines Series

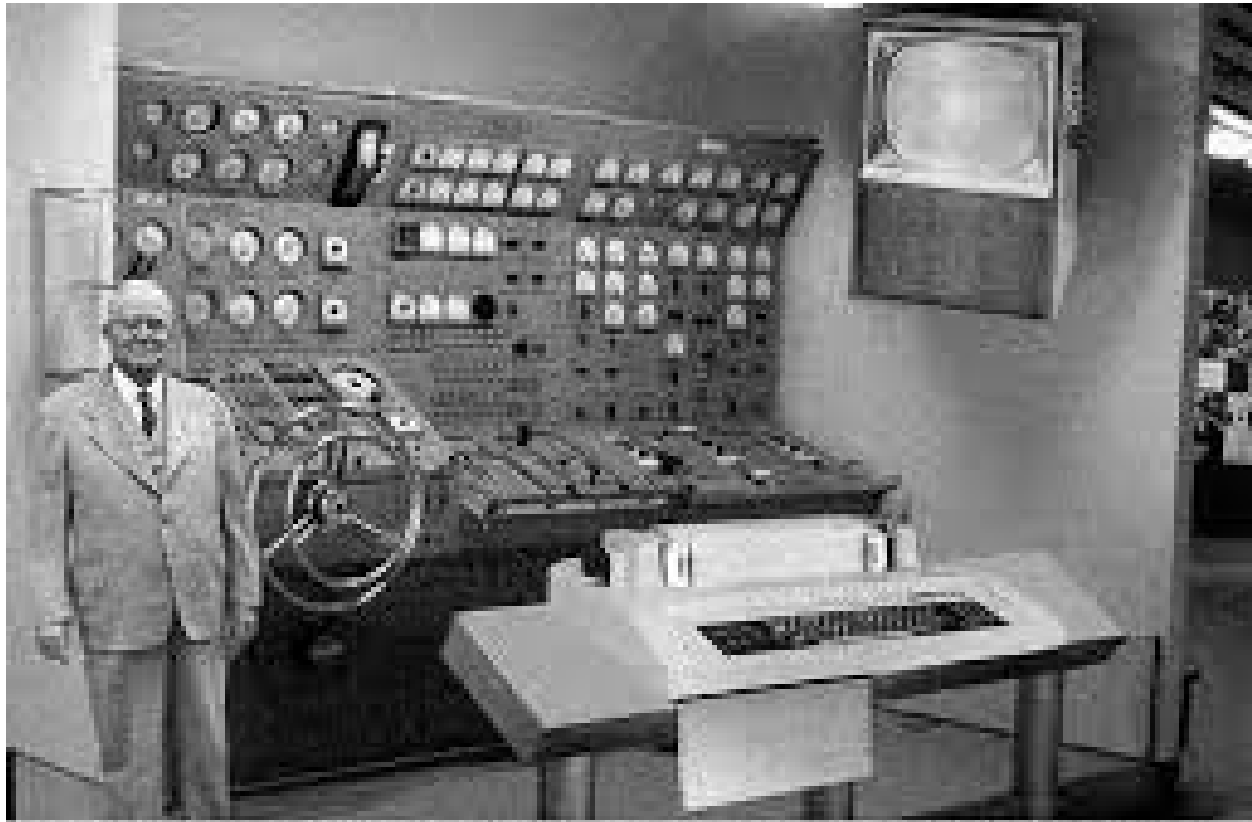
### ○ Syllabus:

- Computer Fundamentals,
- C Programming (*for details check course website*)

- Course Page:
  - <http://172.16.1.3/~samrat/CS101>
- Contact information:
  - Email: [samrat@iitp.ac.in](mailto:samrat@iitp.ac.in)
  - Office phone: 8063
  - Chamber: R405, Block 3
- Office Hours (Meeting time outside class):
  - Wednesday: 5pm to 6pm
- Attendance is absolutely mandatory

# INTRODUCTION

RAND Corporation Predicted in 1954: How the computers will look like in 2004

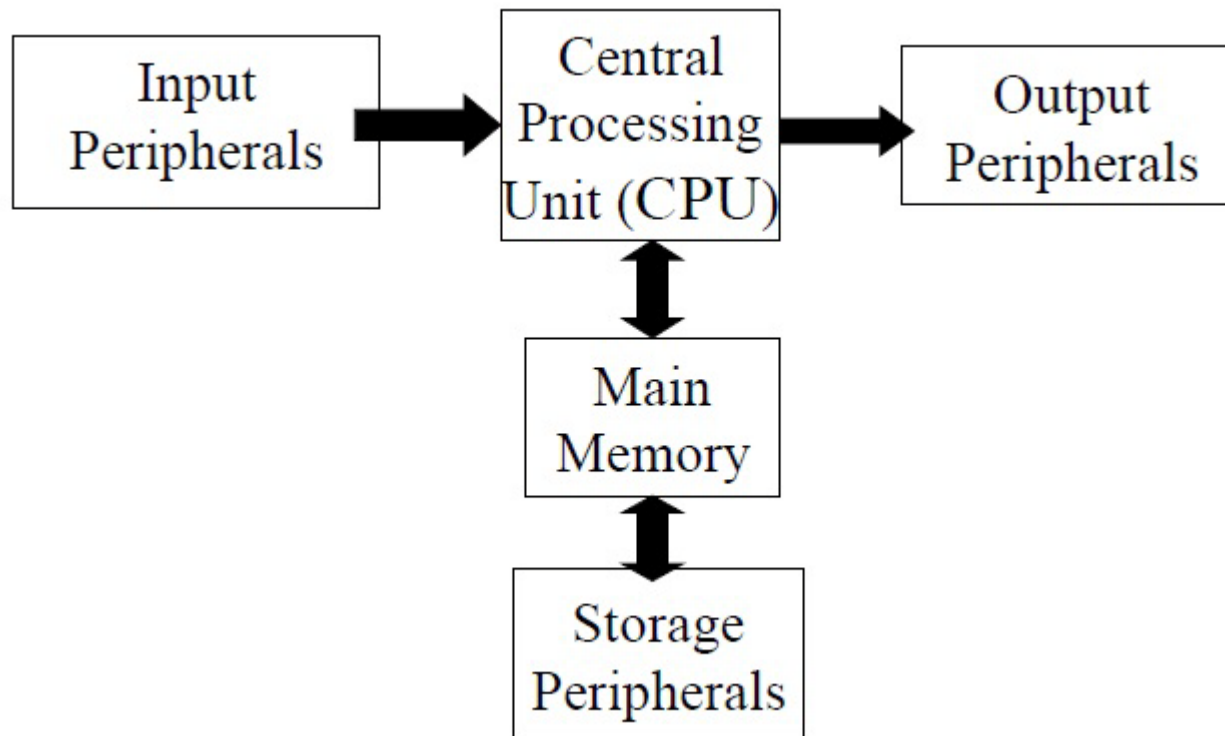


*Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However, the needed technology will not be commercially feasible for the average home. Also, the scientists readily admit that the computer will require our yet invented technology to actually work, but it starts from now, reliable programs is required to solve these problems. With always interface and the Fortran language, the computer will be easy to use.*



