

MA 511: Computer Programming

Lecture 1: Introduction

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics
IIT Guwahati

Time Table

	11:00-11:55 AM		3:00-4:55 PM
Monday			
Tuesday			AL2-Slot: Lab
Wednesday	E-Slot: Lecture		
Thursday	E-Slot: Lab/Lecture		
Friday	F-Slot: Lecture		
Saturday			

Teaching Assistant (TA): Mr. Subhajit Pramanick
 Mr. Saswata Jana

Tests and Marks distribution

- Marks distribution:
 - Quizzes (30%) + (Performance in Lab + Viva) (30%) + Midsem (20%) + Endsem (20%)
- There may be **pop quizzes** (without prior warning or announcement) in the labs/classes.
- If any one fails to write any of the **Tests** then **NO make-up examination will be conducted**.
- The portion of syllabus for a Test will be in general from the first lecture to the last lecture just before the Test.
- Below 30 will be awarded as **F grade**.



Policy of Attendance

- Attendance in all **lecture** and **laboratory** classes is **compulsory**.
- Students, who do not meet minimum (75% as per the policy of the institute) attendance requirement will not be allowed to write the end semester examination.
- You must be **punctual**.

Cheating Gets You “equivalent -ve marks of the Question”

Specific Examples Of Cheating:

- If you **help** somebody for **cheating** or providing your login **passwd**.
- If you're going to copy someone's **assignment** or **copy code** during test.
- If provide your code through e-mail/memory devises during lab class/test.

Reference/Text Books

- **A Book on C** by A. L. Kelley & I. Pohl
- **Programming with C** by Byron S Gottfried
- **The C Programming Language** by Brian W. Kernighan & Dennis M. Ritchie
- **Computer Programming in C** by V. Rajaraman

Computer

- **What is a computer?**

A simple definition: A computer is a machine that computes. It stores data (words, numbers, or pictures), interacts with external devices (the monitor, printer, speakers), and executes programs.

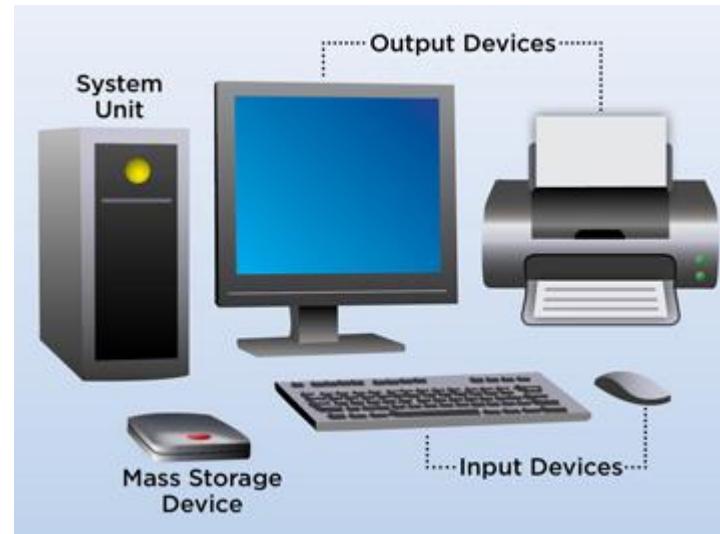
- **What is a computer system?**

A complete computer system consists more than just a computer. The four main components are:

- Hardware - the physical computer
- Operating System - software that allows other pieces of software and human users to interact with the hardware
- Application Programs - specialized software like Word, Excel, Firefox,
- End User - human user or another application program

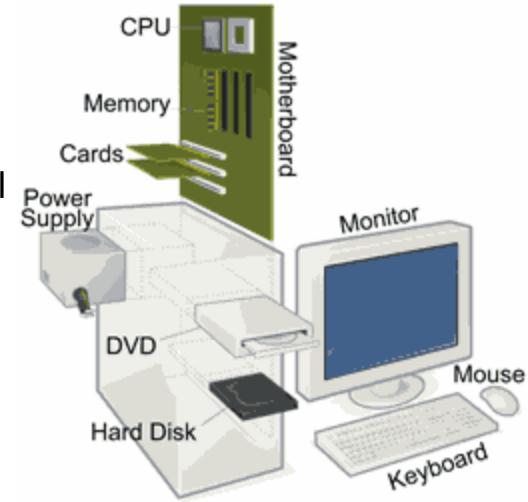
Computer Organization and Architecture

- CPU - central processing unit
 - Where decisions are made, computations are performed, and input/output requests are delegated
- Memory
 - Stores information being processed by the CPU
- Input devices
 - Allows user to supply information to computers
- Output devices
 - Allows user to receive information from computers



CPU

- Central processing unit (CPU) is the heart of the computer
 - Control Unit (CU) fetches, decodes and executes instructions
 - Arithmetic calculations are performed using the Arithmetic/Logical Unit or ALU
 - Register stores processor status and small data
- Arithmetic operations are performed using binary number system
- CPU executes instructions which are stored in memory
- It understands only machine language that is expressed in binary.
- The CPU is mounted on a motherboard. The motherboard has slots for the RAM, and other card slots for the video, network, and sound cards.
- The CPU, RAM, the hard disk, and other devices are connected through a set of electrical lines called a bus.



Memory

- All data stored in memory are encoded as some unique combination of 0 and 1 called *bits (binary digits)*
- 8-bits = one bytes
 - A single character will occupy one byte of memory
 - A single numeric quantity may occupy 1 to 8 bytes, depending on *precision*.
 - Here are some commonly used unites.
 - 1 KiloByte (KB) = 2^{10} Bytes ($\sim 10^3$ Bytes)
 - 1 MegaByte (MB) = 2^{10} KiloBytes ($\sim 10^3$ KB)
 - 1 GigaByte (GB) = 2^{10} MegaBytes ($\sim 10^3$ MB)
 - 1 TeraByte (TB) = 2^{10} GigaBytes ($\sim 10^3$ GB)
 - 1 PetaByte (PB) = 2^{10} TeraBytes ($\sim 10^3$ TB)

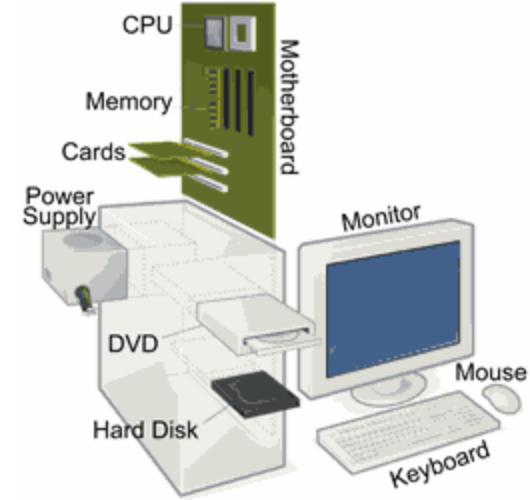
Memory

Primary or Main Memory

- Random Access Memory, or RAM)
 - fast, but **volatile** (i.e. contents lost on power off)
 - access time ~ 100 nano sec ($\text{nano} = 10^{-9}$)
Random access: to any specified part of memory
 - typical capacity today: 512 MB to a few GB
- Read Only Memory, or ROM)
 - fast, but **non-volatile**, used for booting

Secondary Memory

- Comparatively slow, non-volatile
 - Hard Disc, Flash Drive, DVD, CD, Pen drive, External Disc



Speed and Reliability

- Adding two numbers, is usually expressed in terms of microseconds, ($1 \mu\text{sec} = 10^{-6} \text{ sec}$) or nanoseconds ($1 \text{nsec} = 10^{-9} \mu\text{sec} = 10^{-9} \text{ sec}$).
- If a computer can add two numbers in 10 nanosec , 100 million (10^8) additions will be carried out in one sec.
- This high speed is also equally guarantee high reliability. Computer never make mistake unless programming errors and data entry errors occur.

Computer Speed

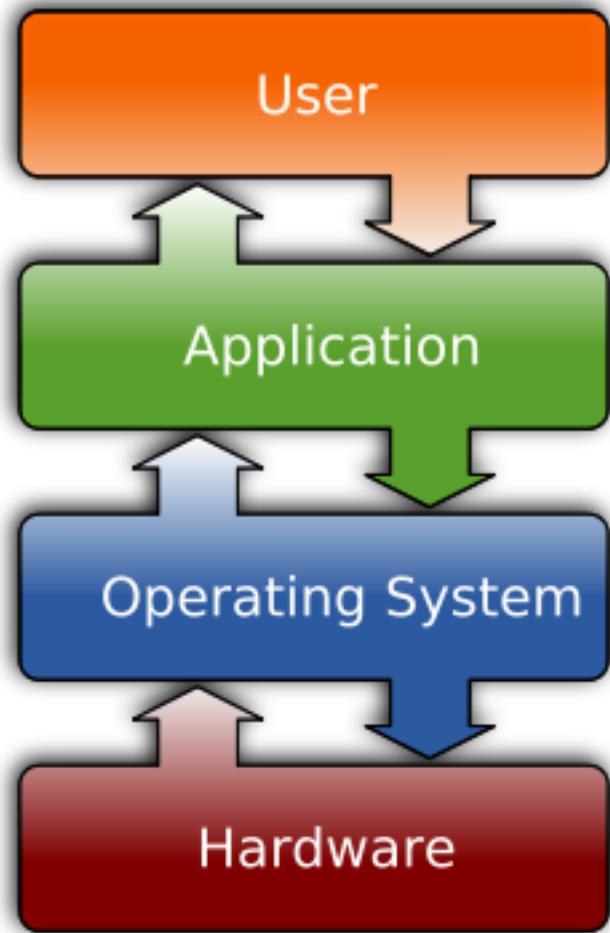
- Clock speed is a measure of how quickly a computer completes basic computations and operations.
- It is measured as a frequency in **hertz (Hz)**, and most commonly refers to the speed of the computer's CPU.
- Since the frequency most clock speed measures is very high, the terms **megahertz (MHz)** and **gigahertz (GHz)** are used.
- **1 Hz means that an event repeats once per second.**
- A **MHz** is one-million cycles per second, while a **GHz** is one-billion cycles per second.
- So a computer with a clock speed of 800MHz is running 800,000,000 cycles per second.
- while a 2.4GHz computer is running 2,400,000,000 cycles per second.

Hardware vs Software

- Physical components are called hardware; CPU, GPU, RAM, ROM, keyboard, mouse, monitor, etc.
- Software (programs) are computer instructions - with help of hardware its capable to perform a desire task.
 - System Software ; Operating System (OS)
 - Application Software
- Computer hardware and software require each other and neither can be realistically used on its own.

Operating System

- It's a system software that help the users to operate the computer and manage its resources.
- Its an interface between the computer and the user.
Ex: MS-DOS, Windows, Linux, MAC, etc.



What is Program?

- *A program is a set of instructions that a computer executes to achieve a certain desired effect - perform a calculation, render a picture, or produce music.*
- A program is written in a specific programming language.
- A computer cannot solve a problem by itself. The programmer must write down in minute details each step the computer has to take to solve the problem.
- An algorithm is the set of steps taken to solve a given problem.

MA 511: Computer Programming

Lecture 2

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics
IIT Guwahati

What is Program?

- *A program is a set of instructions that a computer executes to achieve a certain desired effect - perform a calculation, render a picture, or produce music.*
- A program is written in a specific programming language.
- A computer cannot solve a problem by itself. The programmer must write down in minute details each step the computer has to take to solve the problem.
- An algorithm is the set of steps taken to solve a given problem.

What is an algorithm?

- An algorithm is a step-by-step solution to a given problem. An algorithm has the following properties:
 - finiteness - the process terminates, the number of steps are finite
 - definiteness - each step is precisely stated
 - effective computability - each step can be carried out by a computer

How and why programming?

- **How does a computer run a program?**
 - The program is first loaded into RAM. The computer reads the program one instruction at a time. The CPU decodes the instruction, reads data, executes the instruction and stores results (if any). The CPU may also interact with the monitor, printer, speakers or other peripheral devices as instructed.
- **Why everyone should learn computer programming?**
 - Computers are ubiquitous. Programming is a basic skill like reading, writing, and arithmetic.
 - Programming is a discipline that teaches the science and art of solving problems.
 - Programming is different from using an application package like Microsoft Word or Excel. It allows you to blow the power of the computer for your own use.

Type of programming languages

- Lower level language }
 - Machine language
- High level language
 - Example: Fortran, Pascal, Visual Basic, **C**, C++, }
Java, Python, etc.

Machine language instructions

A computer can interpret and execute a set of coded instructions called machine language instructions.

Operation code memory location

- | | |
|------------|----------|
| 1. 0110 | 10001110 |
| 2. 0111 | 10001111 |
| 3. 1000 | 01110001 |

- Load (0110) from memory location 10001110 to CPU register
- Add (0111) the contents of 10001111 to the value of the register
- Result which is in register is to be copied/store (1000) into location 01110001 of the memory.

Problems in machine language coding

- Very cumbersome to work
 - More than 100 different machine instruction codes and hundreds of thousands of locations in memory.
- Different type of computer has its own unique instruction set
 - Operation codes are differ from one machine model to another
- One machine language program written for one type of computer cannot be run on another type of computer without significant alterations.
- Rewrite the program for different machines.

Computer program should be written in a **high level programming languages** which is independent of machine language.

High Level Language

- A single instruction in High Level Language is equivalent to several instructions machine language. }
- Simplicity
 - Instruction set is more compatible with human language.
- Uniformity and portability
 - A program written for one computer can generally be run on many different computer with a little or no alteration.
- General purpose language
 - C, Pascal, Fortran and BASIC.
- Special purpose language
 - LISP: List processing language, is widely used for AI. [LISP]
 - CSMP, SIMAN: simulation language

Computer Characteristics

- Computer are used to transmit, store and manipulate information i.e., data
- Data type:
 - Numeric data
 - Character data
 - Graphic data
 - Sound
- To process a particular set of data, the computer must be given an appropriate set of instructions called a program.

Compilation or Interpretation

✓ Compiler:

- Translate entire program into machine language before executing any instructions.

✓ Interpreter:

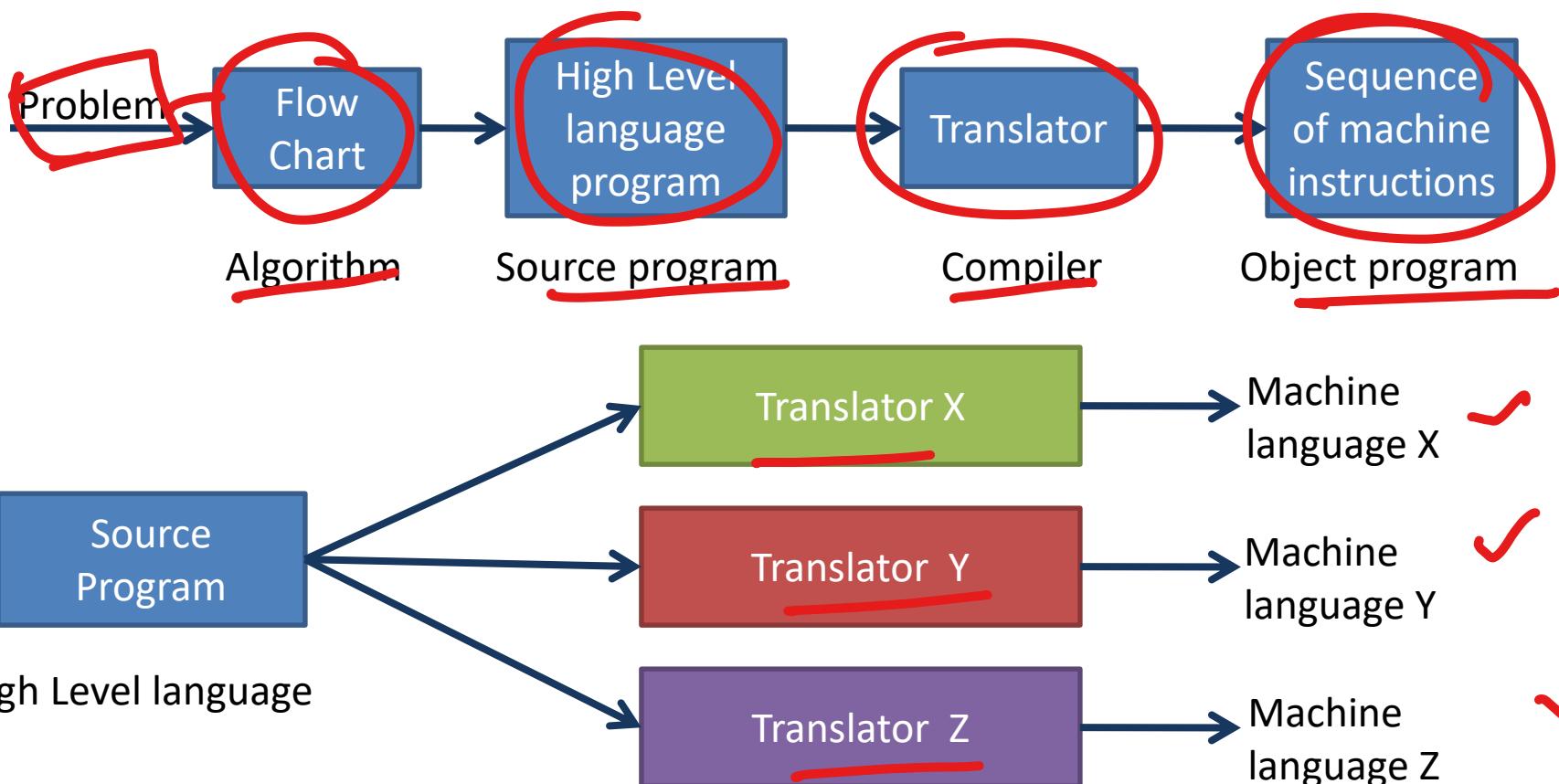
- process through a program by translating and executing single instructions or small group instructions.

Complier/interpreter

- **Complier/interpreter** is itself a computer program. It accept a program in a high level language like C as input, and generates a corresponding machine language program as output.
- The high level program is called source program
- The resulting machine language program is called the object program.
- Every computer mush have its own compiler or interpreter for a particular high level language.



Computer Language



Computer Algorithms

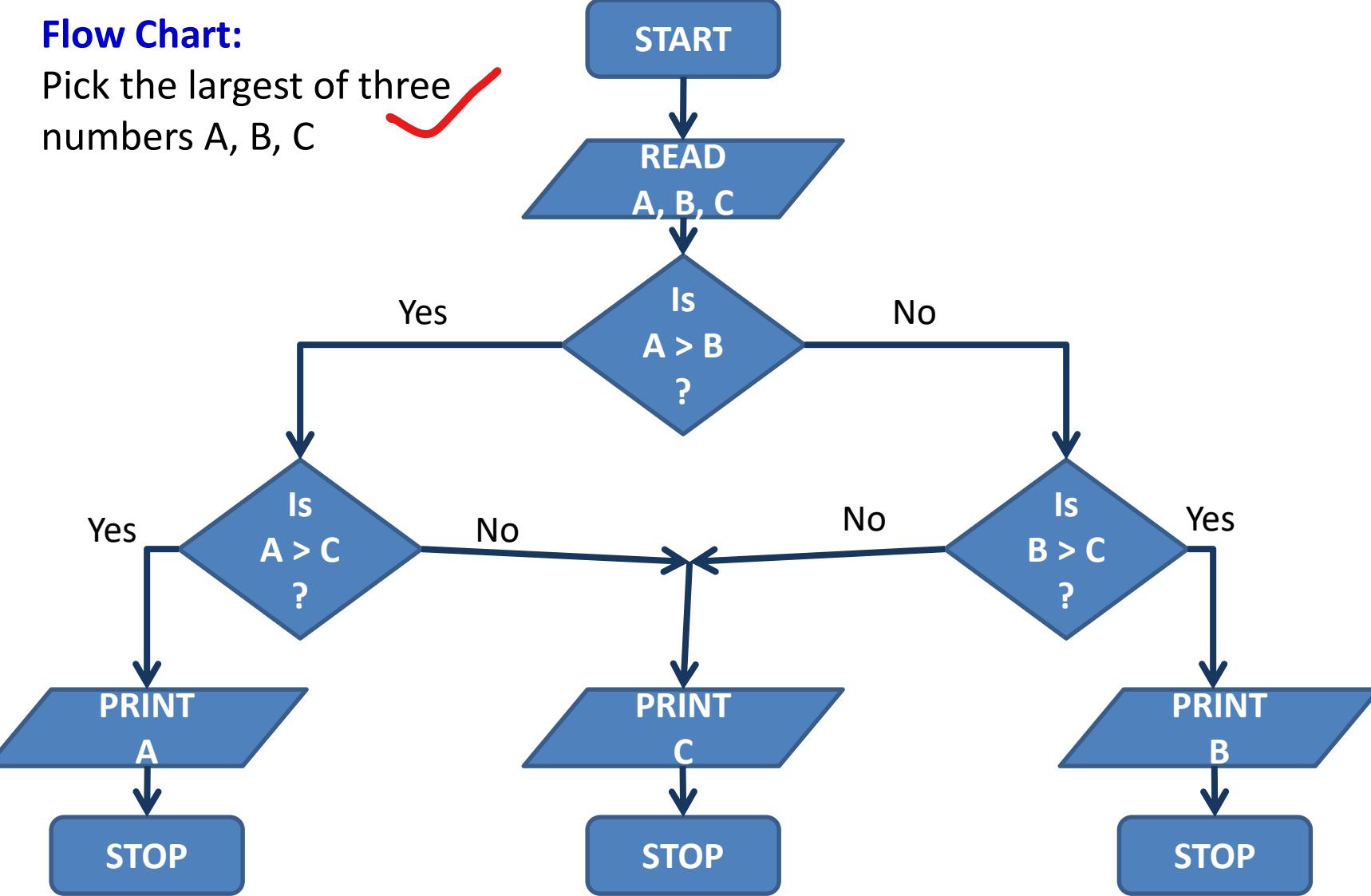
- Fundamental knowledge necessary to solve problems using a computer.
- Definition:
 - finite sequence of instructions to be carried out in order to solve a given problem.
- Instruction must be written in a precious *notation*, can be interpreted and executed by a computing machine are called **computer programming**
- The *notation* is called computer **programming language**.
- **Programming language**
 - artificial language that can be used to control the behavior of a computer
 - defined by **syntactic** and **semantic** rules which describe their **structure** and **meaning** respectively.
- Example:
different syntaxes (languages), but result in the same semantic:
 - $x += y$; (C, Java, etc.)
 - $x := x + y$; (Pascal)
 - Let $x = x + y$; (early BASIC)
 - $x = x + y$ (most BASIC dialects, Fortran)

Developing Algorithms

- Flow Charts
 - illustrates pictorially the sequence in which instructions are carried out in an algorithm.