Modern Computer

# BASIC BUILDING BLOCKS OF A COMPUTER



# I/O AND PERIPHERALS: EXAMPLES

- Input Devices
  - Keyboard, Mouse, Digital Camera
- Output Devices
  - Monitor, Printer, Speaker
- Storage Peripherals
  - Magnetic Disks: hard disk
  - Optical Disks: CDROM, CD-RW, DVD
  - Flash Memory: pen drives



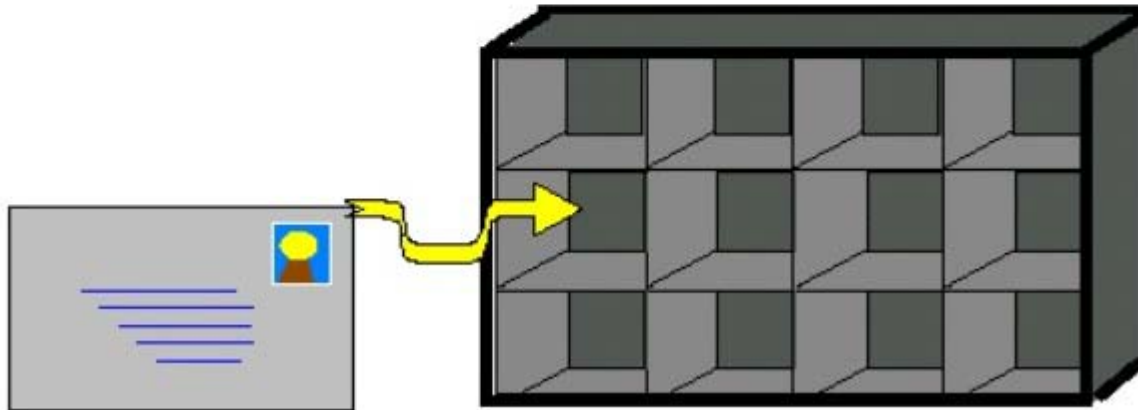
# MEMORY: ADDRESS AND VALUES

Every memory location has a **unique** address

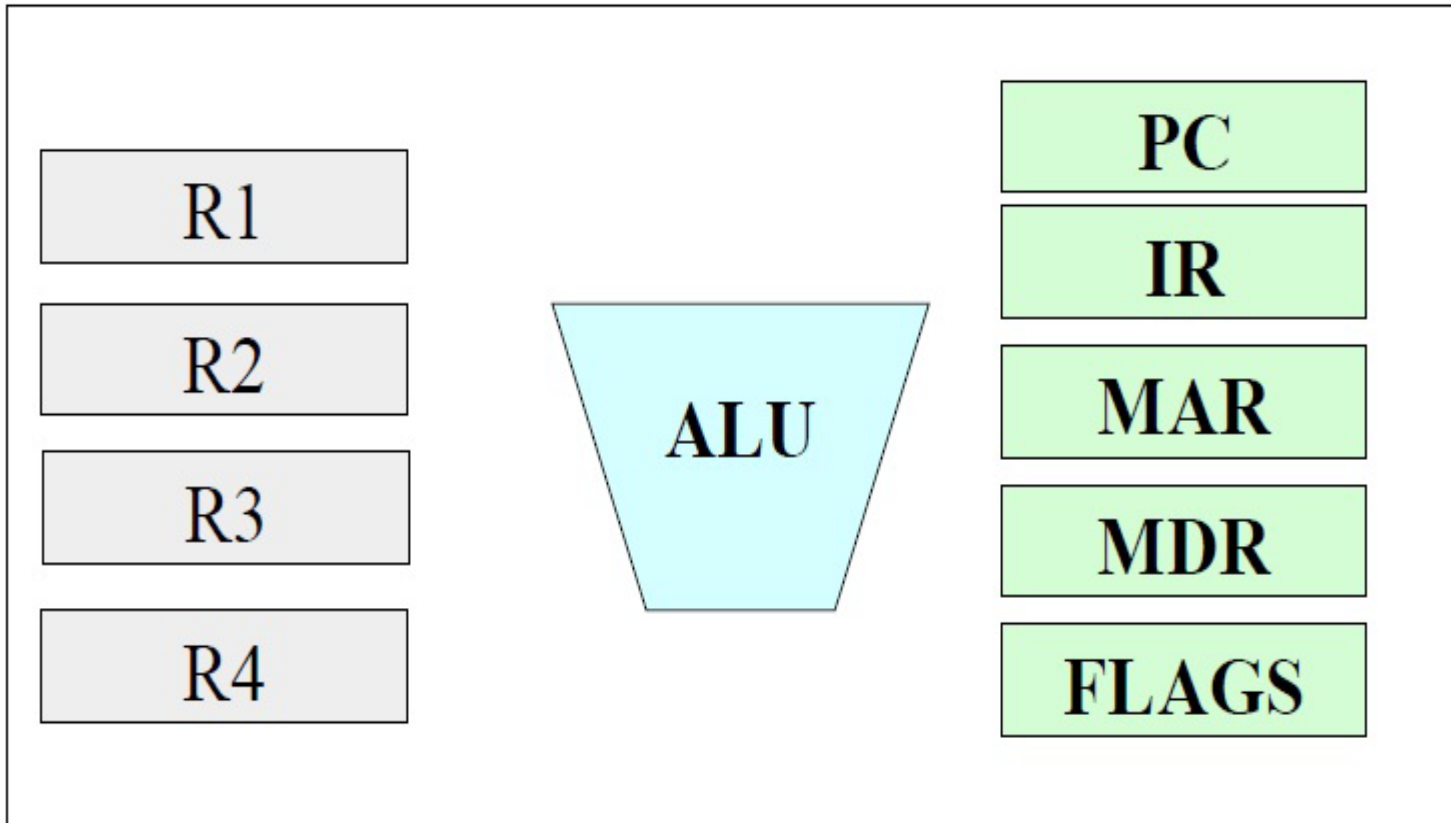
0	0
1	11
2	5
3	23
4	12
5	62

Address of byte

Value of byte (0...255)



# CPU: A FIRST CUT



# WHAT A COMPUTER CAN DO?

- Determining if a given number is a prime or not
- A Palindrome recognizer
- Read in airline route information as a matrix and determine the shortest time journey between two airports
- Missile Control
- Finger-print recognition
- Chess Player
- Speech Recognition
- Controlling robots
- Providing content privacy
- .....

# PROGRAMMING AND SOFTWARE

- Computer needs to be **programmed** to do such tasks
- **Programming** is the process of writing instructions in a language that can be understood by the computer so that a desired task can be performed by it
- **Program**: sequence of instructions to do a task, computer processes the instructions sequentially one after the other
- **Software**: A set of programs for doing some desired tasks on computers

- CPU understands machine language
  - Different strings of 0's and 1's only!!
  - Hard to remember and use
- Instruction set of a CPU
  - Mnemonic names for this strings

- ◆ **Start**
- ◆ **Read M**
- ◆ **Write M**
- ◆ **Load Data, M**
- ◆ **Copy M1, M2**
- ◆ **Add M1, M2, M3**
- ◆ **Sub M1, M2, M3**
- ◆ **Compare M1, M2, M3**
- ◆ **Jump L**
- ◆ **J\_Zero M, L**
- ◆ **Halt**

Instruction Set

0: Start  
1: Read 10  
2: Read 11  
3: Add 10, 11, 12  
4: Write 12  
5: Halt

Program

# PROBLEMS WITH PROGRAMMING USING INSTRUCTION SET DIRECTLY

- Instruction sets of different types of CPUs are different
  - Need to write different programs for computers with different types of CPUs even to do the same thing
- Still hard to remember
- Solution: High level languages (C, C++, Java,...)
  - CPU neutral, one program for many CPUs
  - Compiler to convert from high-level program to low level program that CPU understands

```
Variables x, y;  
Begin  
Read (x);  
Read (y);  
If (x > y) then Write (x)  
                else Write (y);  
End.
```

High Level Program

```
0: Start  
1: Read 20  
2: Read 21  
3: Compare 20, 21, 22  
4: J_Zero 22, 7  
5: Write 20  
6: Jump 8  
7: Write 21  
8: Halt
```

Low Level Program



# THREE STEPS IN WRITING A PROGRAM

- Step 1:
  - Write the program in a high-level language (in your case, C)
- Step 2:
  - Compile the program using a C compiler
- Step 3:
  - Run the program (to execute it)

# BINARY REPRESENTATION

- Numbers are represented inside computers in the base-2 system (Binary Numbers)
  - Only two symbols/digits 0 and 1
  - Positional weights of digits:  $2^0$ ,  $2^1$ ,  $2^2$ ,...from right to left for integers
- Decimal number system uses base-10
  - 10 digits, from 0 to 9, Positional weights  $10^0$ ,  $10^1$ ,  $10^2$ ,...from right to left for integers
  - Example:  $723 = 7 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

# BINARY NUMBERS

Dec	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

## Binary to Decimal Conversion

$$101011 \rightarrow 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 43$$

$$(101011)_2 = (43)_{10}$$

$$111001 \rightarrow 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 57$$

$$(111001)_2 = (57)_{10}$$

$$10100 \rightarrow 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 20$$

$$(10100)_2 = (20)_{10}$$

# BITS AND BYTES

- Bit – a single 1 or 0
- Byte – 8 consecutive bits
  - 2 bytes = 16 bits
  - 4 bytes = 32 bits
- Max. integer that can be represented
  - in 1 byte = 255 (=11111111)
  - in 4 bytes = 4294967295 (= 32 1's)
- No. of integers that can be represented in 1 byte = 256 (the integers 0, 1, 2, 3,...255)

- In the labs, you have to write C programs
  - using *gcc* compiler *under Linux/Unix* operating system
- So some basics of Linux/Unix will be presented next

# UNIX

- Unix is a multi-user, multi-tasking operating system.
- You can have many users logged into a system simultaneously, each running many programs.
- First Version was created in Bell Labs in 1969.
- Developed by Ken Thompson, Dennis Ritchie, Rudd Canaday, and Doug McIlroy designed and the name UNIX given by Brian Kernighan

# LINUX

- Linux is a free Unix-type operating system originally created by Linus Torvalds with the assistance of developers around the world
- It originated in 1991 as a personal project of Linus Torvalds, a Finnish graduate student.
- Developed under the GNU General Public License, the source code for Linux is freely available to everyone

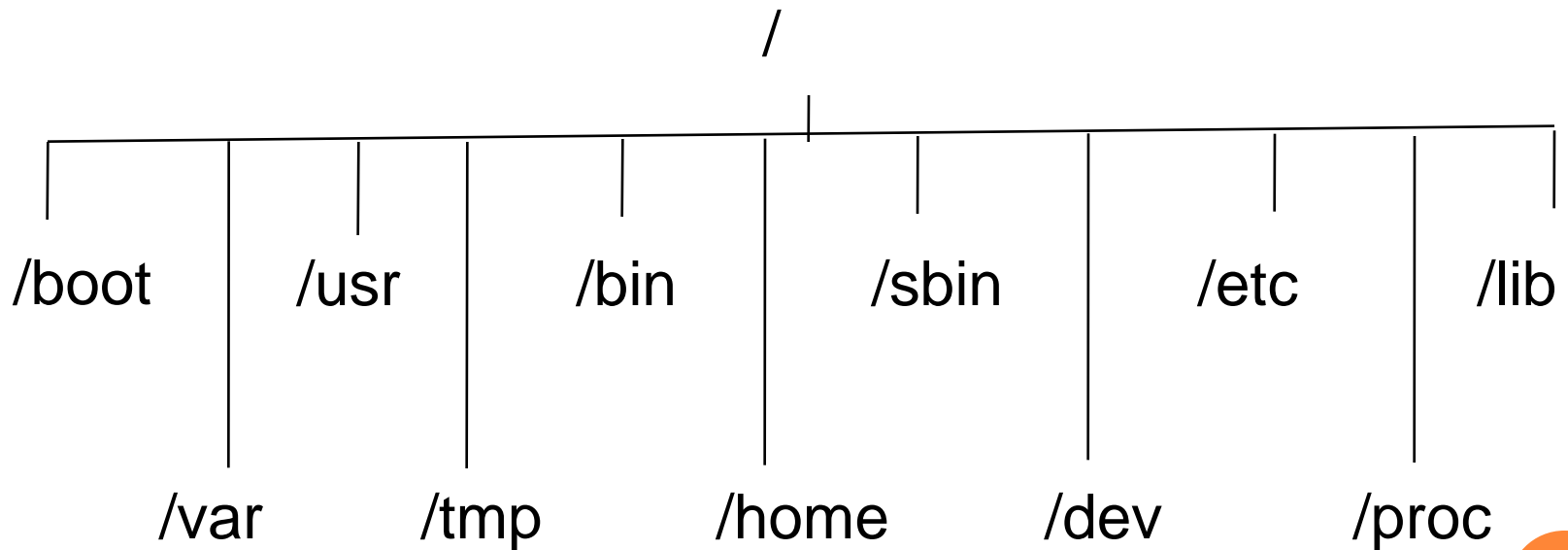
# LINUX BASICS

- Logging in /Authentication.
  - Two modes – text (sober but quicker) and graphical (requires more system resources).
- Remotely login to the server– **Telnet** (password in plain text); Securely connect using **SSH**.
- **Xmanager** is used to remotely login using graphical mode.