Flow Chart:

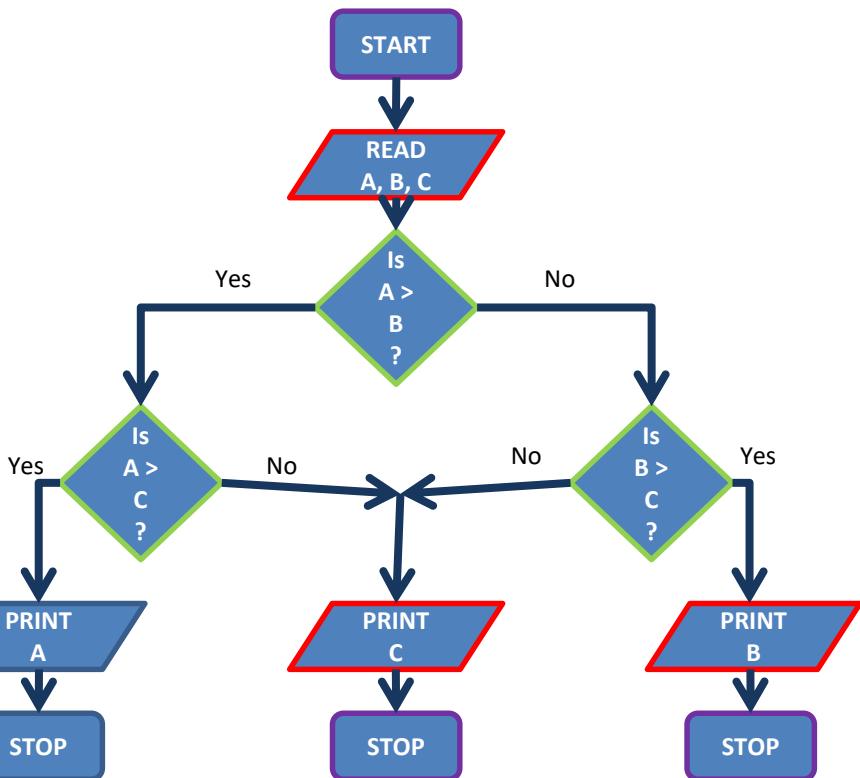
Pick the largest of three numbers A, B, C



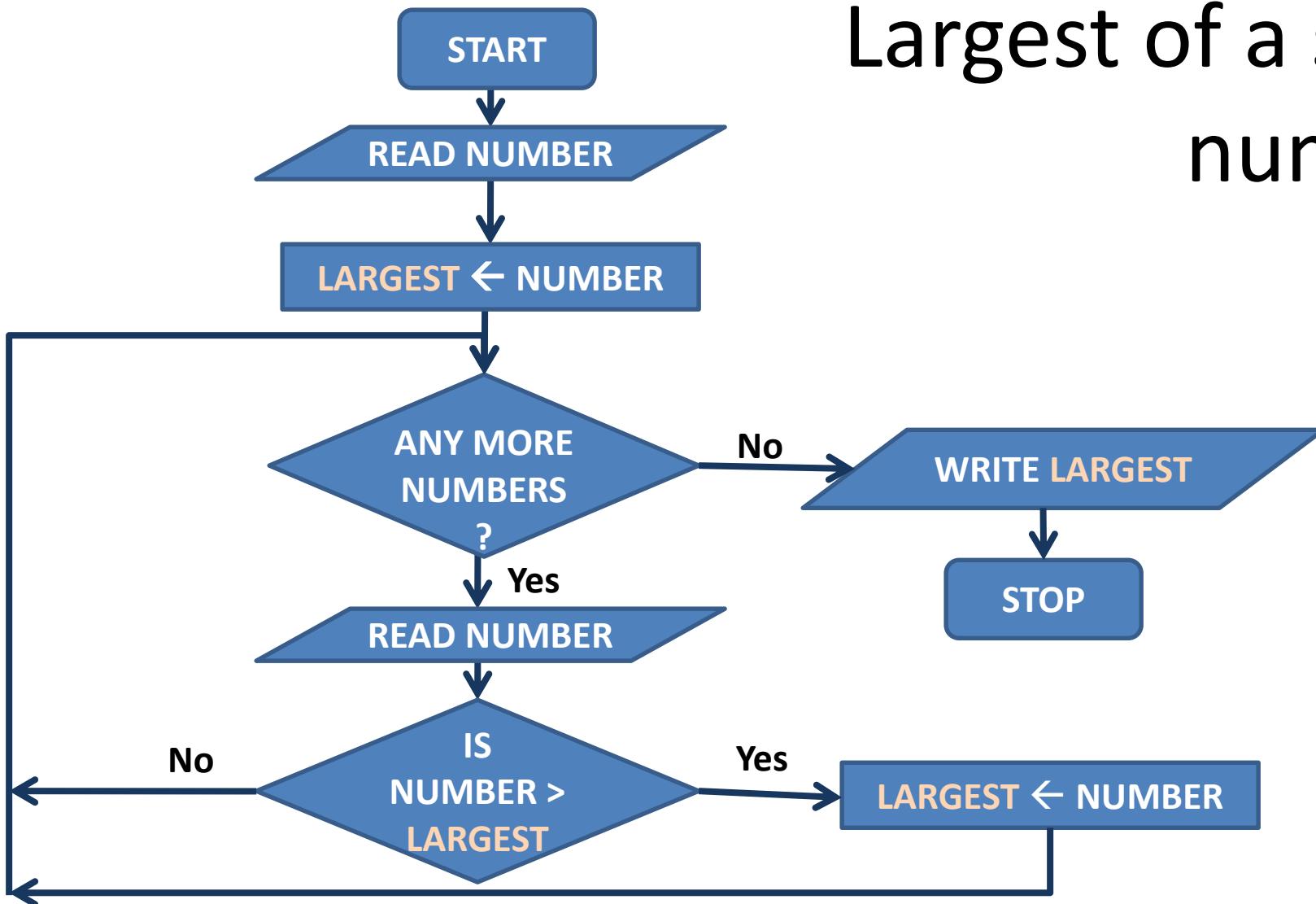
Flow Charts

- Convention

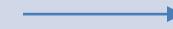
- Parallelograms are used to represent input/output.
- Rectangles are used to indicate any processing operation such as storage and arithmetic.
- Diamond shaped boxes are used to indicate questions asked or conditions tested.
- Rectangles with rounded ends are used to indicate the beginning or end points.
- Circle is used to join different parts of a flow chart, called connector.
- Arrows indicate the direction to be followed in a flow chart.



Largest of a set of numbers



Symbol for flow chart

Start/Stop	Read/ Print	Processing statements	Conditional check	Direction of flow	Connectors
				   	

✓ Assignments

- Design Flow Charts
 1. to test a number is prime (output: Yes/No)
 2. to test two numbers are relatively prime (output: Yes/No)
 3. to find GCD (output is a value)
 4. to find LCM (output is a value)
 5. to test a, b, c are side of a right-angle triangle (output: Yes/No)
 6. to test points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ are collinear (output: Yes/No).

Example C Program

```
/* A program that prints on the screen the
   phase “I love C programming”*/
#include<stdio.h>

int main(void){
    printf("I love C programming");
    return 0;
}
```

Example C Program

```
/* A program that prints on the screen the
   phase “I love C programming”*/
#include<stdio.h>

int main(void){
    printf("I love C programming\n");
    return 0;
}
```

Example C Program

```
/* A program that prints on the screen the
   phase “I love C programming”*/
#include<stdio.h>

int main(void){
    printf("I love ");
    printf("C programming");
    printf("\n");
    return 0;
}
```

Example C Program

```
/* A program that prints on the screen the
   phase “I love C programming”*/
#include<stdio.h>

int main(void){
    printf("I love\n");
    printf("C\n");
    printf("programming\n");
    return 0;
}
```

Example C Program

```
/* A program that prints on the screen the
   phase “I love C programming”*/
#include<stdio.h>
int main(void){
    printf("I love\n");
    printf("C\n programming\n");
    return 0;
}
```

✓ Exercise (Flow Charts & C Programming)

- 
1. Solve: $Ax^2 + Bx + C = 0$ for given (A,B,C)
 2. Find area of a triangle for given side lengths
 3. Celsius to Fahrenheit conversion
 4. The largest of a given set of integers

MA 511: Computer Programming

Lecture 3

 **Partha Sarathi Mandal**

psm@iitg.ac.in

*Partha
Sarathi
Mandal*

Dept. of Mathematics
IIT Guwahati

Example of a C program

```
/* Calculate the area of a circle */
#include<stdio.h>
int main(void){
    int radius;
    float area;
    printf("Radius = ? ");
    scanf("%d", &radius);
    area = 3.14159 * radius * radius;
    printf("Area = %f", area);
    return 0;
} /* end of main */
```

Preprocessor directive, stdio.h is included in the compiled machine code at # . It contains the standard I/O routines. Must be in the first column. Must not end with a semicolon.

In C, program execution starts from the main() function. Every C program must contain a main() function. The main function may contain any number of statements. These statements are executed sequentially in the order which they are written. The main function can in-turn call other functions. When main calls a function, it passes the execution control to that function. The function returns control to main when a return statement is executed or when end of function is reached. It is not followed by a comma or semicolon.

Braces { and } enclosed the computations carried out by main()

Comments (remarks) are placed anywhere within the program within delimiters /* end of main */ or // end...

Example of a C program

```
/* Calculate the area of a circle */
#include<stdio.h>
int main(void){
    int radius;
    float area;
    printf("Radius = ? ");
    scanf("%d", &radius);
    area = 3.14159 * radius * radius;
    printf("Area = %f", area);
    return 0;
}
/* end of main */
```

Preprocessor directive, stdio.h is included in the compiled machine code at # . It contains the standard I/O routines. Must be in the first column. Must not end with a semicolon.

main() is a function, is required in all C pgs, it indicates start of a C pgs., main() it is not followed by a comma or semicolon.

Braces { and } enclosed the computations carried out by main()

Every statement is terminated by a **semicolon**

Declaration : it informs the compiler that radius and area are variables names and that individual boxes must be reserved for them in the memory of the computer.

Comments (remarks) are placed anywhere within the program within delimiters /* end of main */ or // end...

for loop in C programming

Print 1 to 10 numbers in a console.

```
step 1      step 2      step 4  
for ( int number = 1; number <= 10; number++ )  
{  
    printf ( "%d \n ",number );  
}  
step 2      step 3      step 4
```



Example C Program

```
/* Calculate the area of a circle */
#include<stdio.h>
#define PI 3.14159
int main(void){
    float radius, area;
    int i, n;
    printf("n = ? ");
    scanf("%d", &n);
    for(i=1; i<= n; i=i+1){
        printf("Radius = ? ");
        scanf("%f", &radius);
        area = PI * radius * radius;
        printf("Area = %f\n", area);
    } /* end of for */
    return 0;
} /* end of main */
```

Symbolic constant (PI) is a name that substitutes for a sequence of characters; numeric, character or string constant. It is replaced by its corresponding character constant during compile

printf & scanf are not part of the C language; there is no input or output defined in C itself. Its just a useful function from the standard library of functions that are normally accessible to C pgs. The behavior of printf & scanf are defined in the ANSI standard.

for is looping statement , the 1st expression specifies an initial value for an index, 2nd determines whether or not the loop is continued, 3rd allows the index to be modified at the end of each pass.

Some standard keywords

Keywords in C Programming

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

They cannot be used as identifier/variable.

auto
enum
extern
static
union
typedef
volatile
register

Rules for naming identifiers

- But the **first letter** of an **identifier** must be either a letter (both uppercase and lowercase) or an underscore but **NOT a number**.
- A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.

Valid identifier:

x, i, j, i_, y13, sum_1, names, area, PI, tax_rate, _tempatature, TABLE

NOT valid identifier:

- 6th : the first character is a number
- “a” : illegal character (“)
- order-no : illegal character (-)
- error flag : illegal character (blank space)

Data types

1. Integer constant

int a, b; (for 32bit machine)

long int a, b;

Signed (- 2^{31} to + $2^{31} - 1$), Unsigned : (0 to $2^{32} - 1$)

short int a, b;

Signed: Considered 16 bit integer with range (- $2^{15} = -32768$ to + $2^{15} - 1 = +32767$), Unsigned (0 to $(2^{16} - 1) = 65535$)

2. Floating point constant

float a = 2.0, b = 0.9999, sum = 0.; [restricted within 3.4×10^{-38} to 3.4×10^{38}]

double fact = 0.11236E-6; [0. 11236 $\times 10^{-6}$] [within 1.7×10^{-308} to 1.7×10^{308}]

3. Character constant

char a = 'x', b = '3', c = '#', text[18] = "kolkata";

4. String constant

string a = "Delhi, 100011", b = "\$20.95";

Constant

There are four basic types of constants in C

1. Integer constants
 2. Floating-point constants
 - Ex. $3 \times 10^5 = 3e5/.3E6/.003e+3$
 3. Character constants
 - char flag; ←
 4. String constants
 - char name[80]; ←
- Symbolic constants
– #define PI 3.141593 } }

2's complement

Integer constant

int a, b; (for 32-bit machine)

long int a, b;

Signed (- 2^{31} to + $2^{31} - 1$),

Unsigned : (0 to $2^{32} - 1$)

short int a, b;

Signed: Considered 16 bit integer with range
(- $2^{15} = -32768$ to + $2^{15} - 1 = +32767$),

Unsigned (0 to $(2^{16} - 1) = 65535$)

Two's complement	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Subtraction of two numbers using 2's Complement

To subtract b from a.

We can write $(a - b)$ as:

$$(a - b) = a + (-b)$$

$(-b)$ can be written as 2's complement of b.

Now, $(a - b)$ can be written as
 $a + (2\text{'s complement of } b)$

So, the problem now reduces to "Add a to the 2's complement of b".

Compute $a - b$ for
 $a = 2$ and $b = 3$.

• Binary representation of 3 is: 0011

1's Complement of 3 is: 1100

$$2^0 + 2^1 = 3$$

2's Complement of 3 is: (1's Complement + 1) i.e.

$$\underline{1100} \quad (\text{1's Compliment})$$

$$+1$$

$$\underline{\underline{1101}}$$

2's Complement i.e. -3

$$\underline{\underline{0010}}$$

(2 in binary)

$$\underline{\underline{-3}}$$

$$\checkmark + \underline{\underline{1101}} \quad (-3)$$

$$\underline{\underline{1111}} \quad (-1)$$

$$\underline{\underline{0010}}$$

$$\underline{\underline{0011}}$$

Now, to check whether it is -1 or not simply takes 2's

Complement of -1 and kept -ve sign as it is.

$$2\text{'s Complement} = -(0000)$$

$$-(0001)$$

$$-(0000)$$

$$-(0001)$$

$$-(0000)$$

$$-(0001)$$

$$-(0000)$$

$$-(0001)$$

$$-(0000)$$

$$-(0001)$$

Variable declaration

int i,j,k;

long p,q;

short s;
unsigned u;

float r;

double dr;

long double lr;

char c, text[10];

-ve
+ve
0
2's complement
(1's complement
+1)

Operators

Operator purpose

+	addition
-	subtraction
*	multiplication
/	division
%	remainder after integer division (module operator)

Operands conversion

- float @ double = double @ : operator
- float @ char/long int/short int/int = float
- long int @ char/short int/int = long int

int i = 7;

float f = 5.5;

char c = 'w' [w= 119 (ASCII) American Standard Code for Information Interchange]

i+f = 12.5 float

i+c = 126 int

i+c-'0' = 78 int [zero, 0= 48 (ASCII)]

(i+c)-(2*f/5) = 123.8 float

MA 511: Computer Programming

Lecture 4

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Last class highlight

- Example C Program
- Data types
- Variable declaration
- Operators
- Operands conversion

Type Casting

- The value of an expression can be converted to a different data type if desire follows
- (*data type*) *expression*
- Example:

int i = 7;

float f = 8.5;

$(i+f)\%4$ = invalid since $i+f$ is floating point.

((*int*) (*i+f*)%4 =3

Variable

- Identifier, its used to represent some specific type (single data item) of information within a portion of program.

```
main(){  
    int a, b, c;  
    char d;  
    a = 2;  
    b = 1;  
    d = 'u'  
    c= a+b;  
    .....  
  
    a = 6;  
    b = 10;  
    d = 'X'  
    c = a*b;  
}
```

•

Unary operators

- Two unary operators:
 - Increment operator: `++` (increased by 1)
`++i` equivalent `i=i+1` //
`i++` //value of the operand will altered after it is utilized
Example:

```
a=10; b=20;
x= ++a;
y=b++;
printf("x = %d a = %d\n", x, a) : x = 11 a = 11
printf("y = %d b = %d\n", y, b) : y = 20 b = 21
x = ++a equivalent to the following two sequence a = a + 1; then x = a;
Y = b++ equivalent to the following two sequence y = b; then b = b + 1;
X = a++ - ++a; undefined [dependent on compiler's implementation]
```
 - Decrement operator: `--` (decreased by 1)
`--i` equivalent `i=i-1` and `i--`
 - `sizeof(type)`: Example: `printf("integer: %d\n", sizeof(float));`

Relational and logical operators

• Operator	Meaning		
<	less than		
<=	less than or equal to		
>	grater than		
>=	grater than or equal to		
==	equal to		
!=	not equal to		
			or
&&	and		

ASCII (American Code for information Interchange) Table and Description

[0 - 128]

[32 - 47]

[Special charc.]

[48 - 54]

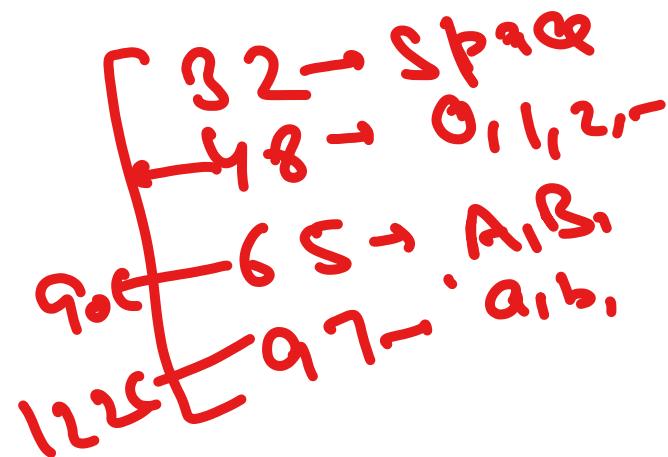
...
65

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	00	NUL	16	10	DLE	32	20	(space)	48	30	0
1	01	SOH	17	11	DC1	33	21	!	49	31	1
2	02	STX	18	12	DC2	34	22	"	50	32	2
3	03	ETX	19	13	DC3	35	23	#	51	33	3
4	04	EOT	20	14	DC4	36	24	\$	52	34	4
5	05	ENQ	21	15	NAK	37	25	%	53	35	5
6	06	ACK	22	16	SYN	38	26	&	54	36	6
7	07	BEL	23	17	ETB	39	27	'	55	37	7
8	08	BS	24	18	CAN	40	28	(56	38	8
9	09	TAB	25	19	EM	41	29)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	.
11	B	VT	27	1B	ESC	43	2B	+	59	3B	.
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	:	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Symbol									
64	40	@	80	50	P	96	60	'	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	
77	4D	M	93	5D]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Conditional Statements

- int i = 7;
 - float f = 5.5;
 - char c = 'w' [w= 119 (ASCII)]
 - Expression:
 - (i>=6)&&(c=='w')
 - (i>=6) || (c==119)
 - (f<11)&&(i>100)
 - (c!='p') || ((i+f)<=10)
- if(*logical expression*)
 CS1
 CS2
 ...
} /* end of if */

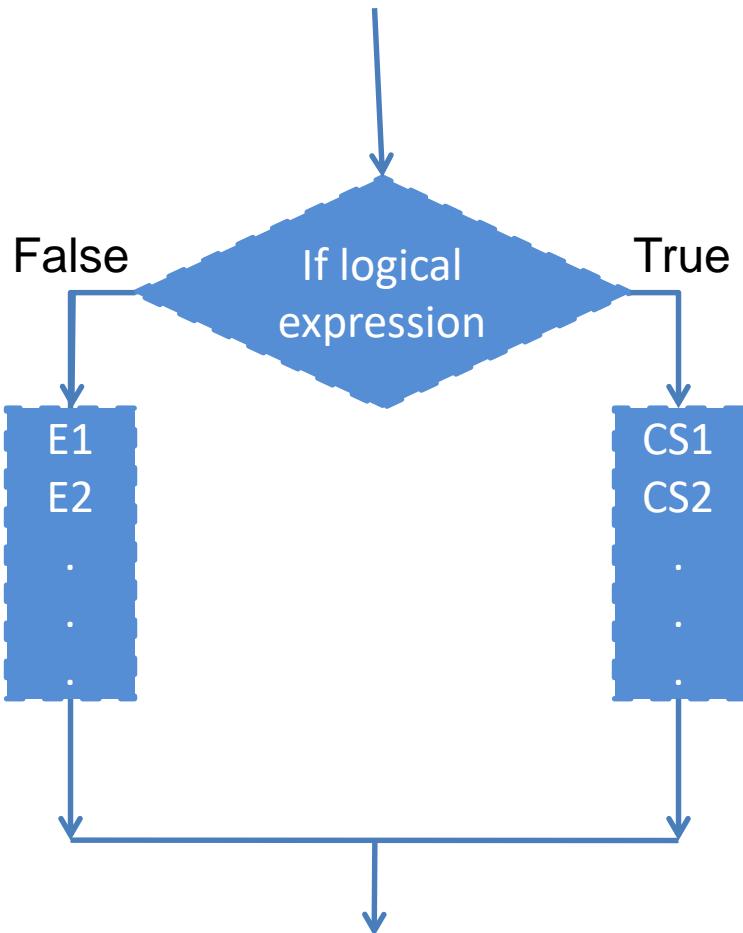


Hierarchy of operator precedence

Operator Category	Operators	Associativity
Unary operators	+ - ! sizeof (type)	R -> L
Arithmetic multiply, divide and remainder	* / %	L -> R
Arithmetic add and subtract	+ -	L -> R
Relational operators	< <= > >=	L -> R
Equality operators	== !=	L -> R
Logical and	&&	L -> R
Logical or		L -> R
Assignment operators	= += -= *= /= %=	R -> L

- $i \geq 6 \ \&\& \ c == 'w'$
- $i \geq 6 \ \parallel \ c == 119$
- $f < 11 \ \&\& \ i > 100$
- $c != 'p' \ \parallel \ i+f \leq 10$

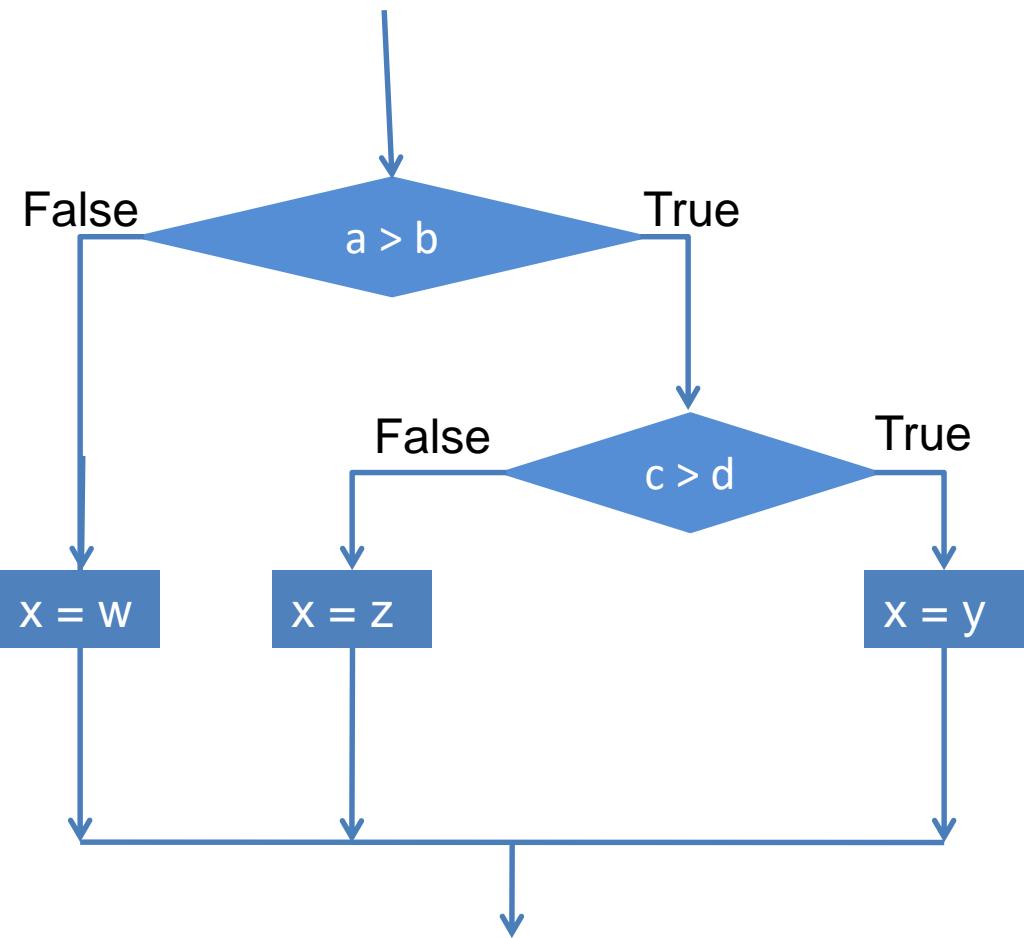
✓ Conditional statements



```
if(logical expression)  
  CS1  
  CS2  
  ...  
} /* end of if */
```