# USERS AND FILE STRUCTURE

- Two types of user – **super user** (root) and **user**
- Directory Structure



## KICKSTART COMMANDS

- Type these command in the command prompt followed by enter.
- **passwd** – Change password
- **exit** or **logout** – Leave the session
- **ls** – List files and directories
- **mkdir** <**name**> - Make Directory
- **cd** <**name**> - Change Directory
- **cd ..** -Go to the parent directory
- **rmdir** <**name**> - Remove Directory
- **cat** <**name**> - Display contents of a file

## KICKSTART COMMANDS (2)

- **file** <name> - Display file type
- **pwd** – Display present working directory
- **rm** <name> - Remove file
- **mv** <name> <new-name> - Move file
- **cp** <src> <dst> - Copy file

### Getting Help

- **man** <command> - Show manual
- **info** <command>; **whatis** <command>

# FILE EDITING USING VI EDITOR

- vi <name> - Text editor.
- Two modes – command and insert.
- Enter “i” for insert mode and type text.
- “Esc” for command mode
  - x : delete letter
  - dw: delete word
  - Use v and y to copy, v and d to cut
  - Use p to paste

## FILE EDITING (2)

- w: write/save file
- x: exit editor
- q: quit editor
- wq: write and quit
- w!: Forceful write
- q!: Quit without saving changes
- Moving the cursor
  - j/return/down-arrow; k/up-arrow
  - h/backspace/left-arrow; l/space/right-arrow

## FILE MANIPULATION COMMANDS

- Let there is a `hello.c` file only in my current directory
- Now type `ls -l`
- `-rwxr-xr-x 1 samrat fac 6664 Jan 3 12:11 hello.c`

Code	Meaning
0 or -	The access right that is supposed to be on this place is not granted.
4 or r	read access is granted to the user category defined in this place
2 or w	write permission is granted to the user category defined in this place
1 or x	execute permission is granted to the user category defined in this place

`chmod ??? <name>` - change file permissions

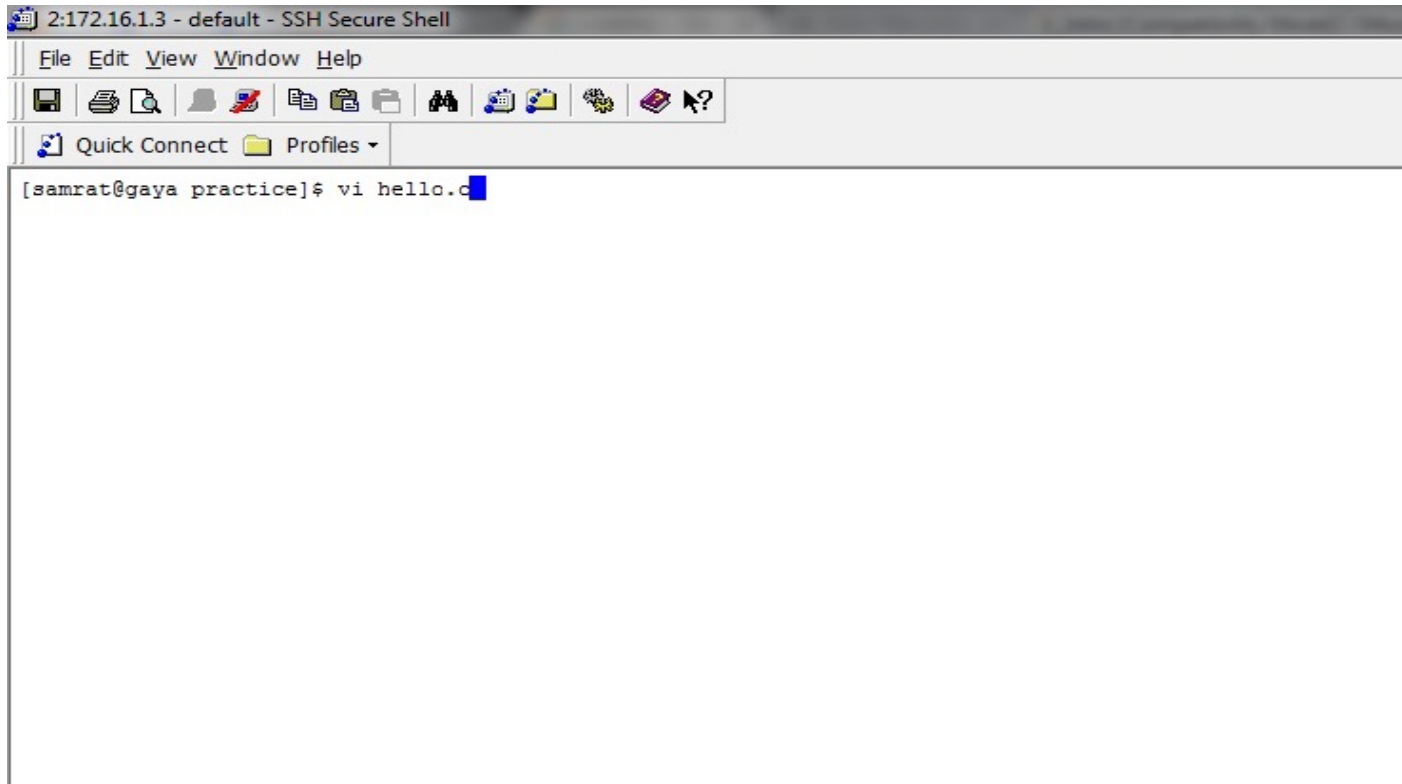


# HISTORY OF C

- C
  - Evolved by Ritchie from two previous programming languages, BCPL and B
  - Used to develop UNIX
  - Used to write modern operating systems
  - Hardware independent (portable)
  - By late 1970's C had evolved to "Traditional C"
- Standardization
  - Standard created in 1989, updated in 1999

# FIRST C PROGRAM

- Write a C program to display the message “Hello World!!!”
- In the command prompt type- `vi hello.c`

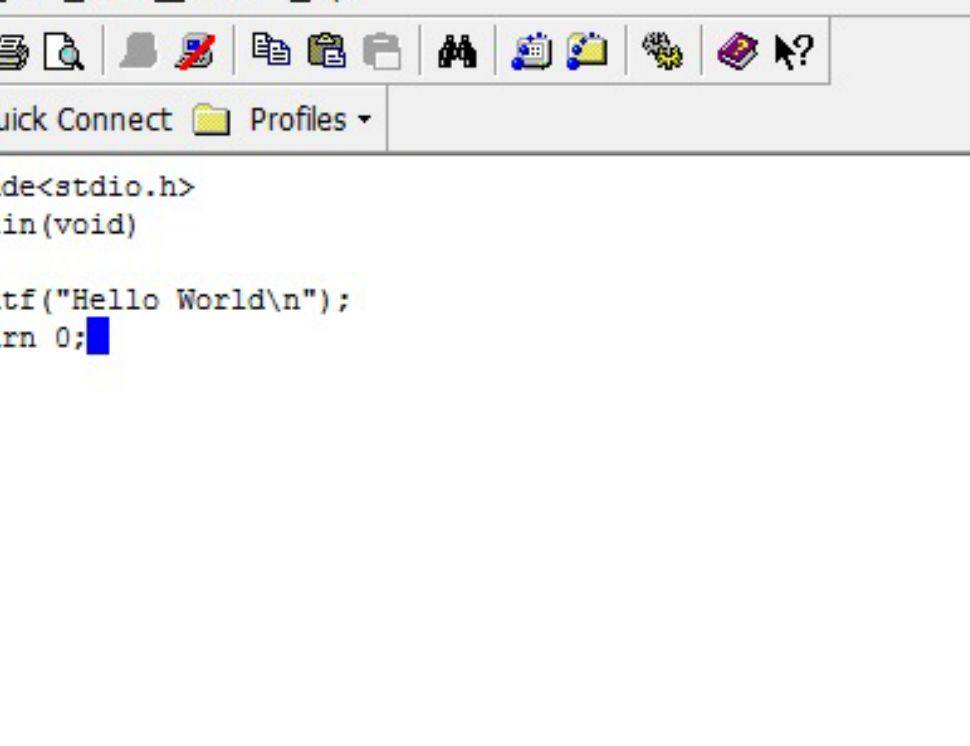


The screenshot shows a terminal window titled "2:172.16.1.3 - default - SSH Secure Shell". The window has a menu bar with "File", "Edit", "View", "Window", and "Help". Below the menu bar is a toolbar with various icons. The terminal content shows the prompt "[samrat@gaya practice]\$ vi hello.c" with a blue cursor at the end of the command.

Once you open the vi editor, press ‘i’ and you can start writing into it



# HELLO WORLD PROGRAM IN VI EDITOR



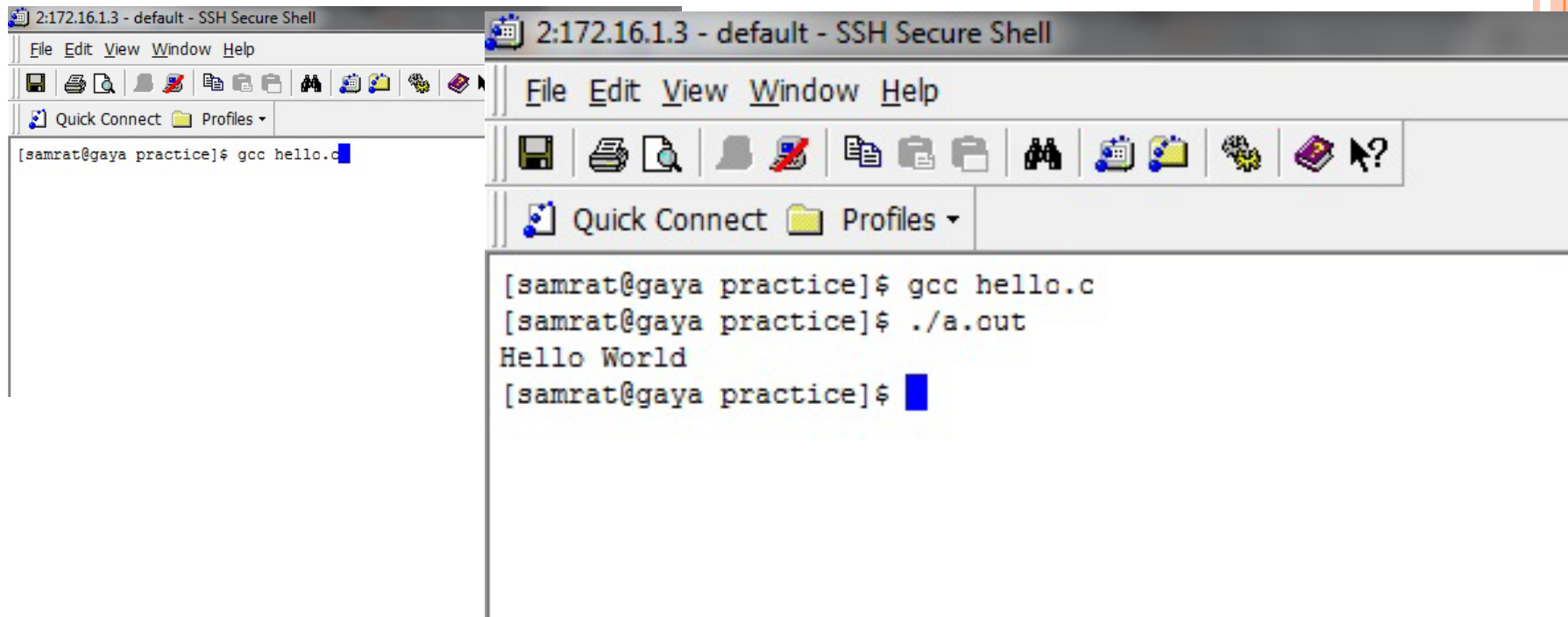
```
2:172.16.1.3 - default - SSH Secure Shell
File Edit View Window Help
[Icons: Save, Print, Find, Undo, Redo, Copy, Paste, Delete, Run, Stop, Break, Help, etc.]
Quick Connect Profiles
#include<stdio.h>
int main(void)
{
    printf("Hello World\n");
    return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
```

Once you finish writing into it then press 'esc' button and then ':' and 'wq'. It will save the contents and exit from the vi editor

# COMPILE AND RUN A C PROGRAM

*gcc* command is used

*./a.out* command is used



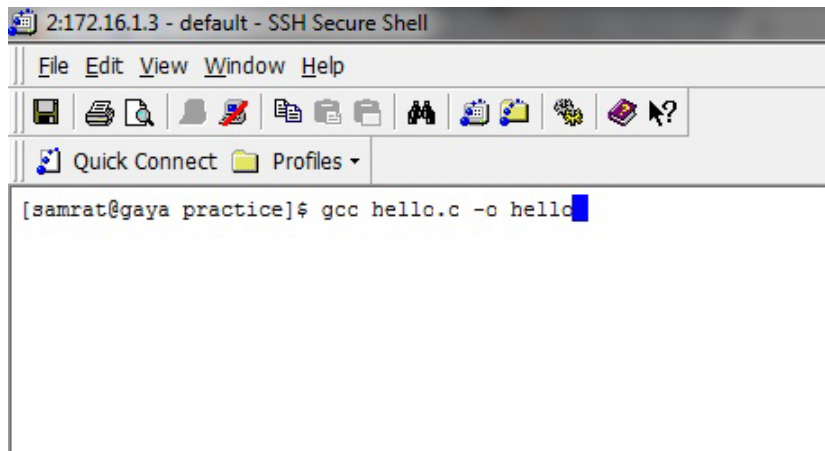
The image displays two screenshots of an SSH terminal window titled "2:172.16.1.3 - default - SSH Secure Shell". The left screenshot shows the command `gcc hello.c` being entered at the prompt `[samrat@gaya practice]$`. The right screenshot shows the same terminal after execution, displaying the output `Hello World` and the command `./a.out` being entered at the prompt `[samrat@gaya practice]$`.

```
[samrat@gaya practice]$ gcc hello.c
```

```
[samrat@gaya practice]$ gcc hello.c
[samrat@gaya practice]$ ./a.out
Hello World
[samrat@gaya practice]$
```