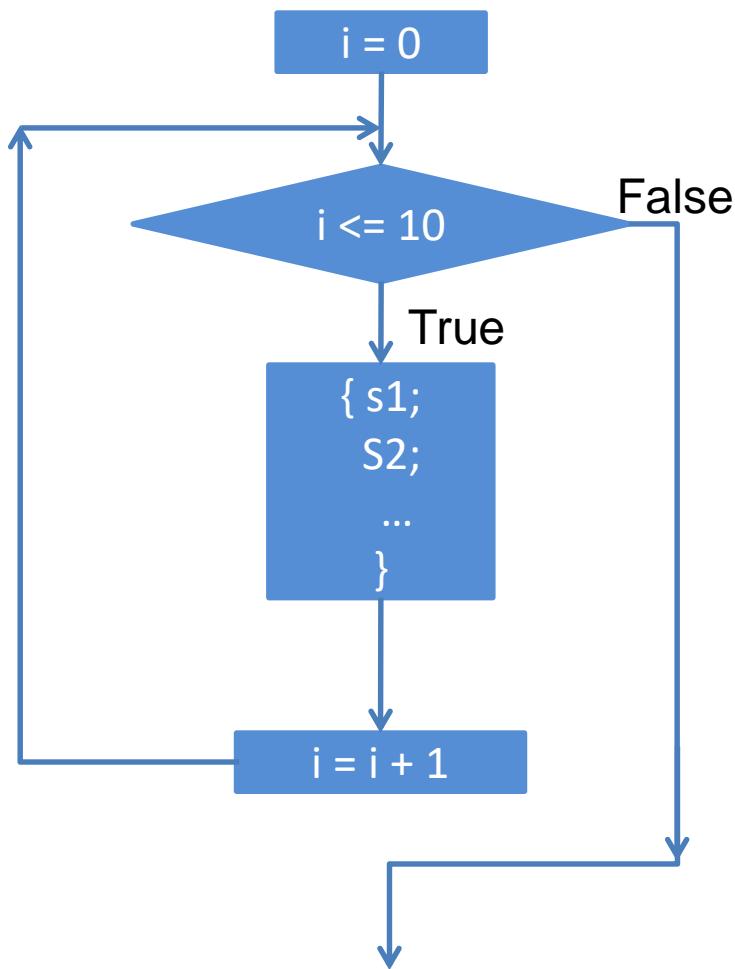
```
if(logical expression)  
  CS1  
  CS2  
  ...  
}  
else{  
  E1  
  E2  
  ...  
} /* end of if */
```

~~Nested conditional statement~~



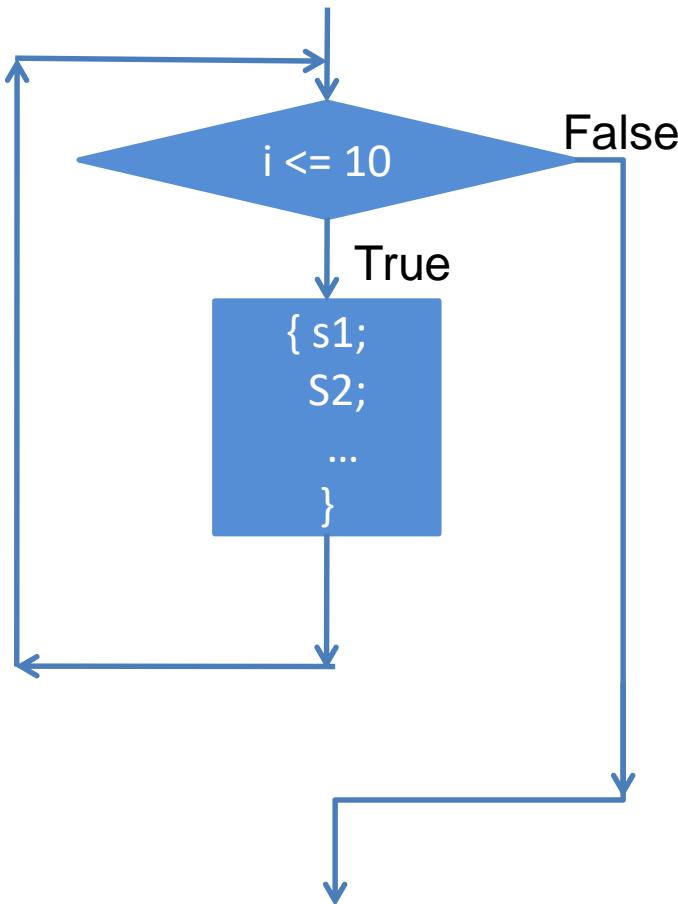
```
if(a > b){  
    if(c > d){  
        x=y;  
    } else{  
        x=z;  
    }  
    else{  
        x=w;  
    } /* end of if */
```

for loop



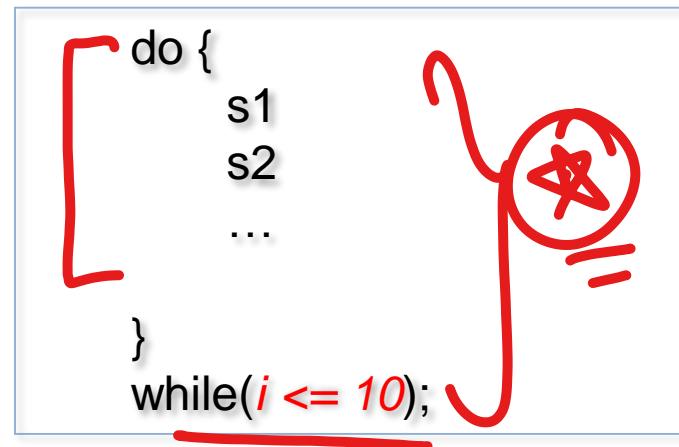
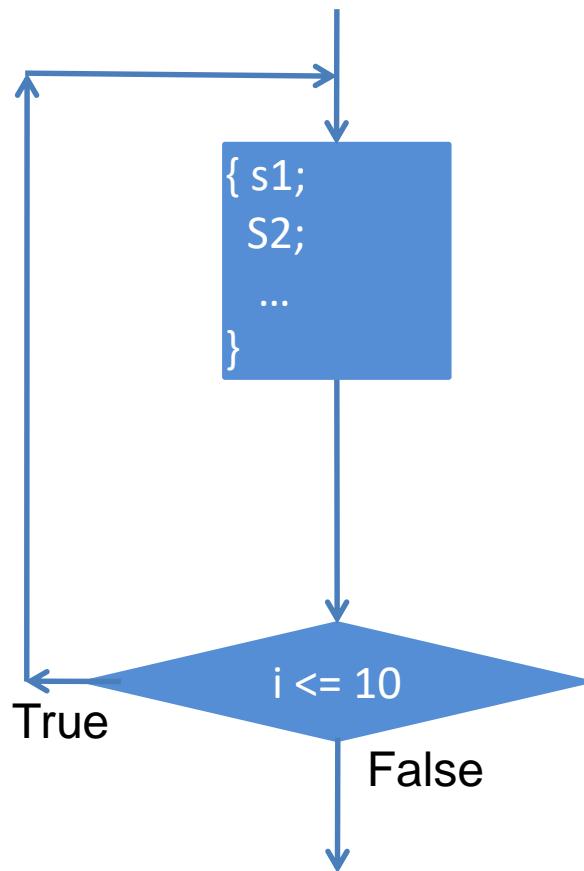
```
for(i = 0; i <= 10; i = i+1){  
    s1  
    s2  
    ...  
} /* end of for */
```

while loop



```
while(i <= 10){  
    s1  
    s2  
    ...  
} /* end of while */
```

do while loop



Assignment operators

- Five additional assignment operators:

$+ =$, $- =$, $* =$, $/ =$, $\% =$

$expression1 @ = expression2$

is equivalent to

$expression1 = expression1 @ expression2$

Where $@ = +, -, *, /, \%$

Example: $i+=5$ is equivalent to $i = i + 5$

$i\%=(j-2)$ is equivalent to $i = i \% (j-2)$

Conditional operator

- $\underline{\text{expression1}} \ ? \underline{\text{expression2}} : \underline{\text{expression3}}$

Example:

1. $(i < 0) ? 0 : 100$

- Expression $i < 0$ is evaluated first. If it is *true* the entire conditional expression takes on the value 0 otherwise 100.

2. $\text{min} = \underline{(f < g)} ? f : g$

3. $c += \underline{(a > 0)} \ \&& \ \underline{a \leq 10} ? \underline{++a} : \underline{a/b};$

- If the operands differ in type, then the resulting data type of the conditional expression will be determined by the rules taught before.

Example: float f, g; int i;

$$(f < g) ? i : g$$

MA 511: Computer Programming

Lecture 5

[**Partha Sarathi Mandal**]
psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Last class highlight

- Type Casting
- Unary operators
- Relational and logical operators
- Conditional Statements
- Hierarchy of operator precedence
- if-else
- loops



Conditional operator

- $\underline{\text{expression1}}$? $\underline{\text{expression2}}$: $\underline{\text{expression3}}$] ✓

Example:

1. $(i < 0) ? 0 : 100$
 - Expression $i < 0$ is evaluated first. If it is *true* the entire conditional expression takes on the value 0 otherwise 100.
 2. $\text{min} = (f < g) ? f : g$
 3. $c += (a > 0 \&\& a \leq 10) ? ++a : a/b;$
- If the operands differ in type, then the resulting data type of the conditional expression will be determined by the rules taught before.

Example: float f, g; int i;
 $(f < g) ? i : g$

Example

```
/* use of conditional operator and type casting */

#include <stdio.h>
int main(void) {
    float f =3.0, g=5.0;
    int i=1;
    //printf("k=%.8f\n", (f < g) ? i/3 : g);
    printf("k=%f\n", (f < g) ? (float)i/3 : g);

    return 0;
}
```

Array



- Variable identifier, refers to a collection of the same type of data items that all have the same name.
- Individual data items are represented by their corresponding array elements.
- Example: int x[5]; // x[5] is basically x[0], x[1], ..., x[4]

How to STORE value in one and two dimensional arrays

```
float A[10], B[4][5];
for(i=0; i< 10; i=i+1){
    printf("type float value for A[%d]", i);
    scanf("%d", &A[i]);
}

-----
for(i = 0; i < 4; i = i + 1 ){
    for(j = 0; j < 5; j = j + 1){
        printf("type float value for B[%d][%d]", i, j);
        scanf("%d", &B[i][j]);
    }
}
```

Array ✓

How to PRINT of one and two dimensional arrays

```
float A[10], B[4][5];  
printf("A[5]\n");  
for(i=0; i< 10; i=i+1){  
    printf("%d ", A[i]);  
}  
printf("\n");
```

```
printf("B[4][5]\n");  
for(i = 0; i < 4; i = i + 1 ){  
    for(j = 0; j < 5; j = j + 1){  
        printf("%d ", &B[i][j]);  
    }  
    printf("\n");  
}  
printf("\n");
```

Example

```
/* 2D array */

#include <stdio.h>

int main(void) {
    int i,j ;
    float A[50], B[4][5];
    for(i = 0; i < 2; i = i + 1 ){
        for(j = 0; j < 2; j = j + 1){
            printf("type float value for B[%d][%d]", i, j);
            scanf("%f", &B[i][j]);
        }
    }
    for(i = 0; i < 2; i = i + 1 ){
        for(j = 0; j < 2; j = j + 1){
            printf("%f ", B[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

return 0;
}
```

Exercise

- Write c-codes for
 1. Write c-code for testing two given integer are relatively prime.
 2. Write a program for testing a given integer is a Fibonacci number.
 3. $\text{Sin}(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots$ x is in radians
 - i. c-code for sum of the first n terms (input x and n)
 - ii. Adding successive terms in the series until the value of the next term smaller than 10^{-5} in magnitude.
 4. Addition of two $m \times n$ matrices.
 5. Multiplication of $m \times k$ and $k \times n$ matrices.
 6. Transpose a square matrix.
 7. Multiplication of two polynomial of degree m and n respectively where coefficients are integer.

MA 511: Computer Programming

Lecture 6

[Partha Sarathi Mandal]

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Some Terminology

- **Programming:** instructing the computer about how to do a certain task.
- **Program:** the set of instructions.
- **Programmability:** the ability of the computer to accept a set of instructions and carry them out.

Programmability is what primarily distinguishes a computer from a calculator.

Algorithms and Programs

- An algorithm is also a sequence of steps to solve a problem.
 - But usually written in a natural language (such as English), precisely enough so as to be unambiguously understood by humans.
- A program is an algorithm translated to a programming language, that a computer can understand.

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of result to 1
3. While $n > 1$ do (steps 31 and 32)
 31. Set new value of the result to (old value of result times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of result.

n []

result []

Algorithm to compute $n!$

1. Read the value of n *// input $n = 5$*
2. Set the value of $result$ to 1
3. While $n > 1$ do (steps 31 and 32)
 31. Set new value of the $result$ to (old value of $result$ times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of $result$.

n [**5**] $result$ []

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of **result** to 1
3. While $n > 1$ do (steps 31 and 32)
 31. Set new value of the *result* to (old value of *result* times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of *result*.

n [5]

result [1]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
3. **While $n > 1$ do (steps 31 and 32)**
 31. Set new value of the *result* to (old value of *result* times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of *result*.

n [5]

result [1]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of **result** to 1
3. While $n > 1$ do (steps 31 and 32)
 - 31. Set new value of the *result* to (old value of *result* times the value of n).**
 32. Set new value of n to (old value of $n-1$)
4. Print the value of **result**.

n [5]

result [5]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
3. While $n > 1$ do (steps 31 and 32)
 31. Set new value of the *result* to (old value of *result* times the value of n).
 - 32. Set new value of n to (old value of $n-1$)**
4. Print the value of *result*.

n [4]

result [5]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
3. **While $n > 1$ do (steps 31 and 32)**
 31. Set new value of the *result* to (old value of *result* times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of *result*.

n [4]

result [5]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of **result** to 1
3. While $n > 1$ do (steps 31 and 32)
 - 31. Set new value of the *result* to (old value of *result* times the value of n).**
 32. Set new value of n to (old value of $n-1$)
4. Print the value of **result**.

n [4]

result [20]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
3. While $n > 1$ do (steps 31 and 32)
 31. Set new value of the *result* to (old value of *result* times the value of n).
 - 32. Set new value of n to (old value of $n-1$)**
4. Print the value of *result*.

n [3]

result [20]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
3. **While $n > 1$ do (steps 31 and 32)**
 31. Set new value of the *result* to (old value of *result* times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of *result*.

n [3]

result [20]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of **result** to 1
3. While $n > 1$ do (steps 31 and 32)
 - 31. Set new value of the *result* to (old value of *result* times the value of n).**
 32. Set new value of n to (old value of $n-1$)
4. Print the value of **result**.

n [3]

result [60]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
3. While $n > 1$ do (steps 31 and 32)
 31. Set new value of the *result* to (old value of *result* times the value of n).
 - 32. Set new value of n to (old value of $n-1$)**
4. Print the value of *result*.

n [2]

result [60]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of result to 1
3. **While $n > 1$ do (steps 31 and 32)**
 31. Set new value of the result to (old value of result times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of result .

n [2]

result [60]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of **result** to 1
3. While $n > 1$ do (steps 31 and 32)
 - 31. Set new value of the *result* to (old value of *result* times the value of n).**
 32. Set new value of n to (old value of $n-1$)
4. Print the value of **result**.

n [2]

result [120]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
3. While $n > 1$ do (steps 31 and 32)
 31. Set new value of the *result* to (old value of *result* times the value of n).
 - 32. Set new value of n to (old value of $n-1$)**
4. Print the value of *result*.

n [1]

result [120]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
- 3. While $n > 1$ do (steps 31 and 32)**
 31. Set new value of the *result* to (old value of *result* times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of *result*.

n [1]

result [120]

Algorithm to compute $n!$

1. Read the value of n
2. Set the value of *result* to 1
3. While $n > 1$ do (steps 31 and 32)
 31. Set new value of the *result* to (old value of *result* times the value of n).
 32. Set new value of n to (old value of $n-1$)
4. Print the value of *result*.

n [1]

result [120] **Output: 120**

Exercise

- Write an algorithm to compute the quotient and remainder when a given integer x ($x \geq 0$) is divided by another given integer y ($y > 0$).

Assume that only addition and subtraction are available as primitive arithmetic operators.

Properties of Quotient and remainder

- If q are r quotient and remainder respectively obtained after dividing x by y .

$$q * y + r = x$$

$$r < y$$

Algorithm for calculating quotient and remainder when $x|y$ and $x>y>0$.

1. Read the values of x and y

2. Set value of q to 0

3. While ($x \geq y$) do

$x \leftarrow x-y$

$q \leftarrow q+1$

4. Print results: remainder is x and quotient is q

Program to calculate quotient and remainder when $x|y$ when $x>y>0$

```
/* Program to calculate quotient and remainder when x|y where x>0 and y>0 */

#include<stdio.h>
main(){
    int x, y, q=0;

    printf("Enter two intergers x and y (x>=y): ");
    scanf("%d%d", &x, &y);
    if(x<y)
        printf("x=%d < y= %d, division is not possible\n",x,y);
    else {
        while(x>=y){
            x=x-y;
            q=q+1;
        }

        printf("remainder is %d and quotient is %d\n", x, q);
    }
} /* end of main*/
```

Program to calculate quotient and remainder when $x|y$ when $x>y>0$

```
/* Program to calculate quotient and remainder when x|y where x>0 and y>0 */

#include<stdio.h>
main(){
    int x, y, q=0;

    printf("Enter two intergers x and y (x>=y): ");
    scanf("%d%d", &x, &y);
    while(x>=y){
        x=x-y;
        q=q+1;
    }

    printf("remainder is %d and quotient is %d\n", x, q);
}
} /* end of main*/
```

Algorithm to compute n^{th} Fibonacci number, $F(n)$

```
#include<stdio.h>

main(){
    int i, a, b, c, n;
    printf("Enter an interger: ");
    scanf("%d", &n);
    if(n<0)
        printf("%d is not a non-negative number\n", n);
    else {
        i=1;
        a=1;
        b=0;
        while( i < n ){
            c=a+b;
            b=a;
            a=c;
            i=i+1;
        }
        printf("F(%d) = %d\n", n, a);
    }
} /* end of main*/
```

Exercise

1. For a given positive integer k check whether or not k is a Fibonacci number.



Library Functions

ceil → g.i.f
floor → l.i.f

math.h : ceil(d), floor(d), sin(d), cos(d), tan(d),
sqrt(d), cosh(d), exp(d), fabs(d), log(d),
pow(d1,d2),...

[stdlib.h] : rand(), srand(u), abs(i), tolower(c),
toupper(c),.....

stdio.h : printf(), scanf(), getchar(), putchar()...

string.h : strcpy(s1, s2), strcmp(s1,s2), strlen(s1)...

Random number generator

```
int seed, s; ✓  
double r;  
seed = 10000; // choose a seed value ✓  
srand(seed); // initialize random number generator  
s=rand(); // random integer  
r=((double)rand() / ((double)(RAND_MAX)+(double)(1))); }  
// random number in [0, 1)
```

Where RAND_MAX may be the largest positive integer the architecture can represent.

Assignments

1. Write a c-code for generating n arbitrary (random) points in a square of size n , then identify and report which of them are placed
 - i) inside ii) outside and iii) on a given circle (center and radius are given as input parameters).
2. over the above given points calculate distance between all pair of points and report the maximum and minimum distances.

MA 511: Computer Programming

Lecture 7

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Input & Output

- Following functions permits the transfer of information between computer and standard input and output.
 - getchar, putchar, scanf, printf

[○ getchar & putchar] Example: single character

```
char c;  
printf("Enter any char value: ");  
c = getchar();  
printf("the corresponding uppercase char : ");  
putchar(toupper(c));
```

scanf: String reading

- `char text[80];`
- `scanf(" %[^\\n]", text);` $\wedge \backslash n$ --- circumflex and backslash n

$\%[^\wedge \backslash n]$

$\cdot \backslash . c$

```
/* write a c program to find the length of the string using strlen() function */
#include<stdio.h>
#include<string.h>
main(){
    char name[100];
    int length;
    printf("Enter the string");
    scanf("%s", name);
    length=strlen(name);
    printf("\nNumber of characters in the string is=%d\n",length);
}
```

- `scanf` statement has a **draw back** it just terminates the statement as soon as it finds a **blank space**, suppose if we type the string New York then only the string new will be read and since there is a blank space after word “New” it will terminate the string.

scanf: String reading

- char text[80];
- scanf("%[^\\n]", text);

```
/* writr a c program to find the length of the string using strlen() function */
```

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

main(){
    char name[100];
    int length;

    printf("Enter the string");
    scanf("%[^\\n]", name);
    length=strlen(name);
    printf("\nNumber of characters in the string is=%d\n",length);
}
```

scanf

scanf("%3d %3d %3d", &a, &b, &c)

Input: 1 2 3

Output 1 2 3

Input 123 234 456

Output 123 234 456

Input 123234345

Output 123 234 345

Input 1234 2345 5

Output 123 4 234

~~scanf("%3d, %3d, %3d", &a, &b, &c)~~



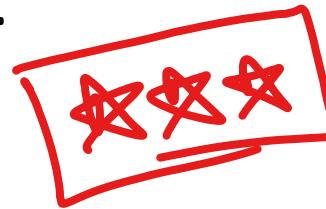
scanf

- float f
 - short ix, iy;
 - long lx, ly;
 - double dx, dy;
 - `scanf("%4f %hd %ld %lf", &f, &ix, &lx, &dx);`
 - `scanf("%3ho %7lx %15le", &iy, &ly, &dy);`
- o: octal x: hexadecimal e: double-precision
h: short l: long
- Short Octal*
- short int*
- long int*
- double*
- long hexa.*
- double Precision*

~~Strings manipulation~~ [Strings manipulation ?]

- We cannot manipulate strings since C does not provide any operators for string. For instance we cannot assign one string to another directly.

String="xyz";
string1=string2;



- Are not valid.
- To copy the chars in one string to another string we may do so on a character to character basis.
- char a = 'x', b = '3', c = '#', text[18] = "guwahati";
- Are valid

String operations (string.h)

- Length (number of characters in the string).
 - length=strlen(name);
- Concatentation (adding two are more strings)
 - strcpy(string1,"sri");
 - strcpy(string2,"Bhagavan");
 - Printf("%s",strcat(string1,string2));
- Comparing two strings.
 - strcmp(string1,string2)
- Copy(copies one string over another)
- Exercise: Substring (Extract substring from a given string)

Example

Read a string than replace each character with an equivalent encoded character

```
char line[80];
int i;
printf("Type a line of text\n");
scanf("%[^n]", line);
for(i=0; line[i] != '\0'; ++i){
    if(((line[i]>='0') && (line[i]<'9')) || ((line[i]>='A') && (line[i]<'Z')) || ((line[i]>='a') && (line[i]<'z')))
        putchar(line[i]+1);
    else if (line[i] == '9')
        putchar('0');
    else if (line[i] == 'Z')
        putchar('A');
    else if (line[i] == 'z')
        putchar('a');
    else
        putchar('.');
}
```

Input: IIT Guwahati, 781039, Assam, India.

Output: JJU.Hvxbibuj..892140..Bttbn....Joejb.

ASCII

Character ASCII value

0 → 48

9 → 57

A → 65

Z → 90

a → 97

z → 122



Assignments

1. Read a string of alphabets (a to z or A to Z) than replace each character with an equivalent encoded character as follows.
2. A or a – 1
3. B or b – 2
26. Z or z- 26

Assignment

Solve the following algebraic Equation:

$$x^5 + 3x^2 - 10 = 0$$

MA 511: Computer Programming

Lecture 8

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

switch Statement

- Switch statement causes a particular group of statement to be chosen from several available groups.
- The selection is based upon the current value of a expression which is included within the switch statement.

switch Statement



Example:

```
switch(choice = toupper(getchar()) ){  
    case 'R':  
        printf("RED");  
        break;  
    case 'W':  
        printf("WHITE");  
        break;  
    case 'B':  
        printf("BLUE");  
        break;  
    default:  
        printf("ERROR");  
}
```

A red arrow points from the word "choice" in the first line of the code to the handwritten note "(single =)" above the code.

switch Statement

Example:

```
switch(flag){  
    case -1:  
        printf("y=%f", y=abs(x));  
        break;  
    case 0:  
        printf("y=%f", y=sqrt(x));  
        break;  
    case 1:  
        printf("y=%f", y=x);  
        break;  
    case 2:  
    case 3:  
        printf("y=%f", y= 2* (x-1));  
        break;  
    default:  
        printf("y=0");  
}
```

switch Statement

Example: $ax^2+bx+c=0$

discernment = $b*b - 4.0*a*c$

```
If (discernment < 0) i=1;  
else if(discernment==0) i=2;  
else i=3;  
switch(i){  
    case 1:      discernment = -discernment;  
                imag = sqrt(discernment)/ (2.0*a);  
                real  = -b/ (2.0*a);  
                break;  
    case 2:      equal_root = -b/(2.0*a);  
                break;  
    case 3:      roor_1 = (- b + sqrt(discernment))/ (2.0*a);  
                root_2 = (- b - sqrt(discernment))/ (2.0*a);  
}  
}
```

comma Operator (,)

Everybody knows that in C, a for loop has the syntax:

```
for (variable initialization; exit condition; variable update) {  
    // the body comes here  
}
```

a typical example being:

```
int i;  
  
for (i = 0; i < 5; i++) {  
    printf("%d ", i);  
}
```

comma Operator (,)

- comma operator is used in conjunction with for statement

Example:

- for(exp1a, exp1b; exp2; exp3)
 - Initialize two separate indices would be used simultaneously within a single for loop
- for(exp1; exp2; exp3a, exp3b)
 - two different indices would be used simultaneously within a single for loop for example one counts forward other counts backward.
 - int i, j;
for (i=-1, j=0; i=i+1, i<5; j=j+1, printf("%d\n", j)){
 printf("%d\t", i);
}

Assignments

- A **Palindrome** is a word, phase or sentence that reads the same way either forward or backward.
- Example: *noon; Rise to vote, sir!* (if we disregard punctuation and blank space)
- Write a C program that will enter a line of text containing a word, a phase or a sentence, and determine whether or not the text is a palindrome.

break Statement

- When the **break** statement is encountered inside a **loop**, the **loop** is immediately terminated, and program control resumes at the next **statement** following the **loop**. The **break** statement can be used with all loops such as **for**, **while**, **do-while** or **switch** statements.

Example:

```
main(){
    int i, n;
    scanf("%d", &n);
    for(i=2; i<=ceil(sqrt(n)); i=i+1){
        if(!(n%i)){
            printf("n is composite");
            break;
        }
    }
    if(i=>ceil(sqrt(n))) printf("n is prime");
}
```

return Statement

- The **return** statement terminates the execution of a function and **returns** control to the calling function.

Example:

```
main(){  
    int i, n;  
    scanf("%d", &n);  
    for(i=2; i<=ceil(sqrt(n)); i=i+1){  
        if(!(n%i)){  
            printf("n is composite");  
            return;  
        }  
    }  
    printf("n is prime");  
}
```

continue Statement

- It used to bypass the remainder of the current pass through a loop such as **for**, **while**, **do-while**.

Example:

```
do{
    printf("Print x ");
    scanf("%f", &x);
    if(x < 0){
        printf("ERROR-Negative value for x");
        continue;
    }
}while(x<=100);
```

goto statement

goto *label*:

label: statement



Example:

```
int num, i=1;  
printf("Enter the number whose table you want to print: ");  
scanf("%d",&num);  
table:  
printf("%d x %d = %d\n",num , i , num*i);  
i++;  
if(i<=10)  
    goto table;
```

A large red curly brace is drawn around the entire if block, starting from the "if(i<=10)" line and ending at the "table" label.

Assignments

A gas company bills its customers according to the following rate schedule:

First	50 cubic meters	Rs. 40 (Flat Rate)
Next	300 cubic meters	Rs. 1.25 per 10 cubic meters
Next	3000 cubic meters	Rs. 1.20 per 10 cubic meters
Next	5000 cubic meters	Rs. 1.10 per 10 cubic meters

Above this Rs. 0.80 per 10 cubic meters

Given an input for each customer in the format:

<customer number, previous meter reading, current meter reading>

Write a program to output the following:

<customer number, previous meter reading, current meter reading,
gas used, Total Bill>

MA 511: Computer Programming

Lecture 9: Function Prototypes

.

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Function (int)

Example: Multiplication of 2 numbers

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

```
int multip(int a, int b);
int main(void){
    int x, y, k;
    printf("input X and Y: ");
    scanf("%d %d", &x,&y);
    k = multip(x,y);
    printf("multiplication = %d", k);
    return 0;
}
```

```
int multip(int m, int n){
    int p, q, r;
    p = m;
    q = n;
    r = p*q;
    return(r);
}
```

Alternative:

```
int multip(int m, int n){
    int r;
    r = m*n;
    return(r);
}
```

Example: Multiplication of 2 numbers

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

```
int multip(int m, int n){
    int p, q, r;
    p = m;
    q = n;
    r = p*q;
    return(r);
}
```

int main(void){

```
    int x, y, k;
    printf("input X and Y: ");
    scanf("%d %d", &x,&y);
    k = multip(x,y);
    printf("multiplication = %d", k);
    return 0;
}
```

Alternative:

```
int multip(int m, int n){
    return (m*n);
}
```

Function (float)

Example: Division of 2 numbers

```
#include<stdio.h>
float division(int a, int b);
int main(void){
    int x, y;
    float k;
    printf("input X and Y: ");
    scanf("%d %d", &x,&y);
    k = division(x,y);
    printf("division = %f", k);
    return 0;
}
float division(int m, int n){
    int p, q;
    float r;
    p = m;
    q = n;
    r = (float)p/(float)q;
    return(r);
}
```

Example: Division of 2 numbers

```
#include<stdio.h>
float division(int m, int n){
    int p, q;
    float r;
    p = m;
    q = n;
    r = (float)p/(float)q;
    return(r);
}
int main(void){
    int x, y;
    float k;
    printf("input X and Y: ");
    scanf("%d %d", &x,&y);
    k = division(x,y);
    printf("division = %f", k);
    return 0;
}
```

Function (**void**)

Example: Multiplication of 2 numbers

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

```
void multip(int a, int b);
```

```
int main(void){
```

```
    int x, y, k;
```

```
    printf("input X and Y: ");
```

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

```
    multip(x,y);
```

```
    return 0;
```

```
}
```

```
void multip(int m, int n){
```

```
    int p, q, r;
```

```
    p = m;
```

```
    q = n;
```

```
    r = p*q;
```

```
    printf("multiplication = %d", r);
```

```
}
```

Example: Multiplication of 2 numbers

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

```
void multip(int m, int n){
```

```
    int p, q, r;
```

```
    p = m;
```

```
    q = n;
```

```
    r = p*q;
```

```
    printf("multiplication = %d", r);
```

```
}
```

```
int main(void){
```

```
    int x, y, k;
```

```
    printf("input X and Y: ");
```

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

```
    multip(x,y);
```

```
    return 0;
```

```
}
```

Definition of Function

- It's a self-contained *program segment* that carries out some specific, well defined task.
- Every C program consists at least one function called **main**.
- Execution of a C program always begins by carrying out the instructions in **main**. Other functions are the subordinate to **main**.
- A function processes information that is passed (*arguments*, also called *parameters*) to it from the *calling portion* of the program, and *return* a single value.
- Some functions accept arguments but do not return anything called **void**.
- Some functions may return *multiple values*, with the help of pointer and arguments.

Function Prototypes

date-type name(*type 1 arg 1, type 2 arg 2, ... type n arg n*);

Example:

```
int multip(int a, int b);
void multion(int a, int b);
float division(int a, int b);
```

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

date-type name(*type 1 arg 1, type 2 arg 2, ... type n arg n*);

```
int main(void){
```

```
    name(arg 1, arg 2, ... arg n);
```

```
    return 0;
}
```

```
date-type name(type 1 arg 1, type 2 arg 2, ... type n arg n);
}
```

Function Prototypes

```
#include<stdio.h>

date-type name(type 1 arg 1, type 2 arg 2, ... type n arg n){  
    .  
    .  
}  
  
int main(void){  
    .  
    name(arg 1, arg 2, ... arg n);  
  
return 0;  
}
```

Exercise

- Write two functions to perform addition and subtraction of two numbers. Use them in a program to compute the sum and difference of input numbers.
- Write a function to swap two variables without using another variable. Use them in a program to print the numbers before and after the swap.

Assignments

- Write c program for calculating $\text{Sin}(x)$ up to nth terms for given inputs x and n.
- Write c program for calculating $\text{Sin}(x)$ up to the term where the value of the term is less than 10^{-5} for a given input x.

For above two problems you must use function such that the function should returns the final answer to the main.

MA 511: Computer Programming

Lecture 10: Recursion

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Programming for $n!$ without function

```
#include<stdio.h>
main(){
    int i, n, fact=1;
    printf("Type n");
    scanf("%d",&n);
    if(n<0)
        printf("%d is a -ve INPUT\n",n);
    else if(n==0)
        printf("n!=%d\n",fact);
    else{                                /* n>0 */
        for(i=1;i<=n;i=i+1){
            fact=fact*i;
        }
        printf("n!=%d\n",fact);
    }
}
```

Programming for n! with function

```
#include<stdio.h> //prog. type 1
long int factorial(int n){
    int i;
    long int product = 1;
    if (n>1){
        for(i=2; i<=n; i=i+1)
            product *=i;
    }
    return(product);
}
main(){
    int n;
    print("Type n = ");
    scanf("%d", &n);
    printf("n! = %ld ", factorial(n));
}
```

```
#include<stdio.h> //prog. Type 2
long int factorial(int n); ← function declaration
main(){
    int n;
    print("Type n = ");
    scanf("%d", &n);
    printf("n! = %ld ", factorial(n));
}
long int factorial(int n){
    int i;
    long int product = 1;
    if (n>1){
        for(i=2; i<=n; i=i+1)
            product *=i;
    }
    return(product);
}
```

Function declaration is required if the function is placed after the main.
This program can not detect wrong input, for example -ve integer.

Programming for $n!$ with function

```
#include<stdio.h>
long int factorial(int n){
    int i;
    long int product = 1;
    if (n>1){
        for(i=2; i<=n; i=i+1)
            product *=i;
    }
    return(product);
}
main(){
    int n;
    printf("Type n = ");
    scanf("%d", &n);
    if(n<0)
        printf("%d IS A WRONG INPUT\n", n);
    else
        printf("n! = %ld\n", factorial(n));
}
```

Here the function is placed before main,
so, function declaration is not required.

This program can detect wrong input, for
example -ve integer.

Assignment

Write a program using functions to

1. concatenate two strings,
2. extract a substring from a string,
3. replace a substring in a string and
4. to find the length of a string

without using standard functions such as **strcat**,
strlen, etc.



- Recursion is a process by which a function calls itself repeatedly until some specific condition has been satisfied.
- **Recursion** in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem.

Factorial

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{fact}(n - 1) & \text{if } n > 0 \end{cases}$$

function factorial is:

input: integer n such that $n \geq 0$

output: $[n \times (n-1) \times (n-2) \times \dots \times 1]$

if n is 0, **return** 1

otherwise, **return** $[n \times \text{factorial}(n-1)]$

end factorial

Example:

```
long int factorial(int n){  
    if(n<=1)  
        return(1);  
    else  
        return(n*factorial(n-1));  
}  
main(){  
    int n;  
    print("Type n = ");  
    scanf("%d", &n);  
    printf("n! = %ld ", factorial(n));  
}
```

Fibonacci

$$\text{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fib}(n - 1) + \text{fib}(n - 2) & \text{if } n \geq 2 \end{cases}$$

```
int fib(int n){  
    if n == 0:  
        return 0  
    else if n == 1:  
        return 1  
    else  
        return fib(n-1) + fib(n-2)}
```

function fib **is:**

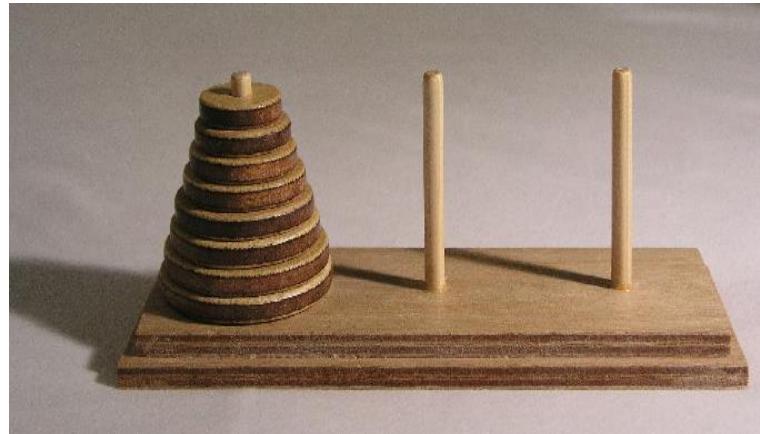
input: integer n such that n ≥ 0

1. if n is 0, **return** 0
2. if n is 1, **return** 1
3. otherwise, **return** [fib(n-1) + fib(n-2)]

end fib

The Towers of Hanoi

- It consists of three rods, and a number of disks of different sizes which can slide onto any rod.
- The puzzle starts with the disks neatly stacked in order of size on one rod, the smallest at the top, thus making a conical shape.
- The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:
 - Only one disk may be moved at a time.
 - Each move consists of taking the upper disk from one of the pegs and sliding it onto another rod, on top of the other disks that may already be present on that rod.
 - No disk may be placed on top of a smaller disk.



Assignment

- Write **C Program** uses recursive function & solves the **tower of Hanoi**, where number of disks is an input parameter.

Passing an array as an argument of a function

// Program to calculate the sum of array elements by passing to a function

```
#include <stdio.h>
float Sum_Func(float B[]);

int main(void) {
    float A[] = {2.5, 7.8, 2.6, 3, 4.5, 20};
    float sum1=0.0;
    // num array is passed to Sum()
    sum1 = Sum_Func(A);
    printf("Result = %.2f", sum1);
    return 0;
}

float Sum_Func(float K[]) {
    float sum = 0.0;
    for (int i = 0; i < 6; ++i) {
        sum += K[i];
    }
    return sum;
}
```

MA 511: Computer Programming

Lecture 11: Structure

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

structures

array: Data Structure whose elements are all of the same data type.

Example:

```
int A[10]; float B[10];
```

structure: individual elements can differ in type:

Example:

```
struct account {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
};
```

where `acct_no`, `acct_type`, `name[80]`, `balance` are the member of the **structure account**.

Contd.. **structures**

Example:

```
struct account {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
} oldcustomer, newcustomer;
```

- **oldcustomer, newcustomer** are the structure variable of type account.

Contd.. **structures**

Example:

```
struct account {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
};
```

- We also can declare the structure variable follows:
- **struct account oldcustomer, newcustomer;**
- oldcustomer, newcustomer are the structure variable of type account.

Contd.. **structures**

Example:

```
struct account {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
}customer[100];
```

- This declarations implies that **customer** is a **100** element array of the structures of the type **account**.

Example

how to access variables acct_no, acct_type, name, balance of the **struct account** ?

```
struct account {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
};
```

```
static/external struct account customer[ ] = {12, 'S', "abcd", 123.0,  
                                             13, 'C', "wert", 234.0,  
                                             14, 'S', "rsdef", 1234.0};
```

or

```
static/external struct account customer[ ] = {{12, 'S', "abcd", 123.0}, {13, 'C', "wert",  
234.0}, {14, 'S', "rsdef", 1234.0}};
```

static/external : storage class

Example

how to access variables acct_no, acct_type, name, balance of the **struct account** ?

```
#include<stdio.h>
void input_data(int i);
void output_data(int i);
struct account {
    int acct_no;
    char acct_type;
    char name[80];
    float balance;
} customer[100];
main(){
    int i, n;
    printf("No of Customers? ");
    scanf("%d", &n);
    for(i = 0; i < n; i = i + 1)
        input_data(i);
    for(i = 0; i < n; i = i + 1)
        output_data(i);
} //end of the main
```

```
void input_data(int i){
    printf("Name of the customer:");
    scanf(" %[^\n]", customer[i].name);
    printf("Acct_no of the customer :");
    scanf("%d", &customer[i].acct_no);
    printf("Balance of the customer :");
    scanf("%f", &customer[i].balance);
    printf("Type of the customer ");
    scanf(" %c", &customer[i].acct_type);
}

void output_data(int i){
    printf("\n Name of the customer: %s", customer[i].name);
    printf("\n Acct_no of the customer : %d", customer[i].acct_no);
    printf("\n Balance of the customer : %f", customer[i].balance);
    printf("\n Acct_type of the customer : %c", customer[i].acct_type);
}
```

Assignments

- Write a c-programming using **struct** for detecting a set of given n randomly generated points are belonging to a given circle* or not.

```
struct coordinates {
```

```
    int x;
```

```
    int y;
```

```
} points[100];
```

- *The input of the circle can be taken as center and radius.

Assignments

- Write a c-programming using **struct** for Semester Performance Index (SPI)

$$\text{SPI} = (U_1 G_1 + U_2 G_2 + \dots) / (U_1 + U_2 + \dots)$$

The grades awarded to student are G_1, G_2, \dots , etc in courses with corresponding credits U_1, U_2, \dots . Where G 's are from 10 to 4 and U 's are from 2 to 12.

- Write a program where input is an array of integer, then use a function for sorting the array, finally print the sorted array from main.

MA 511: Computer Programming

Lecture 12: Structure contd. and Union

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Example

how to access variables acct_no, acct_type, name, balance of the **struct account** ?

```
#include<stdio.h>
void input_data(int i);
void output_data(int i);
struct account {
    int acct_no;
    char acct_type;
    char name[80];
    float balance;
} customer[100];
main(){
    int i, n;
    printf("No of Customers? ");
    scanf("%d", &n);
    for(i = 0; i < n; i = i + 1)
        input_data(i);
    for(i = 0; i < n; i = i + 1)
        output_data(i);
} //end of the main
```

```
void input_data(int i){
    printf("Name of the customer:");
    scanf(" %[^\n]", customer[i].name);
    printf("Acct_no of the customer :");
    scanf("%d", &customer[i].acct_no);
    printf("Balance of the customer :");
    scanf("%f", &customer[i].balance);
    printf("Type of the customer ");
    scanf(" %c", &customer[i].acct_type);
}

void output_data(int i){
    printf("\n Name of the customer: %s", customer[i].name);
    printf("\n Acct_no of the customer : %d", customer[i].acct_no);
    printf("\n Balance of the customer : %f", customer[i].balance);
    printf("\n Acct_type of the customer : %c", customer[i].acct_type);
}
```

typedef

```
struct account {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
} customer[100];
```

int i, j; equivalent to

```
typedef int mydef;
```

```
mydef i, j;
```

```
typedef struct {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
} account ;
```

```
account customer[100], oldcustomer, newcustomer;
```

Member of a struct may be a struct

```
typedef struct {  
    int day;  
    int month;  
    int year;  
} date;  
  
typedef struct {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
    date update;  
} account;  
  
account customer[100];  
  
customer[i].acct_no: variable of the structure account  
customer[i].update.month: variable of the structure date
```

Equivalent to:

```
struct date {  
    int day;  
    int month;  
    int year;  
};  
  
struct account {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
    struct date update;  
};  
  
struct account customer[100];
```

Member of a struct may be a struct and so on

```
typedef struct {  
    int day;  
    int month;  
    int year;  
    TIME time;  
} date;  
  
typedef struct {  
    int acct_no;  
    char acct_type;  
    char name[80];  
    float balance;  
    date update;  
} account;  
account customer[100];
```

```
typedef struct {  
    int sec;  
    int min;  
    int hrs;  
} TIME;
```

customer[i].acct_no: variable of the **structure account**
customer[i].**update**.month: variable of the **structure date**
customer[i].**update.time**.min: variable of the **structure TIME**

Example

how to access variables acct_no, acct_type, name, balance of the **struct account** ?