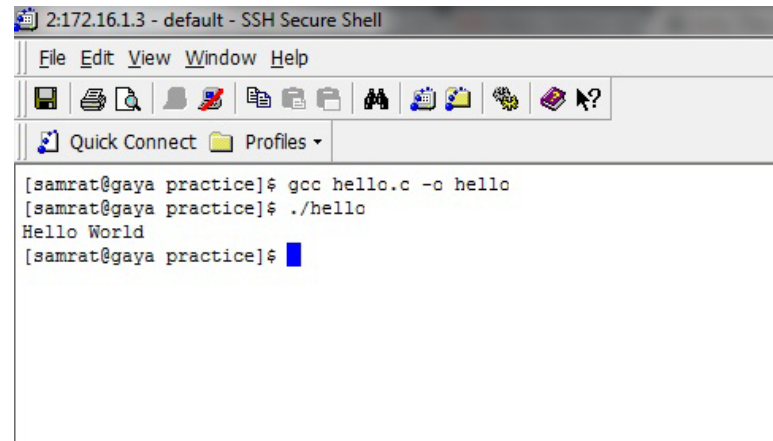
# COMPILE AND RUN A C PROGRAM- ALTERNATIVELY

*gcc* command is used  
with `-o` option

A terminal window titled '2:172.16.1.3 - default - SSH Secure Shell' with a menu bar (File, Edit, View, Window, Help) and a toolbar. The command prompt shows the user 'samrat' at host 'gaya' in the 'practice' directory. The command `gcc hello.c -o hello` has been entered and executed, with a blue cursor at the end of the line.

```
2:172.16.1.3 - default - SSH Secure Shell
File Edit View Window Help
[samrat@gaya practice]$ gcc hello.c -o hello
```

*Directly use the file name  
used* with `-o` option

A terminal window titled '2:172.16.1.3 - default - SSH Secure Shell' with a menu bar (File, Edit, View, Window, Help) and a toolbar. The command prompt shows the user 'samrat' at host 'gaya' in the 'practice' directory. The commands `gcc hello.c -o hello` and `./hello` have been entered and executed. The output 'Hello World' is displayed. A blue cursor is at the end of the final command line.

```
2:172.16.1.3 - default - SSH Secure Shell
File Edit View Window Help
[samrat@gaya practice]$ gcc hello.c -o hello
[samrat@gaya practice]$ ./hello
Hello World
[samrat@gaya practice]$
```

For more options- `gcc --help`

# REVISIT “HELLO WORLD” PROGRAM

```
1. #include<stdio.h>
2. int main(void)
3. {
4.     printf(“Hello World\n”);
5.     return 0;
6. }
```

# REVISIT “HELLO WORLD” PROGRAM

- The first line is `#include<stdio.h>`
  - `#` is a pre-processing directive
  - `#include` tells the pre-processor to include the header file *stdio.h* into the program
  - The angle brackets tell the preprocessor that the header file *stdio.h* will be available in the standard directory where all header files are available.

# PROGRAM “HELLO WORLD”

```
1. #include<stdio.h>
2. int main(void)
3. {
4.     printf(“Hello World\n”);
5.     return 0;
6. }
```

## REVISIT “HELLO WORLD” PROGRAM(2)

- The second line is `int main(void)`
  - `main()` is the name of a function. Execution of all C programs start with the function `main`.
  - The word `int` and `void` are keywords.
  - `int` tells the compiler that this function returns an integer value.
  - `void` tells the compiler that this function does not take any argument.

# PROGRAM “HELLO WORLD”

```
1. #include<stdio.h>
2. int main(void)
3. {
4.     printf(“Hello World\n”);
5.     return 0;
6. }
```



## REVISIT “HELLO WORLD” PROGRAM(3)

- The third and last line are braces `{ }`
  - braces are used to surround the body of a function
  - braces are used to group statements together.
  - The right and left braces should match.

# PROGRAM “HELLO WORLD”

```
1. #include<stdio.h>
2. int main(void)
3. {
4.     printf(“Hello World\n”);
5.     return 0;
6. }
```

## REVISIT “HELLO WORLD” PROGRAM(4)

- The fourth line is `printf(“Hello World\n”)`
  - *printf* is a standard library function. Information about *printf* is included in the header file *stdio.h*
  - The string Hello World is an argument to printf.
  - ‘\n’ represents a single character called **newline**. It is used to move the cursor to the next line

# PROGRAM “HELLO WORLD”

```
1. #include<stdio.h>
2. int main(void)
3. {
4.     printf(“Hello World\n”);
5.     return 0;
6. }
```

## REVISIT “HELLO WORLD” PROGRAM(5)

- The fifth line is `return 0`
  - It causes the integer value 0 to be returned to the operating system.
  - The returned value may or may not be used.
  - The return value is sometimes used to keep the *compiler happy*.

# MORE PRINT

- `#include <stdio.h>`
- `int main()`
- `{`
- `printf ("Hello, World! ") ;`
- `printf ("Hello \n World! \n") ;`
- `return 0;`
- `}`

## SOME MORE PRINT

- `#include <stdio.h>`
- `int main()`
- `{`
- `printf ("Hello, World! \n") ;`
- `printf ("Hello \n World! \n") ;`
- `printf ("Hell\nno \t World! \n") ;`
- `return 0;`
- `}`

# READING VALUES FROM KEYBOARD

- `#include <stdio.h>`
- `int main()`
- `{`
- `int num ;`
- `scanf ("%d", &num) ;`
- `printf ("No. of students is %d\n", num) ;`
- `return 0;`
- `}`



# CENTIGRADE TO FAHRENHEIT

- `#include <stdio.h>`  $(fahr - 32)/9 = cent/5$
- `int main()`
- `{`
- `float cent, fahr;`
- `scanf("%f",&cent);`
- `fahr = cent*(9.0/5.0) + 32;`
- `printf( "%f C equals %f F\n", cent, fahr);`
- `return 0;`
- `}`

# LARGEST OF TWO NUMBERS

- `#include <stdio.h>`
- `int main()`
- `{`
- `int x, y;`
- `scanf("%d%d",&x,&y);`
- `if (x>y)`
- `printf("Largest is %d\n",x);`
- `else`
- `printf("Largest is %d\n",y);`
- `return 0;`
- `}`

## CONVERT MILES, YARDS TO KMS.

```
1. #include <stdio.h>
2. /* Convert miles, yards to kms */
3. int main(void)
4. {   int miles, yards; float kms;
5.     miles = 26; yards = 385;
6.     kms = 1.609 * (miles + yards/1760.0);
7.     printf("\nThe distance in kms is
           %f.\n\n", kms);
8.     return 0; }
```

## CONVERT MILES, YARDS TO KMS.

```
1. #include <stdio.h>
2. /* Convert miles, yards to kms */
3. int main(void)
4. {   int miles, yards; float kms;
5.     miles = 26; yards = 385;
6.     kms = 1.609 * (miles + yards/1760.0);
7.     printf("\nThe distance in kms is
           %f.\n\n", kms);
8.     return 0; }
```

# COMMENTS

- Anything written between `/* ... */` is a comment and is ignored by the compiler
- Lines starting with `//` are also comments and ignored by the compiler.