```
#include<stdio.h>
void input_data(int i);
void output_data(int i);
struct account {
    int acct_no;
    char acct_type;
    char name[80];
    float balance;
} customer[100];
main(){
    int i, n;
    printf("No of Customers? ");
    scanf("%d", &n);
    for(i = 0; i < n; i = i + 1)
        input_data(i);
    for(i = 0; i < n; i = i + 1)
        output_data(i);
} //end of the main
```

```
void input_data(int i){
    printf("Name of the customer:");
    scanf(" %[^\n]", customer[i].name);
    printf("Acct_no of the customer :");
    scanf("%d", &customer[i].acct_no);
    printf("Balance of the customer :");
    scanf("%f", &customer[i].balance);
    printf("Type of the customer ");
    scanf(" %c", &customer[i].acct_type);
}

void output_data(int i){
    printf("\n Name of the customer: %s", customer[i].name);
    printf("\n Acct_no of the customer : %d", customer[i].acct_no);
    printf("\n Balance of the customer : %f", customer[i].balance);
    printf("\n Acct_type of the customer : %c", customer[i].acct_type);
}
```

Union

```
Union tag {
    member 1;
    ...
    member m;
};
```

```
Union account {
    int acct_no;
    char acct_type;
    char name[80];
    float balance;
};
```

- Like structures, contain members whose individual data types may differ from one another.
- union allocates the memory equal to the maximum memory required by the member of the union but structure allocates the memory equal to the total memory required by the members.
- In union, one block is used by all the member of the union but in case of structure, each member have their own memory space
- Union is useful for application where values need not be assigned to all of the members simultaneously.

MA 511: Computer Programming

Lecture 13: Pointer

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

pointers

- Is a **variable** that represents the **location** (**address**) of a data item.
- Each data item occupies one or more contiguous memory cells in computer memory.
- No of memory cells depends on the type of data item.
 - A single character needs 1 byte (8bits)
 - An integer usually needs 2 contiguous bytes
 - A floating point no needs 4 contiguous bytes
 - Double-precision quantity may needs 8 contiguous bytes

pointers

- Let v is a variable of some data item.
`float v;`
- Then data item can then be assessed if we know the location of *first* memory cell.
- `&v` = address of v' s memory location.

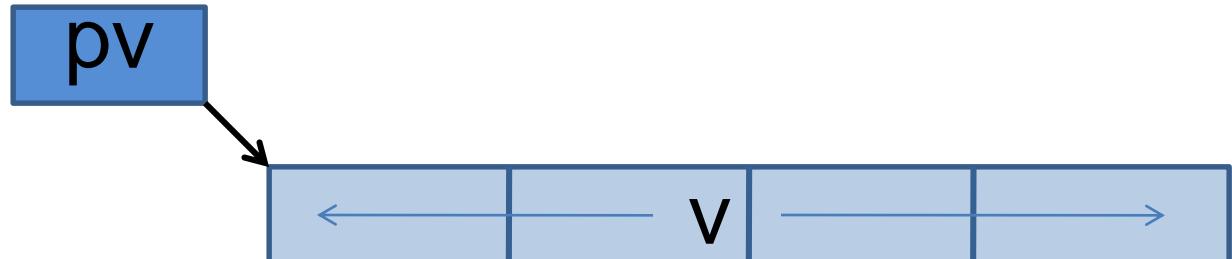
`pv = &v;` // & unary operator, *address operator*

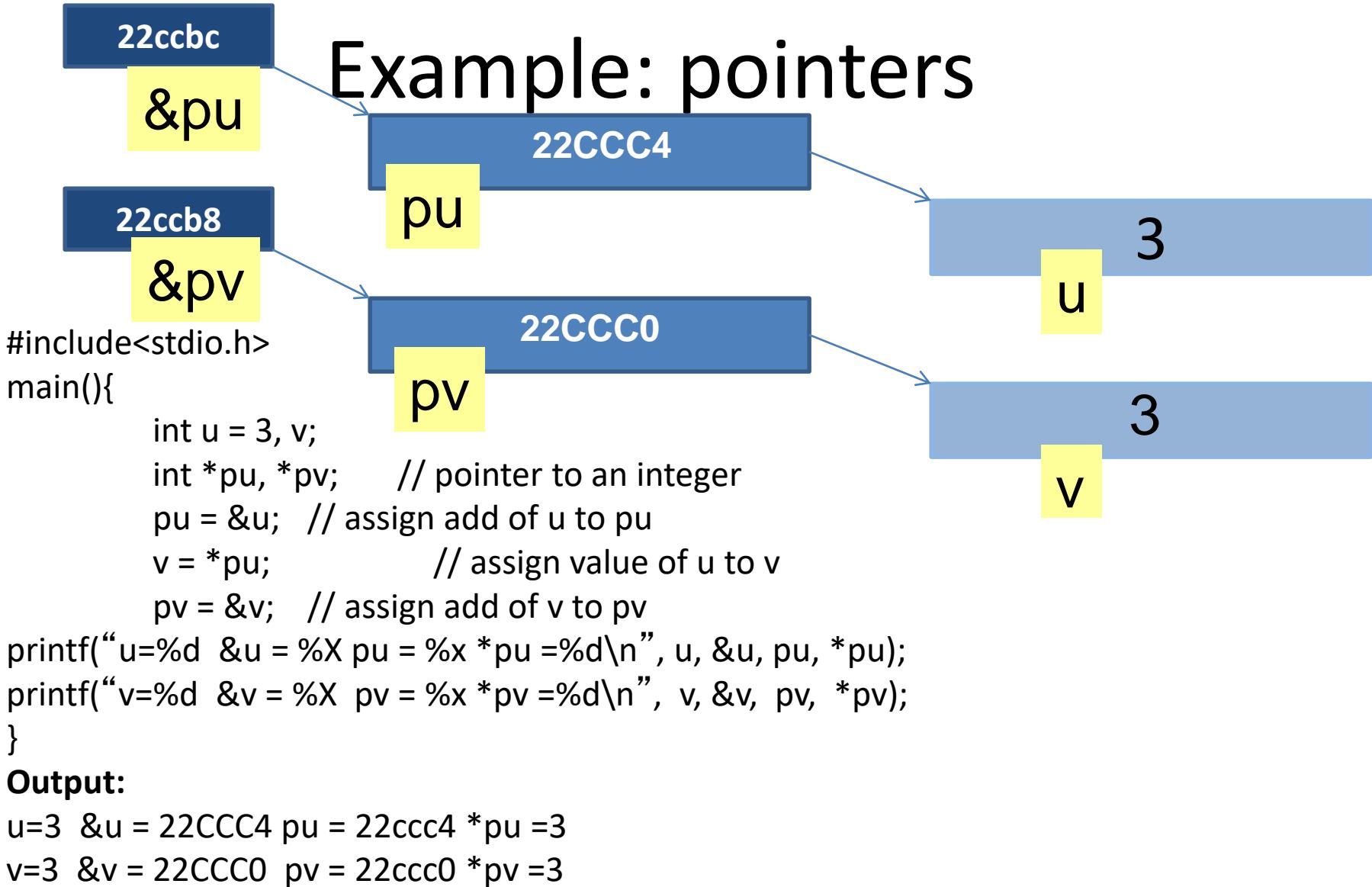
pv = pointer of the variable v

`v = *pv;` // * unary operator, *indirection operator*

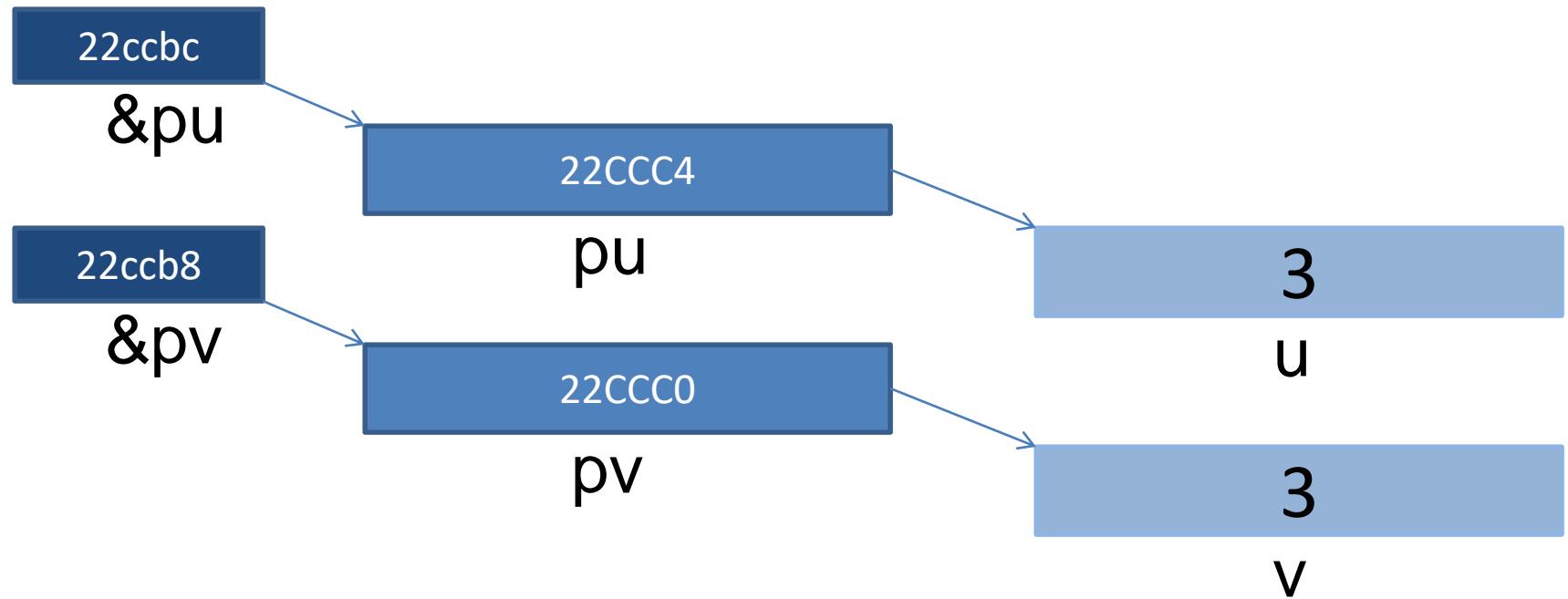
v, *pv = represent same data type.

Now if `pv = &v` and `u = *pv` then u and v represent the same value.





Example: pointers



Assignment

- Swap values of two variables using function and passing pointer as arguments.

Swap

```
/* swap values of two variables using function and passing pointer as arguments */
void swap(int *a, int *b){
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
main(){
    int x = 15, y = 24, *px, *py;
    px = &x; py = &y;
    printf("x= %d y= %d, px = %x, py = %x\n", x, y, px, py);
    swap(px, py);
    printf("x= %d y= %d, px = %x, py = %x\n", x, y, px, py);
}
```

Output:

x= 15 y= 24, px = 22ccc4, py = 22ccc0
x= 24 y= 15, px = 22ccc4, py = 22ccc0

Swap

```
/* swap values of two variables using function and passing pointer as arguments */
void swap(int *a, int *b){
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
main(){
    int x = 15, y = 24;
    printf("x= %d y= %d, &x = %x, &y = %x\n", x, y, &x, &y);
    swap(&x, &y);
    printf("x= %d y= %d, &x = %x, &y = %x\n", x, y, &x, &y);
}
```

Output:

```
x= 15 y= 24, &x = 22ccc4, &y = 22ccc0
x= 24 y= 15, &x = 22ccc4, &y = 22ccc0
```

MA 511: Computer Programming

Lecture 14: Pointer contd.

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Passing Arguments to Function

- Passing values (call by value)
- Passing pointers (call by reference)
- Passing a pointer to an array

Passing values

- Passing values is known as call by value.
- You actually pass a copy of the variable to the function.
- If the function modifies the copy, the original remains unaltered.
Following example demonstrated call by value.

```
int func(int m, int n){  
    int i;  
    i = m + n;  
    return i;  
}  
  
int main(void) {  
    int i=1, j;  
    printf("i before call the func %d", i);  
    j= func(i, i);  
    printf("i after func is executed %d and j = %d. \n", i, j);  
  
    return 0;  
}
```

Output:

i before call the func 1

i after func is executed 1 and j= 2.

Passing pointers

- This is known as **call by reference**.
- We do not pass the data to the function, instead we pass a **pointer** to the data.
- This means that if the function alters the data, the **original is altered**.
- Following example demonstrated **call by reference**.

Passing pointers

contd.

```
void func(int *p) {  
    ++*p; /* Add 1 to the value */  
    return;  
}  
  
int main(void) {  
    int i=4, *ptri;  
    ptri = &i;  
    printf("i before call the func %d\n", i);  
    printf(" *ptri is %d\n", *ptri);  
    func(ptri);  
    printf(" i after call the func %d\n", i);  
    return 0;  
}
```

Output:
i before call the func 4
*ptri is 4
i after call the func 5

Passing a pointer to an array

```
// how to access values of a array using pointers
#define SIZE_ARRAY 2
void func(int*); // Function declaration
Int main(void){
    int A[SIZE_ARRAY]={14, 16}; // array declaration
    int count=0;
    for (count=0; count<SIZE_ARRAY; count++)
        printf(" A before call the func %d ptri = %x\n", A[count], A+count);
    func(A); // Function call
    for (count=0; count<SIZE_ARRAY; count++)
        printf(" A after call the func %d. Ptri = %x\n", A[count], A+count);
    return 0;
}

void func(int *ptr) {
    ++*ptr; // Add 1 to the first element in the array
    ++*(ptr+1); // And the second element
}
```

Output:

```
i before call the func 14 ptri = 22ccc0
i before call the func 16 ptri = 22ccc4
i after call the func 15 Ptri = 22ccc0
i after call the func 17 Ptri = 22ccc4
```

MA 511: Computer Programming

Lecture 15: Pointer contd.

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

outline

- Passing a pointer to a character string
- Passing a pointer to a character
- Function returning a pointer
- Pass function pointer as parameter

Passing pointer to a character string

```
// how to access character of an array using pointers
void func(char *array_f){
    printf("%s %p\n", array_f, array_f);
    array_f +=4;           // Modify the pointer
    *array_f = 'x'; // Modify the data pointed to by 'array'
}
Int main(void){
    char array_s[10]="987654321";          // initialization
    func(array_s);                      // function call
    printf("%s, %p\n", array_s, array_s);
    return 0;
}
```

Output:

987654321 22ccb0
9876x4321, 22ccb0

Passing pointer to a character

```
// how to access and modify pointer to a char
void func(char *ptrc){
    printf("%c\n", *ptrc);
    *ptrc = 'x';           // Modify the data
}

int main(void){
    char achar = 'h';      // initialization
    func(&achar);         // function call
    printf("%c, %x\n", achar, &achar);

    return 0;
}
```

Output:
h
x, 22ccc7

Passing pointer to an array

Example 1:

```
void func(int *p);
int main(void){
    static int a[5]={10,20,30,40,50};
    func(a + 3);
    return 0;
}
void func(int *p){
    int i, sum = 0;
    for(i=0; i<2; i++)
        sum += *(p+i);
    printf("sum = %d", sum);
}
```

Example 2:

```
void func(int p[]);
int main(void){
    static int a[5]={10,20,30,40,50};
    func(a + 3);
    return 0;
}
void func(int p[ ]){
    int i, sum = 0;
    for(i=0; i<2; i++)
        sum += *(p+i);
    printf("sum = %d", sum);
}
```

Assignments

- Polynomial addition
- Polynomial multiplication

MA 511: Computer Programming

Lecture 16: Pointer Contd.

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

- A function can return a pointer
- Passing function to other function

- A function can return a pointer
- Program to pass function pointer as parameter
 - how to pass function itself as a parameter to another function.

A function can return a pointer

- C program allows to return an array from a function
- C program allows to return a pointer from a function with following prototype.

```
int * Function() {  
    . . .  
    return ptr;  
}
```

- Pointers to local variables become invalid when the function exits.
- So, we need to define the local variable as **static** variable for returning local

Example: A function can return a pointer

```
int *func(int *p);

main(){
    int a[5]={10,20,30,40,50};
    int *maxptr;
    maxptr = func(a);
    printf("max=%d", *maxptr);
}
```

```
int *func(int *p){
    int i, imax, max = 0;
    for(i=0; i<5; i++){
        if(*(p+i) > max){
            max =*(p+i);
            imax = i;
        }
    }
    return(p+imax);
}
```

\

Pass function pointer as parameter

- We can only pass variables as parameter to function
- We cannot pass function to another function as parameter. But, **we can pass function reference** to another function using function pointers.
- Using function pointer, we can store reference of a function and can pass it to another function as normal pointer variable.
- Finally in the function we can call function pointer as normal functions.

Example: Passing function to other function

```
float guest1(float x, float y);
float guest2(float x, float y);
float host(float (*pt) (float x, float y));
```

```
int main(){
    float x, y;
    x = host(guest1);
    printf("x = %f\n",x);
    y = host(guest2);
    printf("y = %f\n",y);
    return 0;
} /* end of main */
```

```
float guest1(float x, float y){
    return(x+y);
}
float guest2(float x, float y){
    return(x*y);
}
float host(float (*pt) (float x, float y)){
    float a=5.0, b=3.5, c;
    c=(*pt)(a,b);
    return(c);
}
```

MA 511: Computer Programming
Lecture 17: Pointer Contd.
Pointer & multidimensional arrays

Partha Sarathi Mandal

psm@iitg.ac.in

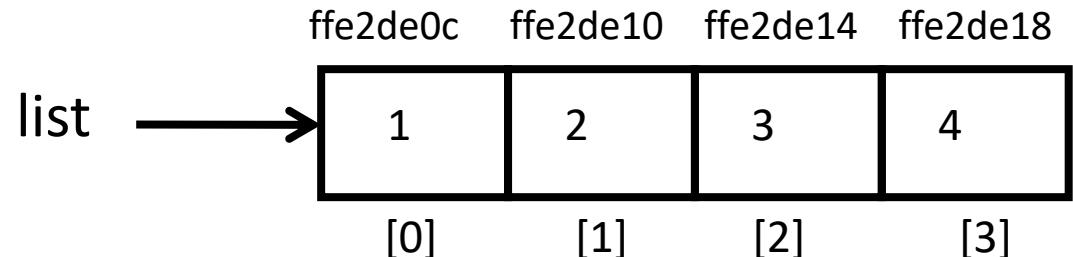
Dept. of Mathematics, IIT Guwahati

Pointers, Arrays, Multidimensional Arrays

- Pointers versus arrays
 - Lots of similarities
- How to deal with 2D, 3D, multidimensional arrays (for storing matrices and other 2D or 3D data!)

Array Name

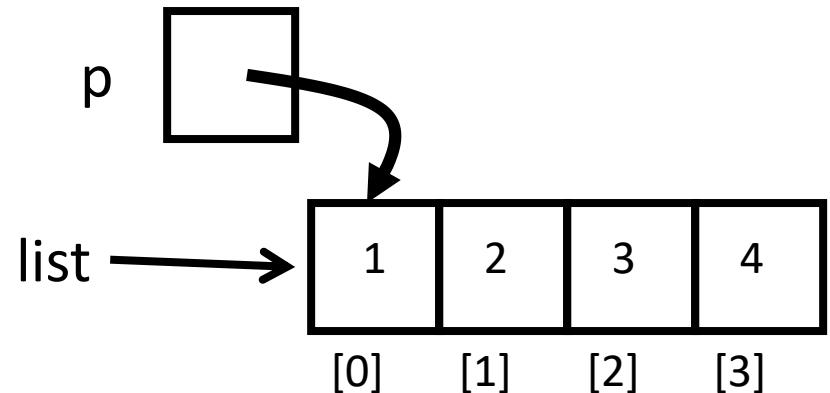
The array name is a pointer to the first element of the array



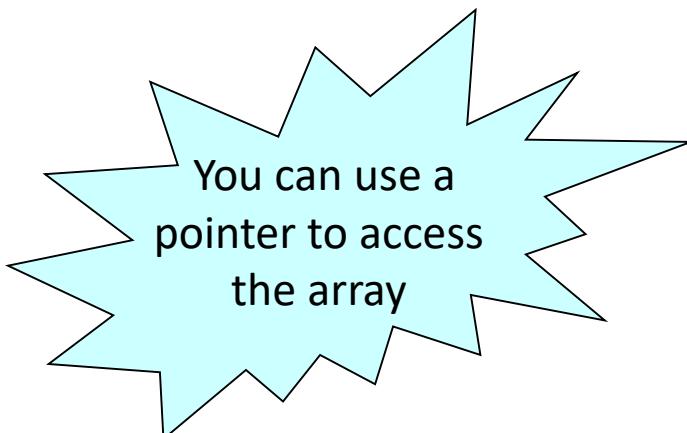
```
int list[]={1,2,3,4};  
printf("%x, %x, %d", list, &list[0], *list);
```

Output: ffe2de0c ffe2de0c 1

Pointers and Arrays

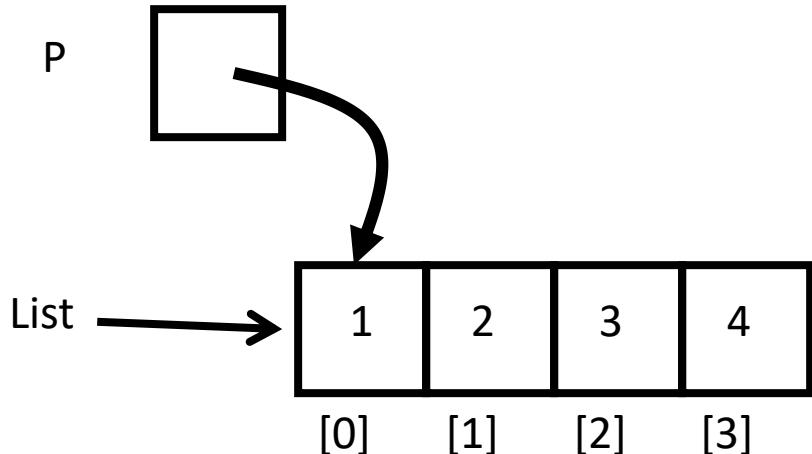


```
int *p,  
int list[]={1,2,3,4};  
p = list;          /* equivalent to p = &list[0] */  
printf("%d\n", *p); /* prints the value "1" */
```



Pointer and []

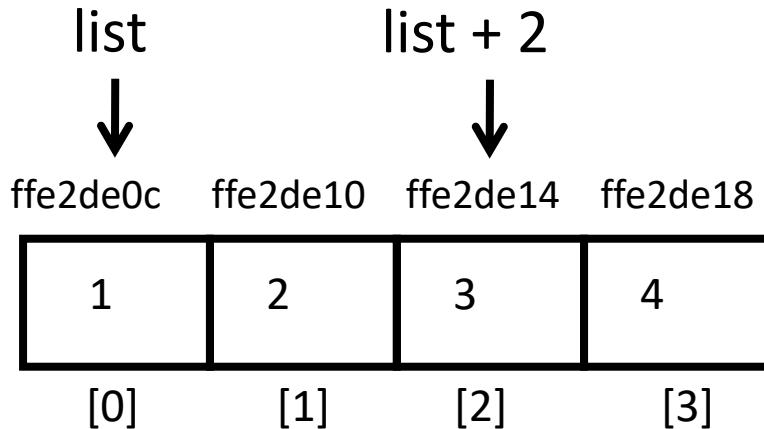
Any pointer to a block of memory can use the [] syntax, even if it is not declared as an array!



```
int *p,  
int list[]={1,2,3,4};  
p = list;  
printf("%d\n", p[2]); // prints 3
```

```
int *v; and int v[]; /* Mean the same thing */
```

Array indexing []



*list – Contents pointed to by list
*(list + 2) – Contents at list[2]

Indexing an array is just a way of finding a particular address in that block

```
int list[] = {1,2,3,4}          // array of 4 ints
    printf("%d", list[2]);
```

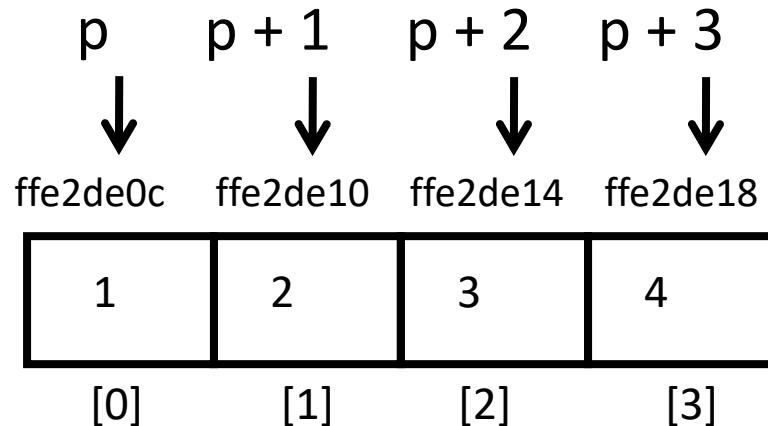
This is equivalent to

```
printf("%d", *(list+2));
```

Pointer Arithmetic

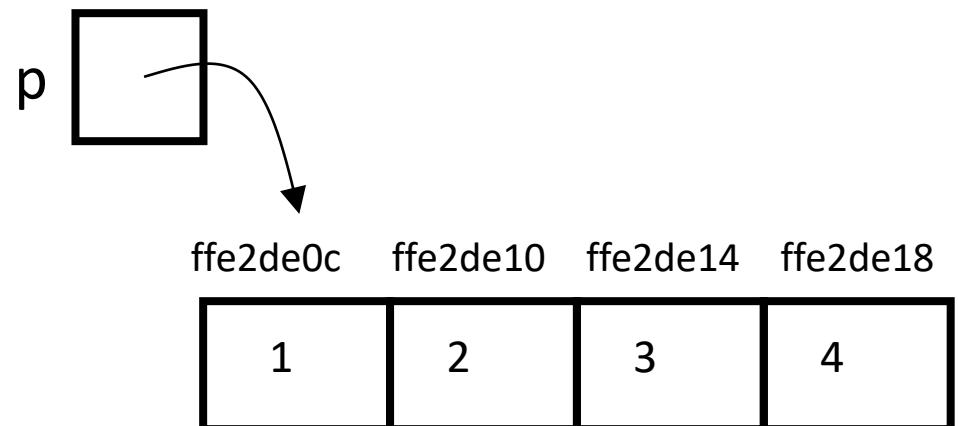
When we add to a pointer, such as $(p + 1)$, we don't literally add 1 to the pointer address

Instead we add one “address” to the pointer



Pointer Arithmetic

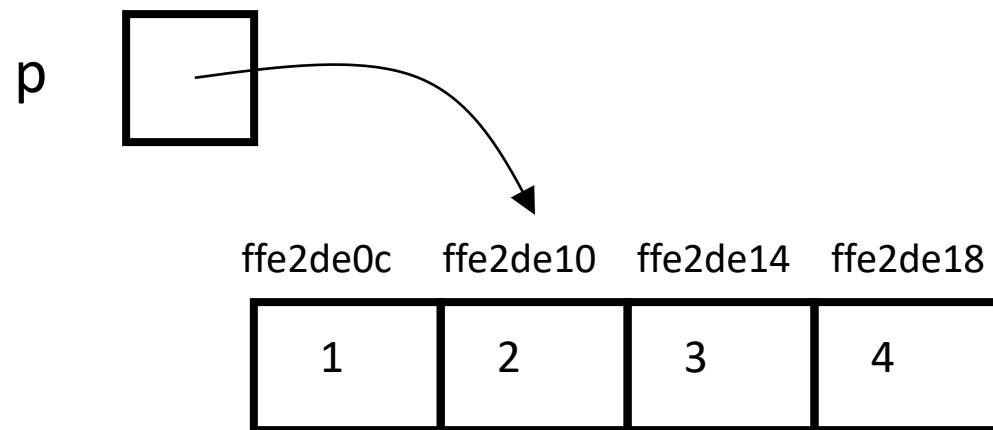
```
int list[] = {1, 2, 3, 4};  
int *p = list; /* same as p = &list[0] */  
printf("%x", p); /* prints ffe2de0c */
```



Pointer Arithmetic

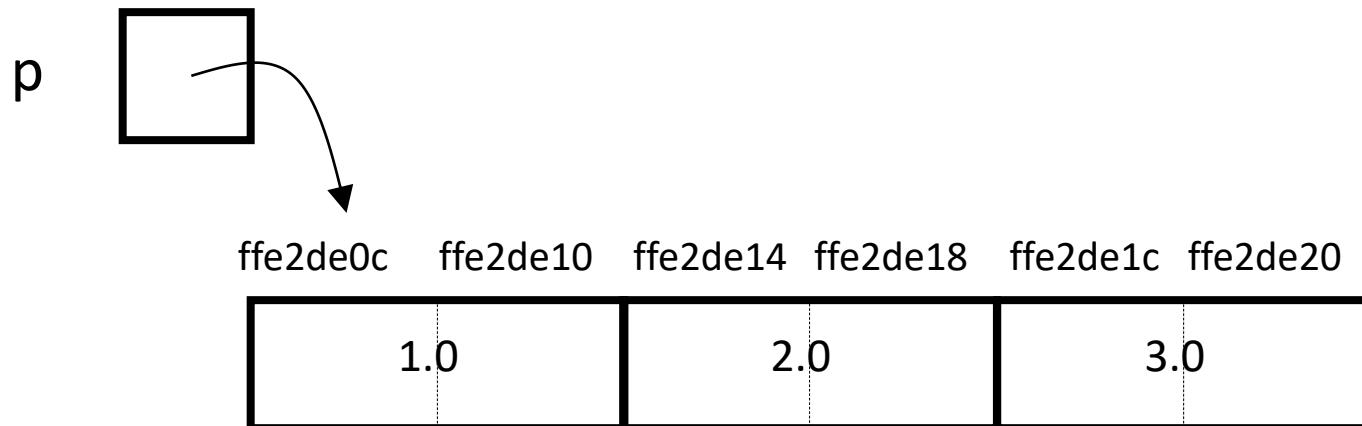
```
int list[] = {1, 2, 3, 4};  
int *p = list; /* same as p = &list[0] */  
printf("%x", p); /* prints ffe2de0c */  
p = p + 1; /* p increases by 4 */  
printf("%x", p); /* prints ffe2de10 */
```

Think of pointer arithmetic as add 1 “location” instead of one byte or address.



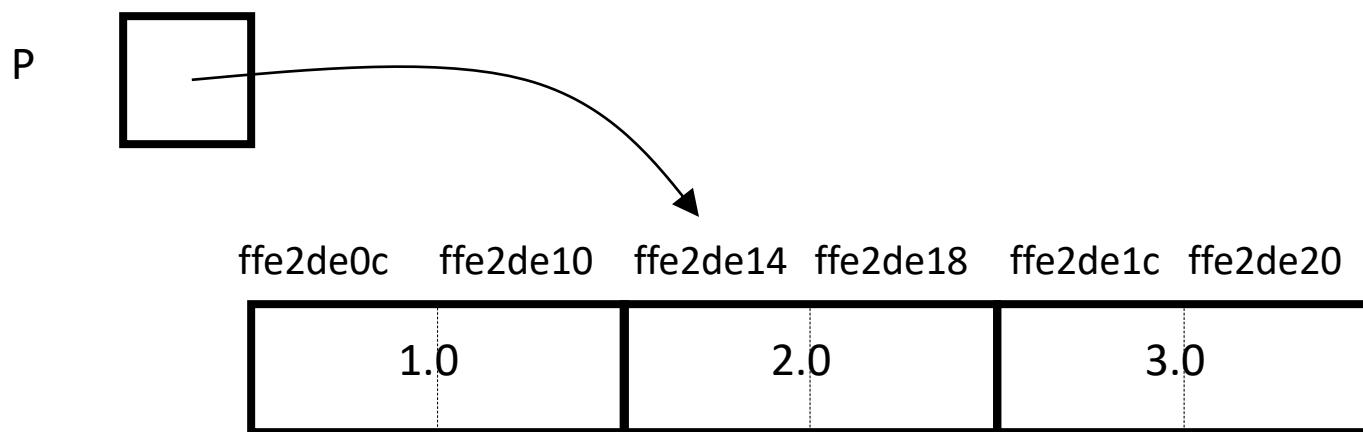
Pointer Arithmetic

```
double list2[] = {1.0, 2.0, 3.0};  
double *p = list2; /* same as p = &list2[0] */  
printf("%x", p); /* prints ffe2de0c */
```



Pointer Arithmetic

```
double list2[] = {1.0, 2.0, 3.0};  
double *p = list2; /* same as p = &list2[0] */  
printf("%x", p); /* prints ffe2de0c */  
p = p + 1; /* P increases by 8 bytes */  
printf("%x", p); /* prints ffe2de14 */
```



Pointer Arithmetic on Arrays

- `*(list+1)` references the next element in the array (equivalent to `list[1]`)
- Be careful: `*(++list)` works too but now we have lost our pointer to the beginning of the array!!!
 - Equivalent to: `list = list + 1; *list;`

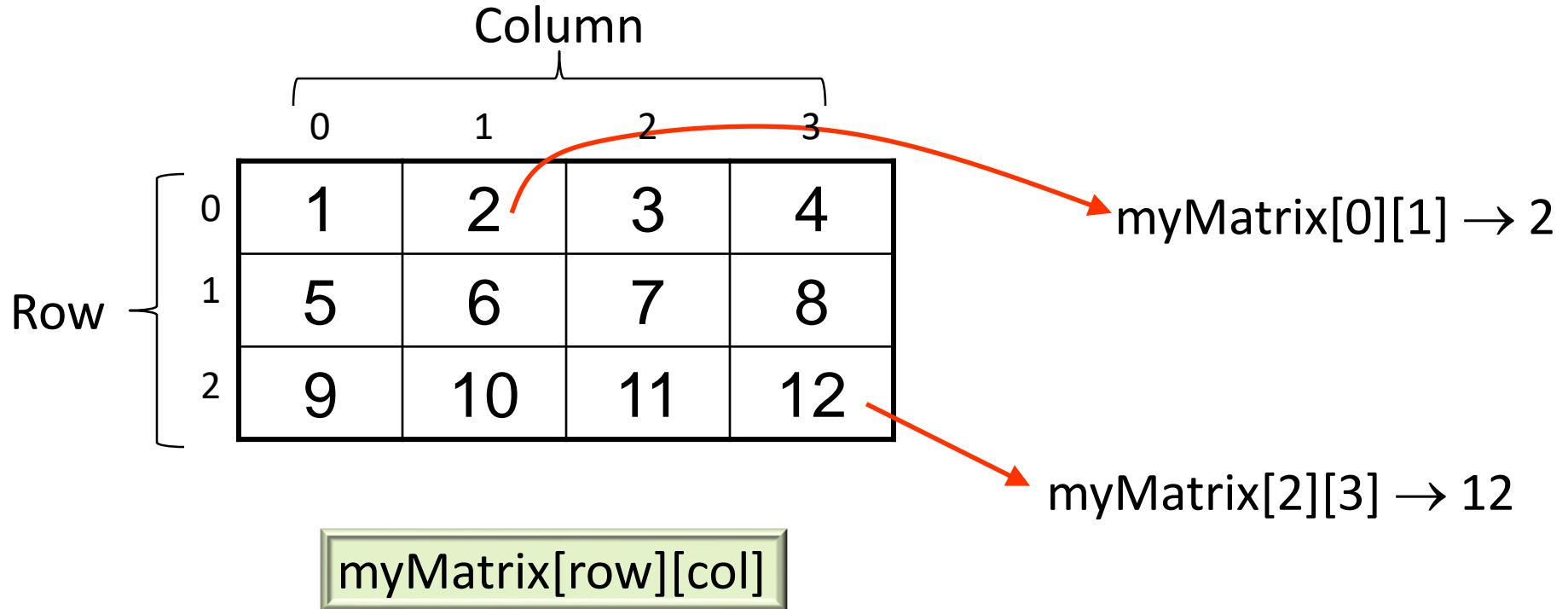
2D Arrays

```
int cave[ArraySize][ArraySize];
```

		Column				
		0	1	2	3	
Row		0	1	2	3	4
		1	5	6	7	8
2	9	10	11	12		
3	13	14	15	16		

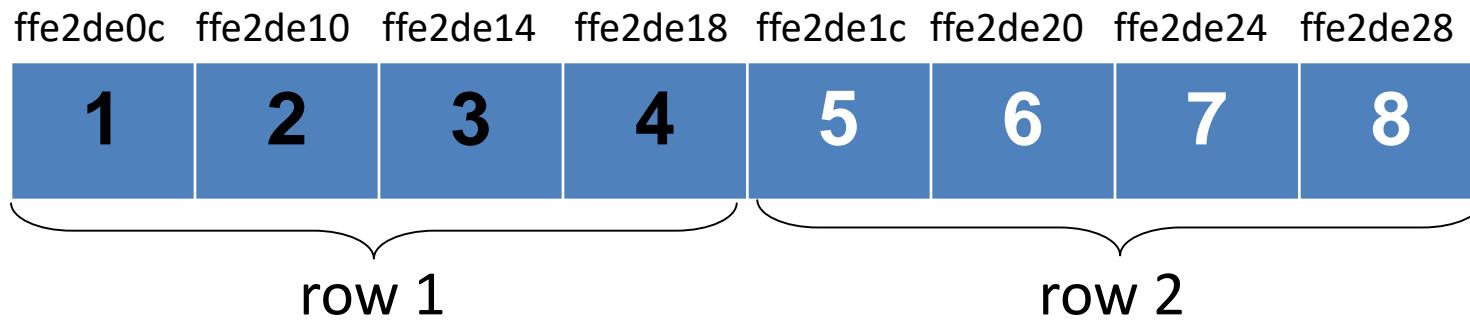
2D Arrays

```
int myMatrix[3][4] = { {1,2,3,4},{5,6,7,8},{9,10,11,12} };
```



Physically, in one block of memory

```
int myMatrix[2][4] = { {1,2,3,4},{5,6,7,8} };
```



Array elements are stored in *row major* order
Row 1 first, followed by row2, row3, and so on

2D Array Name and Addresses

```
int myMatrix[2][4] = { {1,2,3,4},{5,6,7,8} };
```

ffe2de0c	ffe2de10	ffe2de14	ffe2de18	ffe2de1c	ffe2de20	ffe2de24	ffe2de28
1	2	3	4	5	6	7	8

myMatrix: pointer to the first element of the 2D array

myMatrix[0]: pointer to the first row of the 2D array

myMatrix[1]: pointer to the second row of the 2D array

*myMatrix[1] is the address of element myMatrix[1][0]

Accessing 2D Array Elements

```
int myMatrix[2][4] = { {1,2,3,4} , {5,6,7,8} };
```

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Indexing: `myMatrix[i][j]` is same as

- *`(myMatrix[i] + j)`
- `(*(&myMatrix + i))[j]`
- `*(((*(&myMatrix + i)) + j)`
- `*(&myMatrix[0][0] + 4*i + j)`

Pointer & Multidimensional Array

x: 2D array having 10 rows 20 columns

declare as:

`int (*x)[20];` a ptr to a group of contiguous one dimensional, 20-element integer array.

which is same as

`int x[10][20];`

Similarly for 3D array:

`int (*b)[20][30];` a ptr to a group of contiguous two dimensional, 20 X 30 integer arrays.

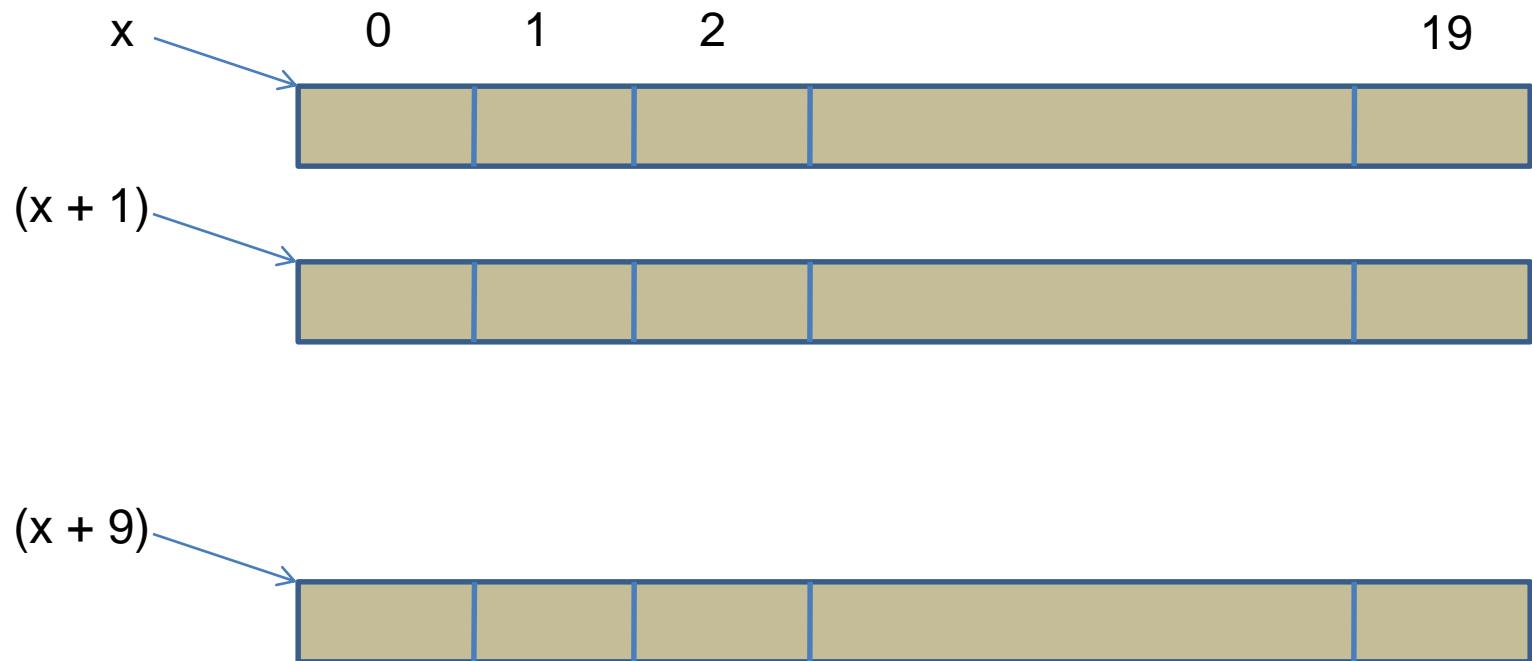
which is same as

`int b[10][20][30];`

Pointer & Multidimensional Array

`int (*x)[20];` a ptr to a group of contiguous one dimensional, 20-element integer array.

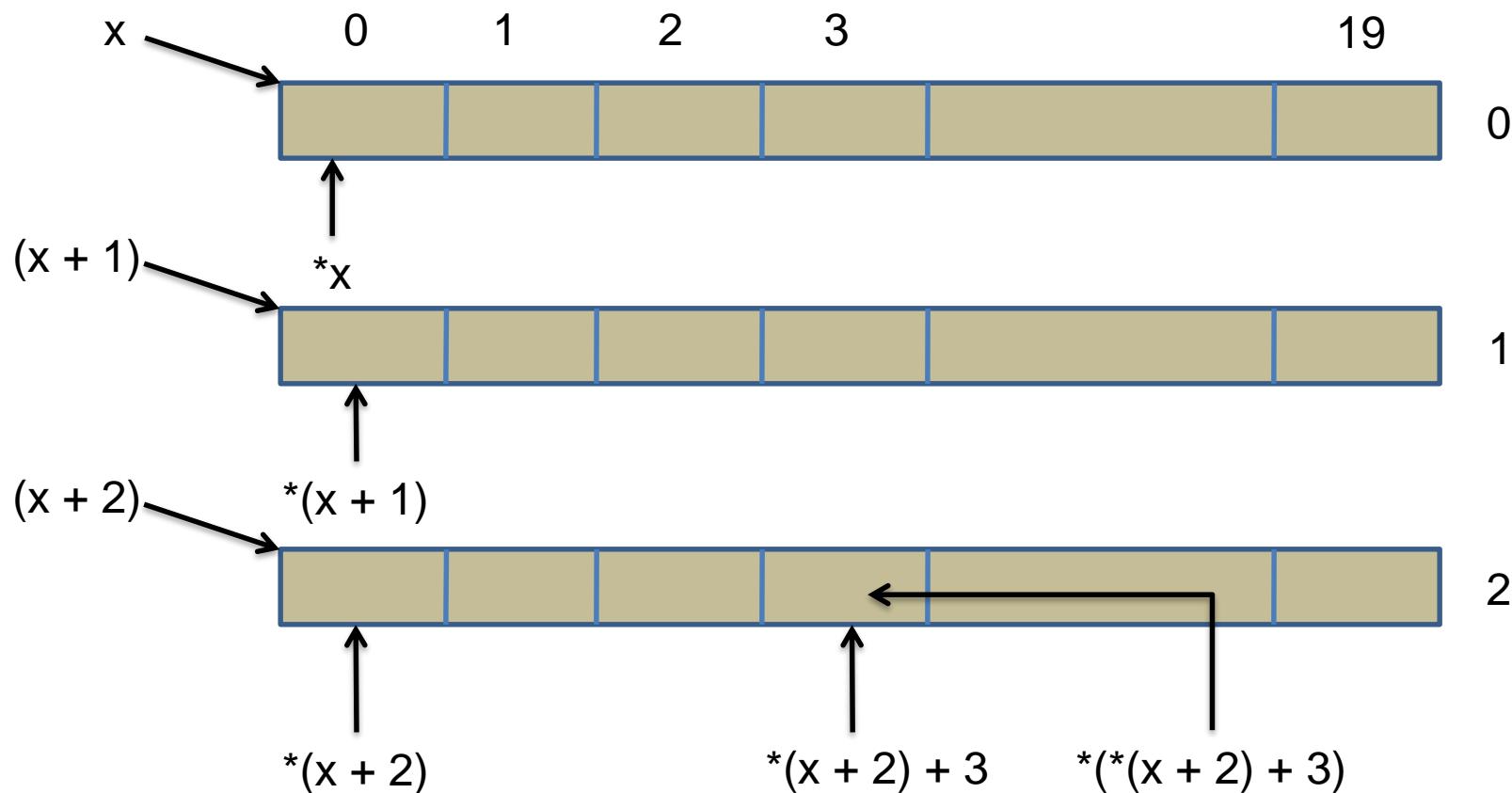
Or `int x[10][20];`



Pointer & Multidimensional Array

How to access the **item** in row 3 and column 4

$x[2][3]$ or $*(*x + 2) + 3$



Exercise

1. `int X[8] = {10, 12, 14, 15, 16, 17, 18, 19};` What is
 - a) `X`
 - b) `(x+2)`
 - c) `*x`
 - d) `*x+2`
 - e) `*(x+2) ?`

2. `float table[2][3] = { {1.1, 1.2, 1.3}, {2.1, 2.2, 2.3} };` What is
 - a) `table,`
 - b) `(table +1)`
 - c) `*(table +1)`
 - d) `(*table +1) + 1`
 - e) `(*table) +1)`
 - f) `*(*(table +1)+1`
 - g) `*(*(table +1))`
 - h) `*(*(table) +1) + 1?`

Dynamic memory allocation

```
int x[10];
```

Or

```
int *x;
```

```
x = (int *) malloc(10*sizeof(int));
```

- **malloc** library function reserves a block of memory size is equivalent to 10 integer for the above example.
- stdlib.h or **malloc.h** header file is required.
- **malloc** return a pointer to the beginning of the allocated space.
- in general **malloc(a)** allocate **a** bytes of memory.

Example: Dynamic memory in 1D

```
#include<stdio.h>
#include<stdlib.h>
main(){
    int *a, i, n;
    printf("Type n");
    scanf("%d",&n);
    a = (int *)malloc(n*sizeof(int));

    for(i=0; i<n; i++){
        printf("type %d entry ",i);
        scanf("%d", a+i);
    }
    for(i=0; i<n; i++){
        printf("%d", *(a+i));
    }
}
```

Example: Dynamic memory in 2D

```
#include<stdio.h>
#include<stdlab.h>
#define ROW 50
void read_ip(int *a[ROW], int nrow, int ncol);
void write_op(int *a[ROW], int nrow, int ncol);
main(){
    int *a[ROW], nrow, row;
    Printf("Type no of rows & cols");
    scanf("%d%d",&nrow, &ncol);
    for(row=0; row<=nrow; row++){
        a[row]=(int *) malloc(ncol*sizeof(int));
    }
    read_ip(a, nrow, ncol);
    write_op(a, nrow, ncol);
}
```

```
void read_ip(int *a[ROW], int m, int n){
    int row, col;
    for(row =0; row<m; row++)
        for(col =0; col<n; col++)
            scanf("%d", (a[row]+col));
}
void write_op(int *a[ROW], int m, int n){
    int row, col;
    for(row =0; row<m; row++){
        for(col =0; col<n; col++)
            printf("%4d", *(a[row]+col));
        printf("\n");
    }
}
```

MA 511: Computer Programming

Lecture 18:

Dynamic memory allocation

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Dynamic memory allocation

```
int x[10];
```

Or

```
int *x;
```

```
x = (int *) malloc(10*sizeof(int));
```

- **malloc** library function reserves a block of memory size is equivalent to 10 integer for the above example.
- stdlib.h or **malloc.h** header file is required.
- **malloc** returns a pointer to the beginning of the allocated space.
- in general **malloc(a)** allocate **a** bytes of memory.

Example: Dynamic memory in 1D

```
#include<stdio.h>
#include<stdlib.h>
main(){
    int *a, i, n;
    printf("Type n");
    scanf("%d",&n);
    a = (int *)malloc(n*sizeof(int));

    for(i=0; i<n; i++){
        printf("type %d entry ",i);
        scanf("%d", a+i);
    }
    for(i=0; i<n; i++){
        printf("%d", *(a+i));
    }
}
```

Example: Dynamic memory in 2D

```
#include<stdio.h>
#include<stdlab.h>
#define ROW 50
void read_ip(int *a[ROW], int nrow, int ncol);
void write_op(int *a[ROW], int nrow, int ncol);
main(){
    int *a[ROW], nrow, ncol, i;
    Printf("Type no of rows & cols");
    scanf("%d%d",&nrow, &ncol);
    for(i=0; i<=nrow; i++){
        a[i]=(int *) malloc(ncol*sizeof(int));
    }
    read_ip(a, nrow, ncol);
    write_op(a, nrow, ncol);
}
```

```
void read_ip(int *a[ROW], int m, int n){
    int row, col;
    for(row =0; row<m; row++)
        for(col =0; col<n; col++)
            scanf("%d", (a[row]+col));