## CONVERT MILES, YARDS TO KMS.

```
1. #include <stdio.h>
2. /* Convert miles, yards to kms */
3. int main(void)
4. {   int miles, yards; float kms;
5.     miles = 26; yards = 385;
6.     kms = 1.609 * (miles + yards/1760.0);
7.     printf("\nThe distance in kms is
           %f.\n\n", kms);
8.     return 0; }
```

# VARIABLES & DECLARATION STMT.

- Line 4: `int miles, yards;`
  - is a declaration statement
  - *miles* and *yards* are called variables. The keyword *int* tells the compiler that these variables are of type integer and take on integer values.
- Line 4 `float kms;`
  - is again a declarative statement
  - Tells the compiler that variable *kms* is of type floating point.

## CONVERT MILES, YARDS TO KMS.

```
1. #include <stdio.h>
2. /* Convert miles, yards to kms */
3. int main(void)
4. {   int miles, yards; float kms;
5.     miles = 26; yards = 385;
6.     kms = 1.609 * (miles + yards/1760.0);
7.     printf("\nThe distance in kms is
           %f.\n\n", kms);
8.     return 0; }
```



# ASSIGNMENT STATEMENTS

- Line 5: miles = 26; yards = 385;
  - are assignment statements
  - = is an assignment operator. Assigns the value 26 to variable miles and 385 to variable yards.

## PROGRAMMING TIP (2):

- Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.



## CONVERT MILES, YARDS TO KMS.

```
1. #include <stdio.h>
2. /* Convert miles, yards to kms */
3. int main(void)
4. {   int miles, yards; float kms;
5.     miles = 26; yards = 385;
6.     kms = 1.609 * (miles + yards/1760.0);
7.     printf("\nThe distance in kms is
8.         %f.\n\n", kms);
9.     return 0; }
```

# EXPRESSIONS

- The sixth line:

`kms = 1.609 * (miles + yards/1760.0);`

- is also an assignment statement.
- the value of the expression on the right hand side is computed and assigned to the floating type variable *kms*.

## CONVERT MILES, YARDS TO KMS.

```
1. #include <stdio.h>
2. /* Convert miles, yards to kms */
3. int main(void)
4. {   int miles, yards; float kms;
5.     miles = 26; yards = 385;
6.     kms = 1.609 * (miles + yards/1760.0);
7.     printf("\nThe distance in kms is
   %f.\n\n", kms);
8.     return 0; }
```

# THE PRINTF() FUNCTION

- The seventh line: `printf("\nThe distance in kms is %f.\n\n", kms);`
  - `printf()` can take variable number of arguments
  - The control string `%f` is matched with the variable `kms`.
  - It will print the variable `kms` as a floating-point number and insert into the stream where the format `%f` occurs.

# OUTPUT

- `printf("<control string>", other argument);`

Control String	How printed
c	as a character
d	as a decimal integer
e	as a floating-point in scientific notation
f	as a floating-point number
g	in e or f format, whichever shorter
s	as string

# INPUT

- Eg: `scanf( "%d", &integer1 );`
  - Obtains a value from the user
  - `scanf` uses standard input (usually keyboard)
  - `%d` - indicates data should be a decimal integer
  - `&integer1` - location in memory to store variable



# THE C CHARACTER SET

- The C language alphabet
  - Uppercase letters 'A' to 'Z'
  - Lowercase letters 'a' to 'z'
  - Digits '0' to '9'
  - Certain special characters:

!	#	%	^	&	*	(	)
-	_	+	=	~	[	]	\
	;	:	'	"	{	}	,
.	<	>	/	?	blank		

# STRUCTURE OF A C PROGRAM

- A collection of functions (we will see what they are later)
- Exactly one special function named `main` must be present. Program always starts from there
- Each function has statements (instructions) for declaration, assignment, condition check, looping etc.
- Statements are executed one by one

# VARIABLES

- Very important concept for programming
- An entity that has a value and is known to the program by a name
- Can store any temporary result while executing a program
- Can have only one value assigned to it at any given time during the execution of the program
- The value of a variable can be changed during the execution of the program

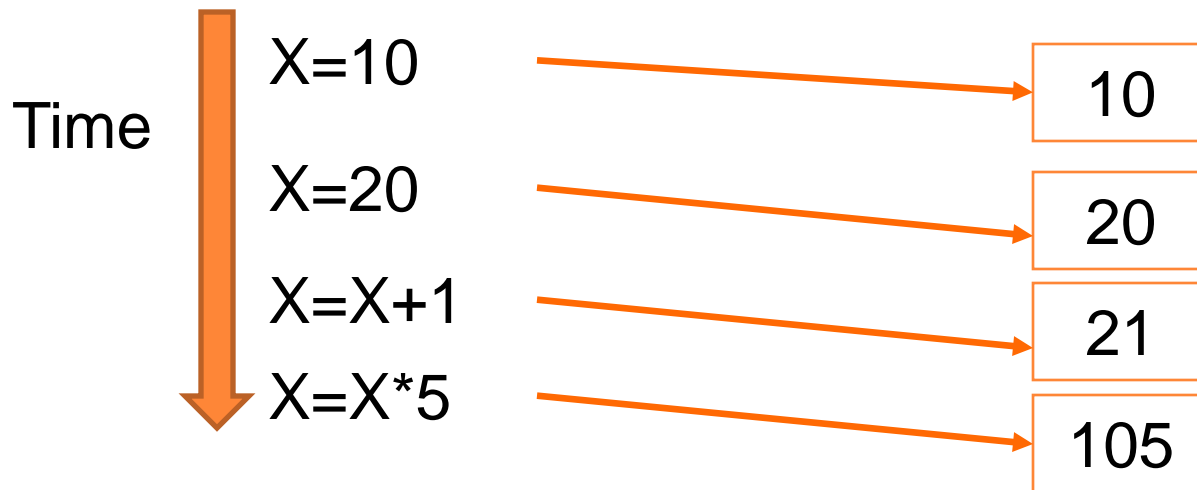
## CONTD.