}
void write_op(int *a[ROW], int m, int n){
    int row, col;
    for(row =0; row<m; row++){
        for(col =0; col<n; col++)
            printf("%4d", *(a[row]+col));
        printf("\n");
    }
}
```

MA 511: Computer Programming
Lecture 19: Pointer to structure and
passing structures to functions

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Structures and pointers

```
typedef struct {  
    int    acct_no;  
    char   acct_type;  
    char   name[80];  
    float  balance;  
} account;  
  
account customer, *ptr;  
ptr = &customer;
```

```
struct account{  
    int    acct_no;  
    char   acct_type;  
    char   name[80];  
    float  balance;  
}  
customer, *ptr;  
ptr = &customer;
```

Structures and pointers

```
typedef struct {  
    int      month;  
    int      day;  
    int      year;  
} date;
```

```
typedef struct {  
    int      acct_no;  
    char    acct_type;  
    char    name[80];  
    float   balance;  
    date    lastpayment;  
} customer, *ptr = &customer;
```

Example: Structures and pointers

```
main(){
```

```
    typedef struct {  
        int month;  
        int day;  
        int year;  
    } date;
```

```
    struct {  
        int acct_no;  
        char *acct_type;  
        char *name;  
        float balance;  
        date lastpayment;  
    } customer, *ptr = &customer;
```

```
    ptr->acct_no = 12341;           //customer.aact_no = 12341;  
    ptr->acct_type = "S/B";        //customer.acct_type ="S/B";  
    ptr->name = "xyz";            //customer.name = "xyz";  
    ptr->balance = 2345.00;         //...  
    ptr->lastpayment.month=02;  
    ptr->lastpayment.day=15;  
    ptr->lastpayment.year=2009;  
  
    printf("Account= %s Name=%s\n", ptr->acct_type, ptr->name);  
    printf("AccNo=%d, Balance=%f, Year=%d\n",  
          ptr->acct_no, ptr->balance, ptr->lastpayment.year );  
}
```

Account= S/B Name=xyz
AccNo=12341, Balance=2345.000000

Example: Structures and pointers

```
main(){

    typedef struct {
        int month;
        int day;
        int year;
    } date;

    typedef struct {
        int acct_no;
        char *acct_type;
        char *name;
        float balance;
        date lastpayment;
    } account;

    account customer, *ptr = &customer;

    ptr->acct_no = 12341;           //customer.aact_no = 12341;
    ptr->acct_type = "S/B";        //customer.acct_type ="S/B";
    ptr->name = "xyz";            //customer.name = "xyz";
    ptr->balance = 2345.00;        //...
    ptr->lastpayment.month=02;
    ptr->lastpayment.day=15;
    ptr->lastpayment.year=2009;

    printf("Account= %s Name=%s\n", ptr->acct_type, ptr->name);
    printf("AccNo=%d, Balance=%f, Year=%d\n",
           ptr->acct_no, ptr->balance, ptr->lastpayment.year );

}
```

Account= S/B Name=xyz
AccNo=12341, Balance=2345.000000

Passing structures to functions

```
typedef struct {  
    int acct_no;  
    char *acct_type;  
    char *name;  
    float balance;  
} account;  
  
void get_data(account *p){  
    p->acct_no = 55555;  
    p->acct_type = "C/A";  
    p->name = "abc";  
    p->balance = 777777.00  
}
```

```
main(){  
    account customer, *ptr;  
    ptr = &customer;  
  
    ptr->acct_no = 12341; //customer.aact_no = 12341;  
    ptr->acct_type = "S/B"; //customer.acct_type ="S/B";  
    ptr->name = "xyz"; //customer.name = "xyz";  
    ptr->balance = 2345.00; //...  
  
    printf("Account= %s Name=%s\n", ptr->acct_type, ptr->name);  
    printf("AccNo=%d, Balance=%f\n", ptr->acct_no, ptr->balance);  
  
    get_data(ptr); //get_data(&customer);  
  
    printf("Account= %s Name=%s\n", ptr->acct_type, ptr->name);  
    printf("AccNo=%d, Balance=%f\n", ptr->acct_no, ptr->balance);  
}  
  
Account= S/B Name=xyz  
AccNo=12341, Balance=2345.000000  
Account= C/A Name=abc  
AccNo=55555, Balance=777777.000000
```

Passing structures to functions

```
typedef struct {  
    int acct_no;  
    char acct_type[10];  
    char name[80];  
    float balance;  
} account;  
  
void get_data(account *p){  
    p->acct_no = 55555;  
    p->acct_type = "C/A";  
    p->name = "abc";  
    p->balance = 777777.00  
}
```

```
main(){  
    account customer, *ptr;  
    ptr = &customer;  
  
    ptr->acct_no = 12341; //customer.acct_no = 12341;  
    ptr->acct_type = "S/B"; //customer.acct_type ="S/B";  
    ptr->name = "xyz"; //customer.name = "xyz";  
    ptr->balance = 2345.00; //...  
  
    printf("Account= %s Name=%s\n", ptr->acct_type, ptr->name);  
    printf("AccNo=%d, Balance=%f\n", ptr->acct_no, ptr->balance);  
  
    get_data(ptr); //get_data(&customer);  
  
    printf("Account= %s Name=%s\n", ptr->acct_type, ptr->name);  
    printf("AccNo=%d, Balance=%f\n", ptr->acct_no, ptr->balance);  
}  
  
Account= S/B Name=xyz  
AccNo=12341, Balance=2345.000000  
Account= C/A Name=abc  
AccNo=55555, Balance=777777.000000
```

Passing structures to functions

```
# define N 2
typedef struct {
    int    acct_no;
    char   *acct_type;
    char   *name;
    float  balance;
} account;

account *get_data(account *p){
    p->acct_no = 55555;
    p->acct_type = "C/A";
    p->name = "abc";
    p->balance = 777777.00;
    return(p)
}
```

```
main(){
    account customer[N], *ptr, *ptr1;
    ptr = customer, ptr1=customer;
    int i;
    for(i=0; i<N; i++){
        ptr1=get_data(ptr+i); //get_data(customer+i);

        printf("Account= %s Name=%s\n", ptr1->acct_type,
               ptr1->name);
        printf("AccNo=%d, Balance=%f\n", ptr1->acct_no,
               ptr1->balance);
    }
}
```

```
Account= C/A Name=abc
AccNo=55555, Balance=777777.000000
Account= C/A Name=abc
AccNo=55555, Balance=777777.000000
```

Passing structures to functions

```
# define N 2
typedef struct {
    int acct_no;
    char acct_type[10];
    char name[80];
    float balance;
} account;

account *get_data(account *p){
    p->acct_no = 55555;
    p->acct_type = "C/A";
    p->name = "abc";
    p->balance = 777777.00;
    return(p)
}
```

```
main(){
    account customer[N], *ptr, *ptr1;
    ptr = customer, ptr1=customer;
    int i;
    for(i=0; i<N; i++){
        ptr1=get_data(ptr+i); //get_data(customer+i);

        printf("Account= %s Name=%s\n", ptr1->acct_type,
               ptr1->name);
        printf("AccNo=%d, Balance=%f\n", ptr1->acct_no,
               ptr1->balance);
    }
}
```

```
Account= C/A Name=abc
AccNo=55555, Balance=777777.000000
Account= C/A Name=abc
AccNo=55555, Balance=777777.000000
```

MA 511: Computer Programming

Lecture 20: Linked list

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Self-Reference Structure

```
struct tag {  
    member 1;  
    member 1;  
    ....  
    struct tag *name;  
};
```

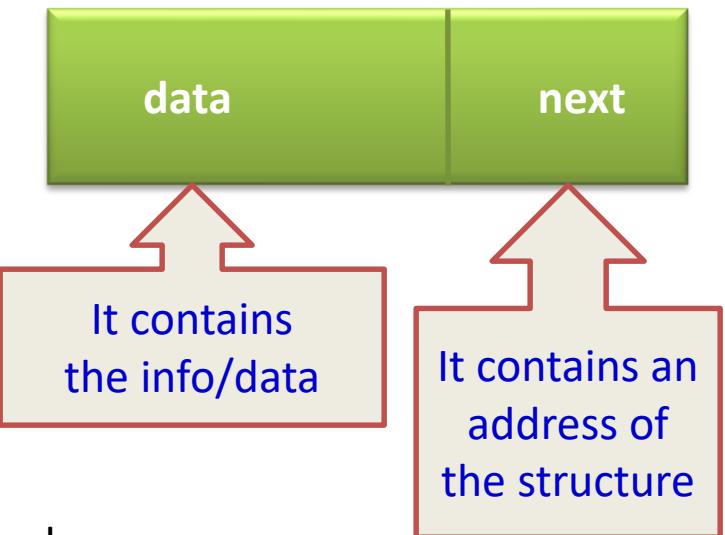
Example:

```
struct list_of_no {  
    int data;  
    struct list_of_no *next;  
};
```

It's a structure of type *list_of_no*. there are two members;

1. **data** is a **integer** type and
2. **next** is a **pointer to a structure** of the same type, i.e., a pointer to a structure of type *list_of_no*.

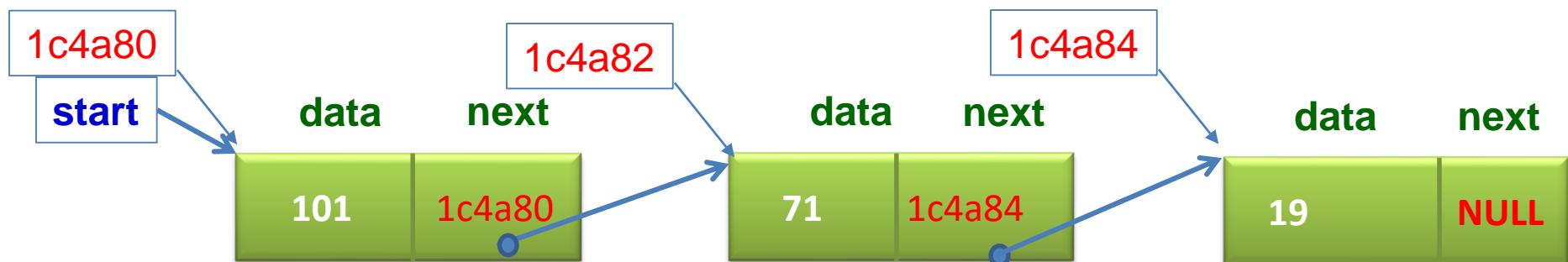
list_of_no structure looks like



Linked Lists using Self-Reference Structure

```
struct list_of_no {  
    int data;  
    struct list_of_no *next;  
};
```

```
typedef struct list_of_no node;  
node *start;  
start = (node *) malloc(sizeof(node));
```

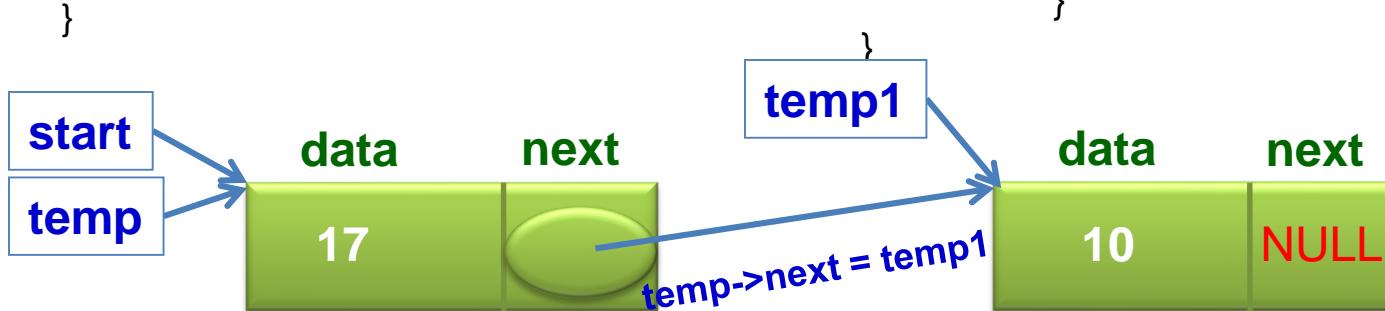


How to create a linked list recursively?

```
#include<stdio.h>
#include<stdlib.h>
struct list_of_no {
    int data;
    struct list_of_no *next;
};
typedef struct list_of_no node;
void create(node *pt);
void print(node *pt);

int main(){
    node *start;
    start = (node *) malloc(sizeof(node));
    printf("Type int data for 1st node: ");
    scanf("%d", &(start->data));
    start->next = NULL;
    create(start);
    print(start);
}
```