- Variables are stored in memory
- Remember that memory is a list of storage locations, each having a unique address
- A variable is like a **bin**
  - The contents of the bin is the value of the variable
  - The variable name is used to refer to the value of the variable
  - A variable is mapped to a location of the memory, called its address

- `#include <stdio.h>`
- `int main( )`
- `{`
- `int x;`
- `int y;`
- `x=1;`
- `y=3;`
- `printf("x = %d, y= %d\n", x, y);`
- `return 0;`
- `}`

# VARIABLES IN MEMORY

Instruction Executed      Memory location  
allocated to a variable X



# DATA TYPES

- Each variable has a type, indicates what type of values the variable can hold
- Four common data types in C
  - `char` - can store a character (1 byte)
  - `int` - can store integers (usually 2/4 bytes)
  - `float` - can store single-precision floating point numbers (usually 4 bytes)
  - `double` - can store double-precision floating point numbers (usually 8 bytes)

## CONTD.

- Must declare a variable (specify its type and name) before using it anywhere in your program
- All variable declarations should be at the beginning of the main() or other functions
- A value can also be assigned to a variable at the time the variable is declared.
  - `int speed = 30;`
  - `char flag = 'y';`



# VARIABLE NAMES

- Sequence of letters and digits
- First character must be a letter or ‘\_’
- No special characters other than ‘\_’
- No blank in between
- Names are case-sensitive (max and Max are two different names)
- Examples of valid names:
  - i, rank1, MAX, max, Min, class\_rank
- Examples of invalid names:
  - a's, fact rec, 2sqroot, class,rank

# MORE VALID AND INVALID IDENTIFIER

## ○ Valid identifiers

- **X**
- **abc**
- **Simple\_interest**
- **a123**
- **LIST**
- **stud\_name**
- **Empl\_1**
- **Empl\_2**
- **Avg\_empl\_salary**

## ○ Invalid identifiers

- **10abc**
- **my-name**
- **“hello”**
- **simple interest**
- **(area)**
- **%rate**

# C KEYWORDS

- Used by the C language, cannot be used as variable names
- Examples:
  - int, float, char, double, main, if else, for, while. do, struct, union, typedef, enum, void, return, signed, unsigned, case, break, sizeof,....
  - There are others, see textbook...

## EXAMPLE 1

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int x, y, sum;
```

```
    scanf("%d%d",&x,&y);
```

```
    sum = x + y;
```

```
    printf( "%d plus %d is %d\n", x, y, sum );
```

```
    return 0;
```

```
}
```

**Three int type variables declared**



**Values assigned**



## EXAMPLE - 2

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    float x, y;
```

```
    int d1, d2 = 10;
```

```
    scanf("%f%f%d",&x, &y, &d1);
```

```
    printf( "%f plus %f is %f\n", x, y, x+y);
```

```
    printf( "%d minus %d is %d\n", d1, d2, d1-d2);
```

```
    return 0;
```

```
}
```

**Assigns an initial value to d2,  
can be changed later**



## READ-ONLY VARIABLES

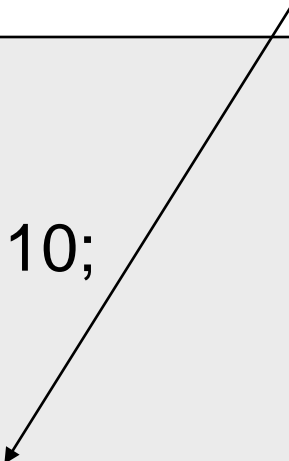
- Variables whose values can be initialized during declaration, but cannot be changed after that
- Declared by putting the `const` keyword in front of the declaration
- Storage allocated just like any variable
- Used for variables whose values need not be changed
  - Prevents accidental change of the value

## Correct

```
int main()
{
    const int LIMIT = 10;
    int n;
    scanf("%d", &n);
    if (n > LIMIT)
        printf("Out of
        limit");
    return 0;
}
```

Incorrect: **Limit changed**

```
int main()
{
    const int Limit = 10;
    int n;
    scanf("%d", &n);
    Limit = Limit + n;
    printf("New limit is %d", Limit);
    return 0;
}
```



# CONSTANTS

## ○ Integer constants

- Consists of a sequence of digits, with possibly a plus or a minus sign before it
- Embedded spaces, commas and non-digit characters are not permitted between digits

## ○ Floating point constants

## ○ Two different notations:

- Decimal notation: 25.0, 0.0034, .84, -2.234
- Exponential (scientific) notation: 2.500000e+01, 3.400000e-03, 8.400000e-01, -2.234000e+00

**e means “10 to the power of”**



## CONTD.

### ○ Character constants

- Contains a single character enclosed within a pair of single quote marks.
- Examples :: '2', '+', 'Z'

### ○ Some special backslash characters

'\n'	new line
'\t'	horizontal tab
'\"'	single quote
'\"'	double quote
'\\'	backslash
'\0'	null

## INPUT: **SCANF** FUNCTION

- Performs input from keyboard
- It requires a *format string* and a *list of variables* into which the value received from the keyboard will be stored
- format string = individual groups of characters (usually ‘%’ sign, followed by a conversion character), with one character group for each variable in the list

```
int a, b;
```

```
float c;
```

```
scanf("%d %d %f", &a, &b, &c);
```

**Variable list (note the &  
before a variable name)**

**Format string**

- Commonly used conversion characters
  - c** for char type variable
  - d** for int type variable
  - f** for float type variable
  - lf** for double type variable

- Examples

```
scanf ("%d", &size) ;  
scanf ("%c", &nextchar) ;  
scanf ("%f", &length) ;  
scanf ("%f%lf", &a, &b);
```

## READING A SINGLE CHARACTER

- A single character can be read using `scanf` with `%c`
- It can also be read using the `getchar()` function  
char c;  
c = getchar();
- Program waits at the `getchar()` line until a character is typed, and then reads it and stores it in c
- The function `putchar(c)` prints a character each time it is called

## OUTPUT: `PRINTF` FUNCTION

- Performs output to the standard output device (typically defined to be the screen)
- It requires a format string in which we can specify:

- The text to be printed out
- Specifications on how to print the values

`printf ("The number is %d\n", num);`

- The format specification `%d` causes the value listed after the format string to be embedded in the output as a decimal number in place of `%d`
- Output will appear as: `The number is 125`

## CONTD.

- General syntax:

`printf (format string, arg1, arg2, ..., argn);`

- format string refers to a string containing formatting information and data types of the arguments to be output
- the arguments arg1, arg2, ... represent list of variables/expressions whose values are to be printed
- The conversion characters are the same as in scanf

At least 3 characters wide

- Examples:

```
printf ("Average of %d and %d is %f", a, b, avg);
```

```
printf ("Hello \nGood \nMorning \n");
```

```
printf ("%3d %3d %5d", a, b, a*b+2);
```

```
printf ("%7.2f %5.1f", x, y);
```

- Many more options are available for both printf and scanf


- Read from the book
- Practice them in the lab

At least 7 characters wide  
and 2 after decimal point

## EXAMPLE: CHARACTER COUNTING

- `#include<stdio.h>`
- `int main()`
- `{`
- `int nc = 0;`
- `while (getchar() != EOF)`
- `nc = nc + 1;`
- `printf(“%d\n”,nc);`
- `return 0;`
- `}`

EOF: (end of file) a value (ctrl-d) that is different from all other characters to mark the end of input





# SYNTAX AND SEMANTICS

- Syntax refers to the grammar structure and semantics to the meaning
- A program may be syntactically correct but semantically wrong.
- Compiler checks for syntax errors.

# COMPILATION PROCESS