```
void create(node *temp){
    node *temp1;
    char val;
    printf("Type Y for continue N for stop:");
    scanf(" %c", &val);
    if(val=='Y') {
        temp1 = (node *) malloc(sizeof(node));
        printf("Type int data for next node: ");
        scanf("%d", &(temp1->data));
        temp1->next = NULL;
        temp->next = temp1;
        create(temp1);
    }
}
void print(node *temp){
    while(temp){
        printf("|%d |->", temp->data);
        temp=temp->next;
    }
}
```



Storage class

- **auto:** default storage class for **local variables**.
`auto int Month;`
- **register:** **local variables** that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).
`register int Miles;`
- **static:** default storage class for **global variables**, it can also be defined within a function. If this is done, the variable is initialized at compilation time and retains its value between calls.
- **extern:** **global variable** that is visible to ALL object modules. When you use 'extern' the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

Dear Students,

First of all we welcome you all to our department computer lab and will describe you the initial steps for operating lab computers.

We are using Ubuntu Linux (present version 16.04) as Operating System on all PCs of our Computer lab. Some of you may not be familiar with Linux operating system, so I am starting with how to login an Ubuntu system.

A. The Desktop

After powering on the system, within few seconds you will get the login screen looks like the image below -

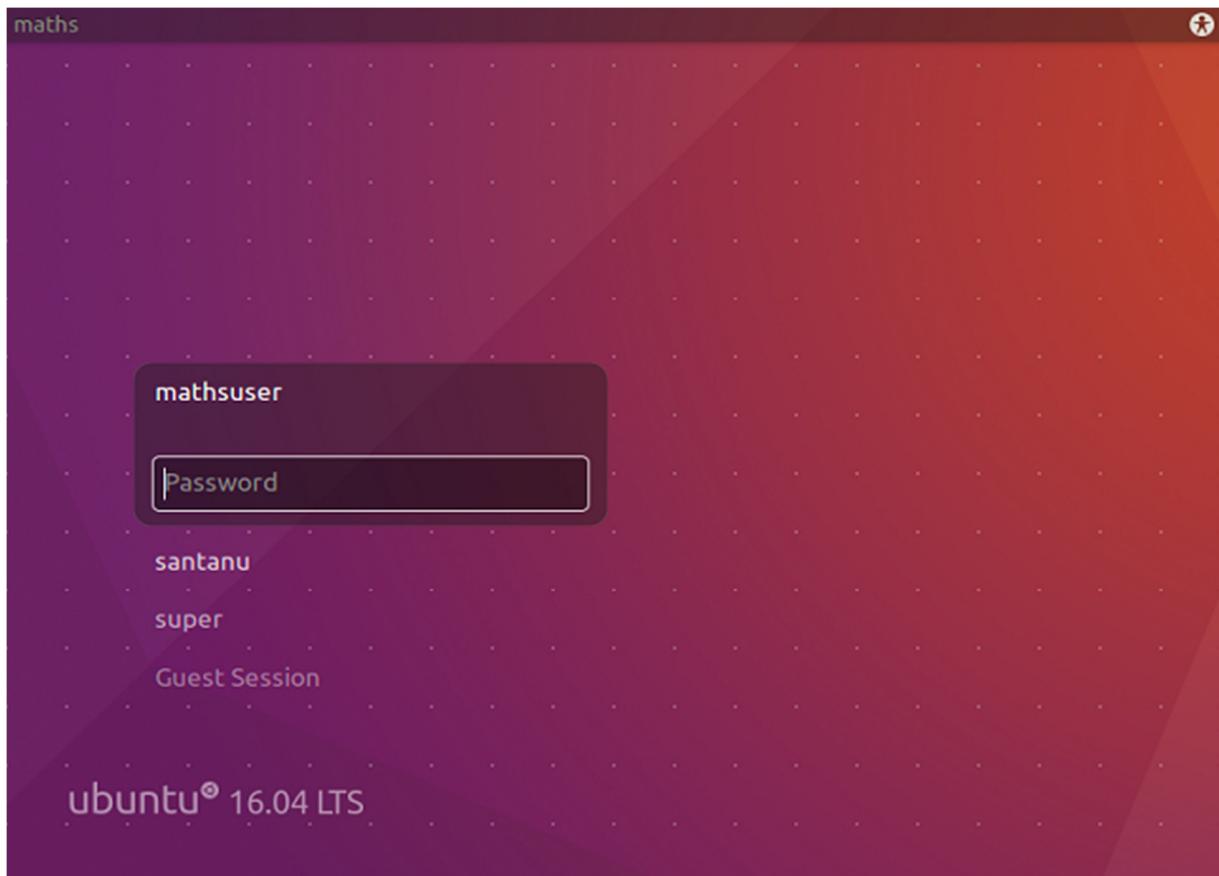


Figure 1: Login screen of Ubuntu 16.04

You have to choose your **USERNAME (WILL PROVIDE BY DEPARTMENT)** and type **PASSWORD (WILL PROVIDE BY DEPARTMENT)** and press enter. You will get the screen like below –

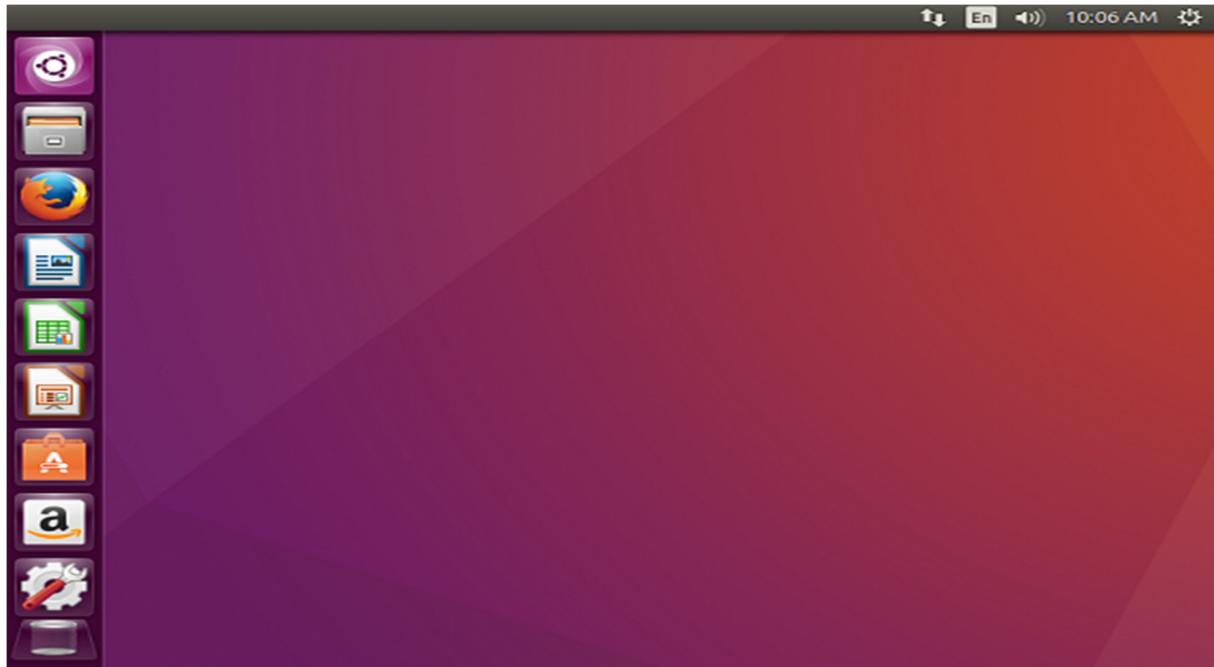


Figure 2. Ubuntu 16.04 Desktop

B. Opening the terminal (A Linux console terminal is one of the system consoles provided in the Linux kernel. The Linux console terminal acts as the medium for input and output operations for a Linux system. A Linux console terminal is similar to command line in Microsoft Windows but it differs in that it can perform any operation on the system)

Click on Dash icon (the very first icon (top left) on the sidebar and on search bar type **terminal**. Then click on the Terminal icon.

(Or use shortcut key combination - **Ctrl+Alt+t**)

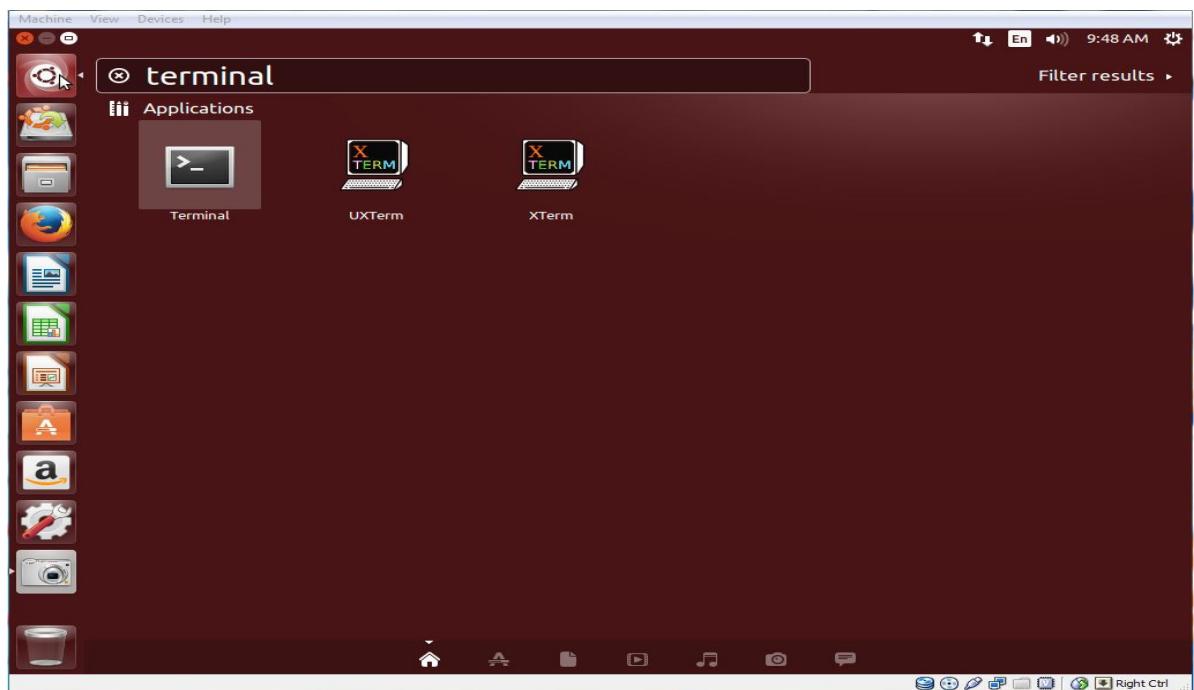


Figure 3: How to open a terminal

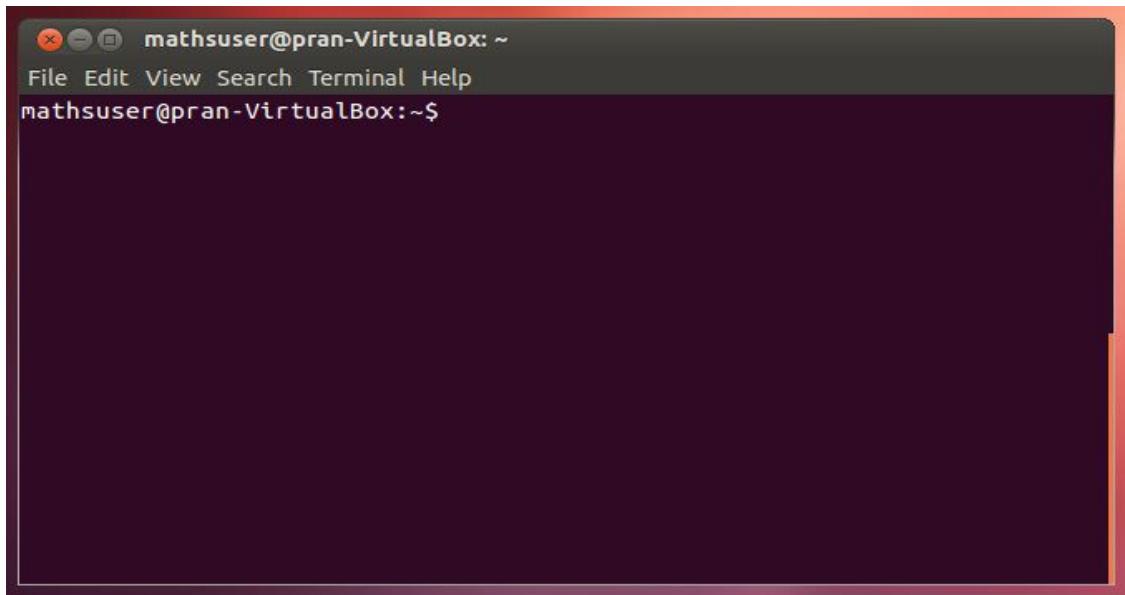


Figure 4: The terminal

C. Login to server using ssh from terminal

For storing your files, Department has provided you a server with 2GB of space. You can logon to the server from terminal. Login to the server is necessary for coding programs on server, printing purpose, copying and storing files on server etc.

On the terminal write the command – **ssh -X username@172.16.70.1 (X in UPPERCASE)**

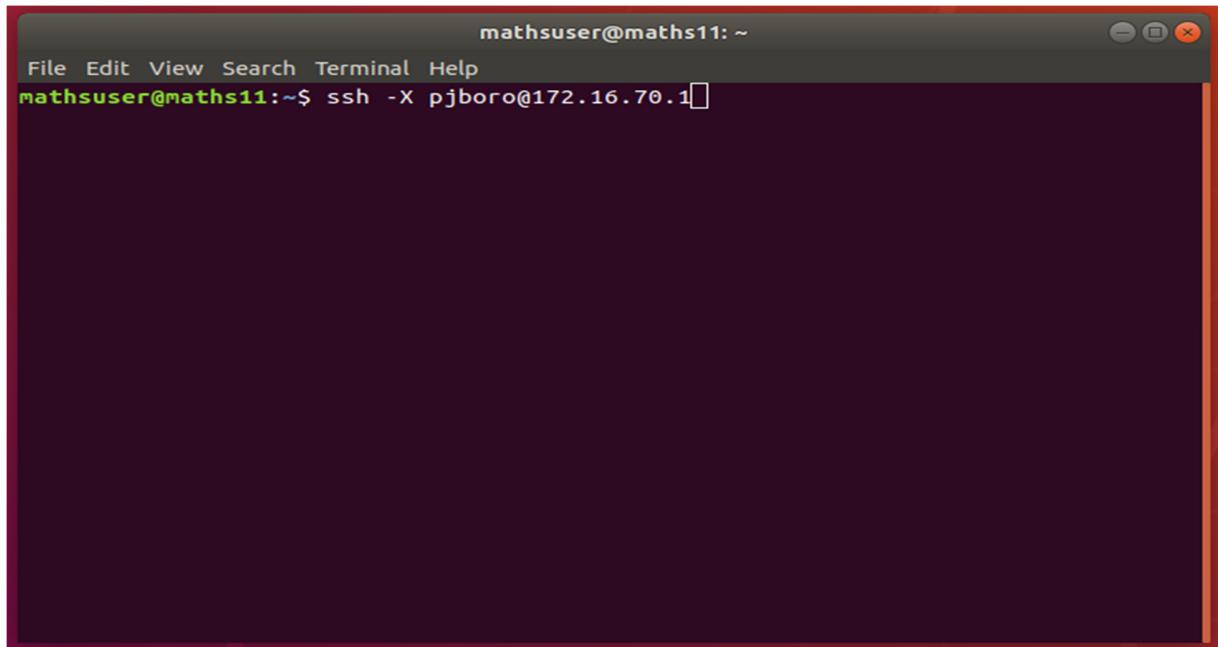


Figure 5: Login in to server from terminal.

(Note: you will get the username and password from lab administrator, but it should be provided only after the email –id provided by Computer center)

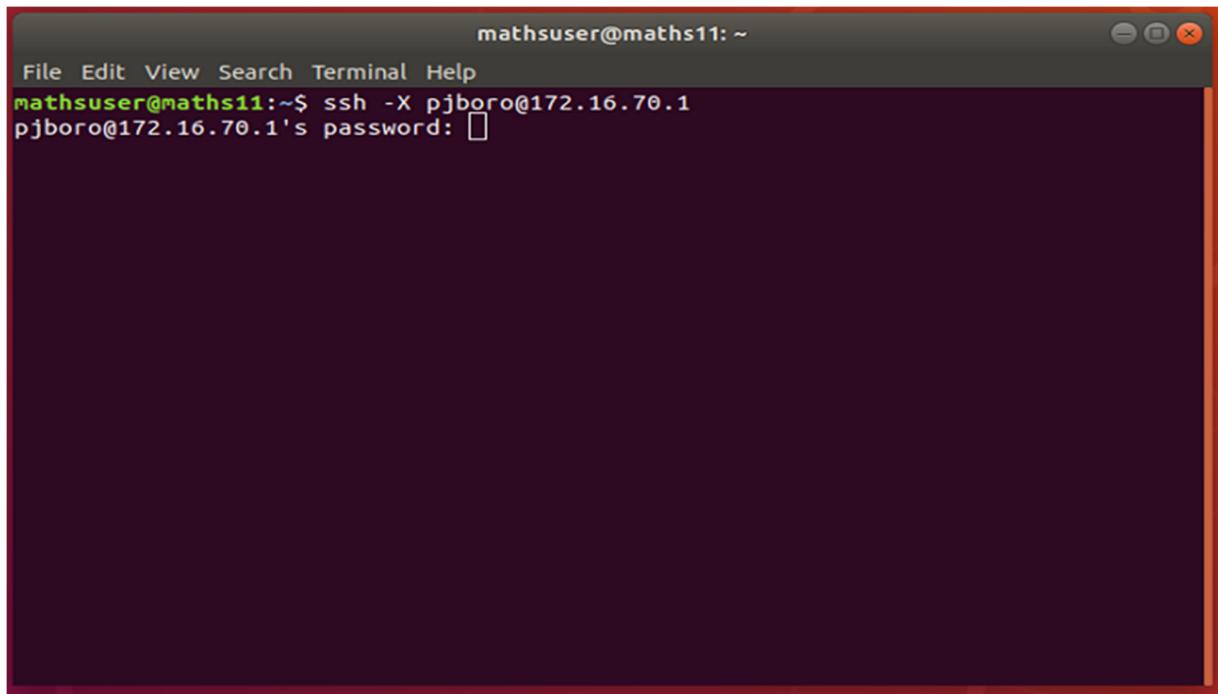


Figure 6: Logged in to server from terminal

After providing the password (1) (you will not see anything written on the screen when you type the password) press enter and you will get the prompt from the server to confirm your password.

Some simple commands are –

\$ vim filename – to create a text file or view / edit existing file

\$ mkdir foldername – to create a folder or directory

\$ cd foldername – to change to a directory

\$ cp source_filename path (path will be destination folder) – copy a file

\$ mv source_filename path (path will be destination folder) – move a file

\$ rm filename – delete a file

\$ ls – to list the files and folder on the current folder

\$ pwd – know your current directory

Transferring files to and from the Server using GFTP.

Opening gFTP client application. To open gFTP, click on Dash icon and type gftp and click on gftp icon.

Here you have to provide Host as **172.16.70.1** (1), Port as 22 (2), User as <USERNAME> (3), password will provide by department (4) and SSH2(5) as server type. After that you have to press Connect button (6).

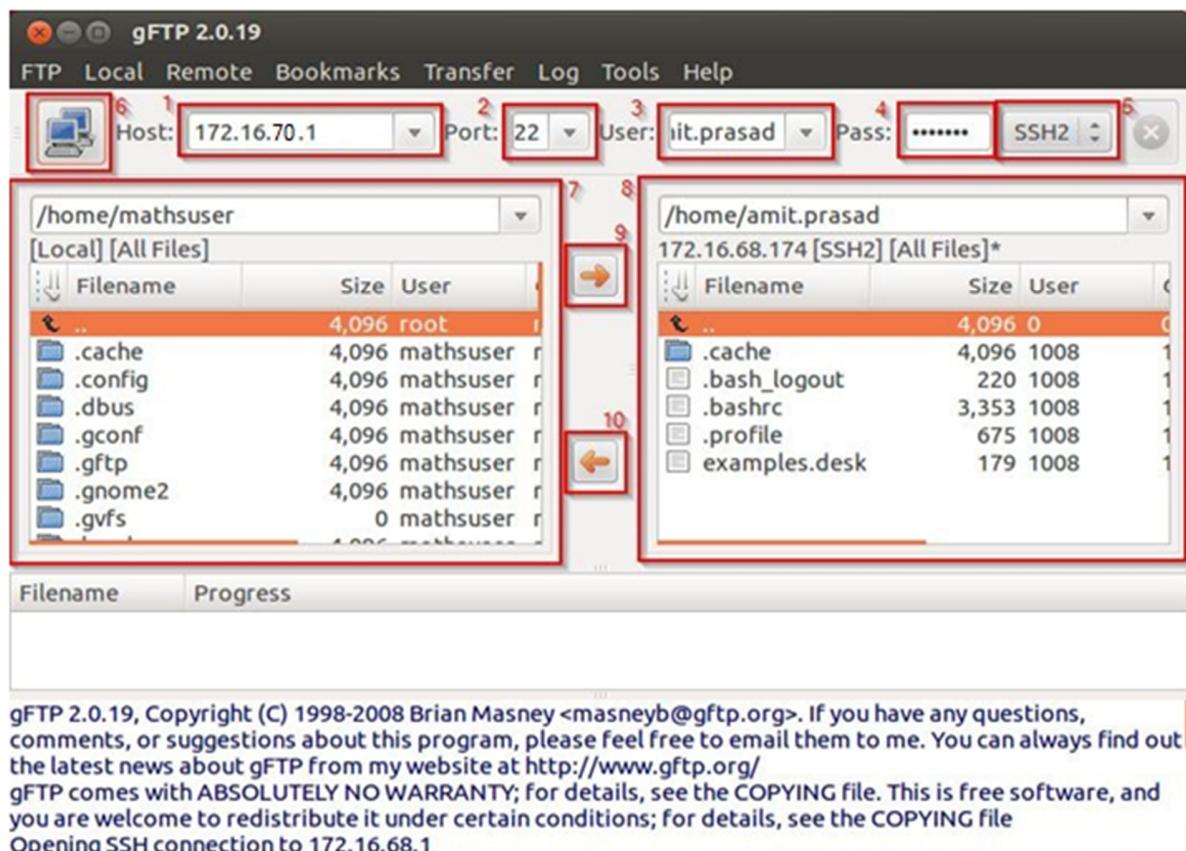


Figure 7: gFTP window after logged in to the server

Here local machine home directory files are already available on the left side browser (7). After login you will get the server/remote machine data on the right side browser (8). There are two arrow buttons, to transfer files from local machine file remote server and vise-versa. Select the file on local machine you want to transfer to server, and click on the (9) arrow button and select file from server to desktop select from right side browser (8) and click the (10) arrow button. Click on the connect button (6) again to disconnect from the server.

Editors for writing programs

There are several editors available in ubuntu like vi or vim, nano, gedit etc. The native is vi editor. To open vi or vim editor, open terminal and type vi <filename> eg. vim test.cpp

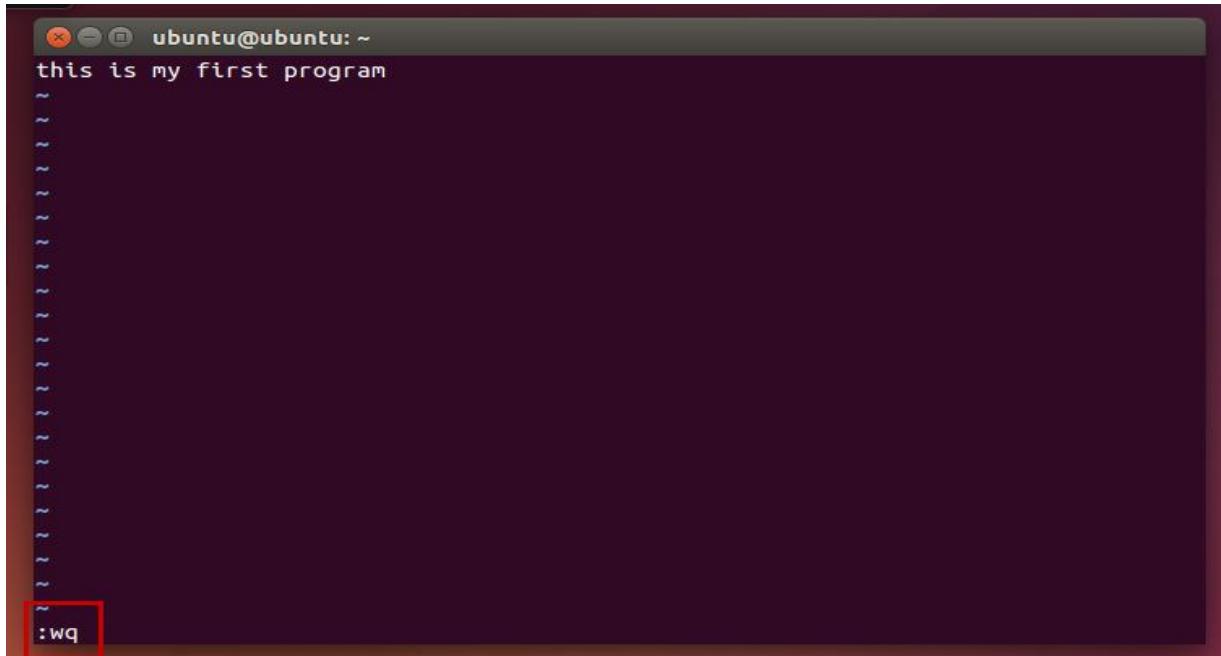


Figure 8: VI editor

This vi editor is little bit hard for new user. But it is one of the most powerful editors. There are actually 3 modes.

1. Command mode
2. Insert mode
3. Last line mode

When starts it is in command mode, where it accepts commands. Eg. you will not be able to write first. To write something you have to go to insert mode, for which you have to first press “i” to go to insert mode. After that you can write.

Now to save the program you have to go to first command mode, by pressing Esc and type “:”, which goes to last line mode than the command wq (wq means write and quite).

For more help type man vi for more help.

Note: man is a command to view help/manual. So, man vi will show all help option for vi editor.

Other Editors are like nano. For nano, open terminal type nano <filename>

For gedit, click on Dash icon and type gedit. This g edit is little bin windows notepad.

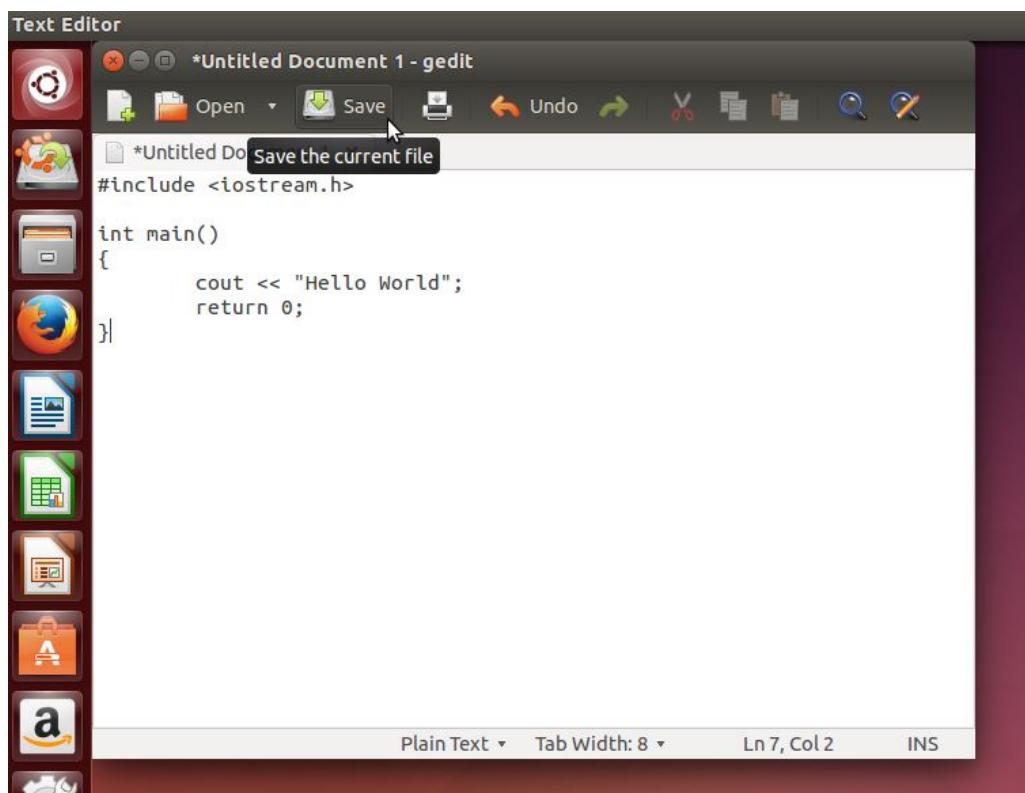


Figure 9: gedit

After writing a program you can click on save button to save the file.

D. Print quota and printing guide

Students are restricted with print quota. B.Tech and M.Sc. students will get 100 nos. of pages per semester and if very much necessarily need extra pages, he/she may get extra page print with approval from his/her guide. Students can't directly give printout from lab PCs. He must transfer his file to the server and then provide print command. The Details Instructions as given below –

Printing guideline

- For Printing file should be in PDF/PS format. So Convert the file into PDF or PS
- You have to transfer the files to the SSH server (172.16.70.1) required to print (Printing is only allowable from the ssh server)
- Now open a terminal and type the following to login to the server
 - ssh -X username@172.16.70.1 (X in UPPERCASE)
 - Type the password when prompted
 - Type **nautilus** to open the file browser
- Go to that location you have copied the file required to print
- Open the file and click on **File – Print** and select **designated printer** as printer name

hp3010: Mathematics Old Computer Lab (Ground Floor, E-Block)

rs3005: Research Scholars Lab (First Floor, E-Block)

rs3010e1: Research Scholars Lab (First Floor, E1-Block)

hp3005: Mathematics Old Computer Lab (Ground Floor, E-Block)

e1lab: Mathematics New Computer Lab (Ground Floor, E1-Block)

hp506rslab3: Research Scholars Lab (Third Floor, E1-Block)

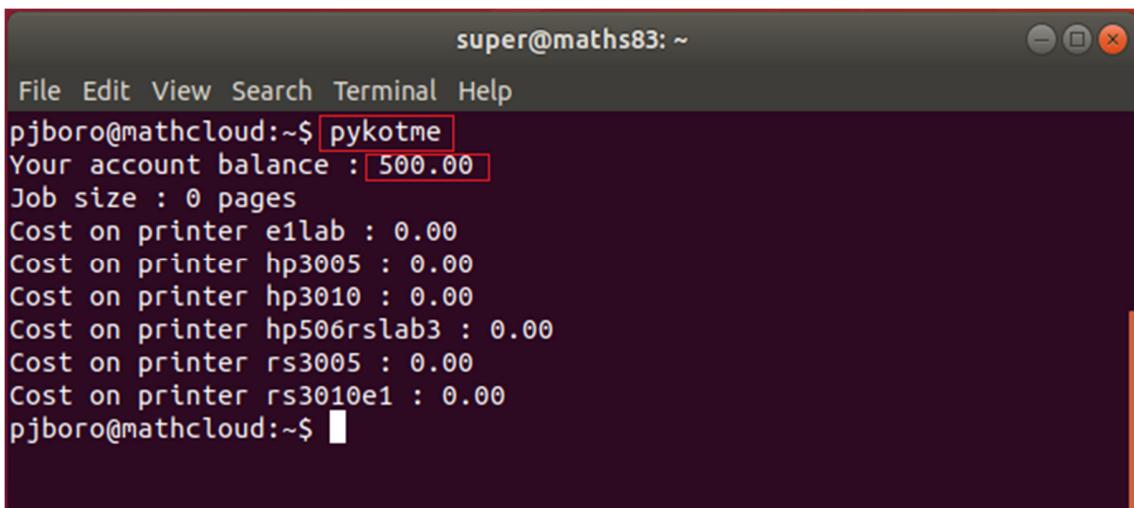
- Specify the print option as required

Some useful command for print purposes

(These requires a terminal windows in the server 172.16.70.1).

- Check print quota:

Type **pykotme** at command Prompt

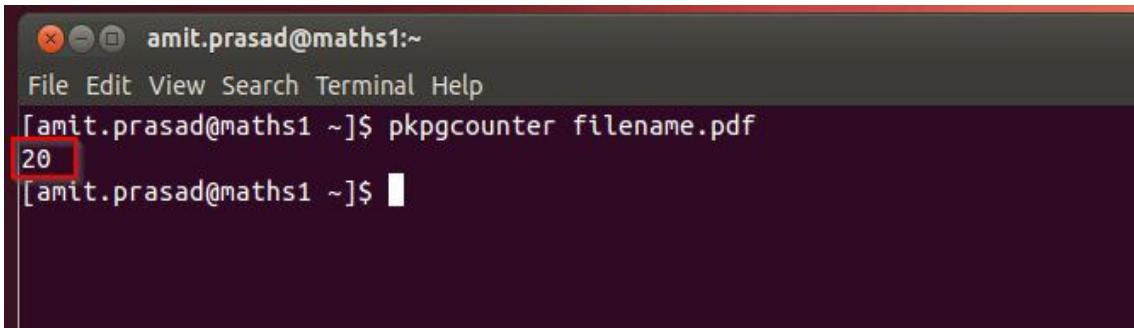


```
super@maths83: ~
File Edit View Search Terminal Help
pjboro@mathcloud:~$ pykotme
Your account balance : 500.00
Job size : 0 pages
Cost on printer e1lab : 0.00
Cost on printer hp3005 : 0.00
Cost on printer hp3010 : 0.00
Cost on printer hp506rslab3 : 0.00
Cost on printer rs3005 : 0.00
Cost on printer rs3010e1 : 0.00
pjboro@mathcloud:~$
```

Figure 10: Execute pykotme command and its output

- Check No. Of Pages in a file:

Type **pkpgcounter filename.pdf**



```
amit.prasad@maths1: ~
File Edit View Search Terminal Help
[amit.prasad@maths1 ~]$ pkpgcounter filename.pdf
20
[amit.prasad@maths1 ~]$
```

Figure 11: Execute pkpgcounter command and its output

Before print a file always makes sure that the READY LIGHT of the printer is **GREEN**. If not, there may be some print job which is in queue. To check the printer queue use **lpq -P printername** at SSH terminal. It will list a print queue with the corresponding user **emalid** who has given print command but the file is not printed yet. Ask the user to cancel the job using **cancel** command at SSH terminal. Or, you can cancel the job using **cancel** button on the **printer** and then restart the printer.

These instructions are important for you for avoiding unnecessary deduction of print quota from your print balance.

E. Accessing Internet

METHOD 1

For accessing Internet you may use the default Mozilla Firefox browser or Google Chrome browser or any other browser which is installed in your system. You should have webmail/Microsoft office 365 ID and password which will be provided by Computer and Communication Centre. Below are the steps for accessing internet

Once logged into the system one window should open automatically (screenshot attached). Type your email ID (without .iitg.ac.in) and password and click on the green color **LOGIN** button to access the internet.

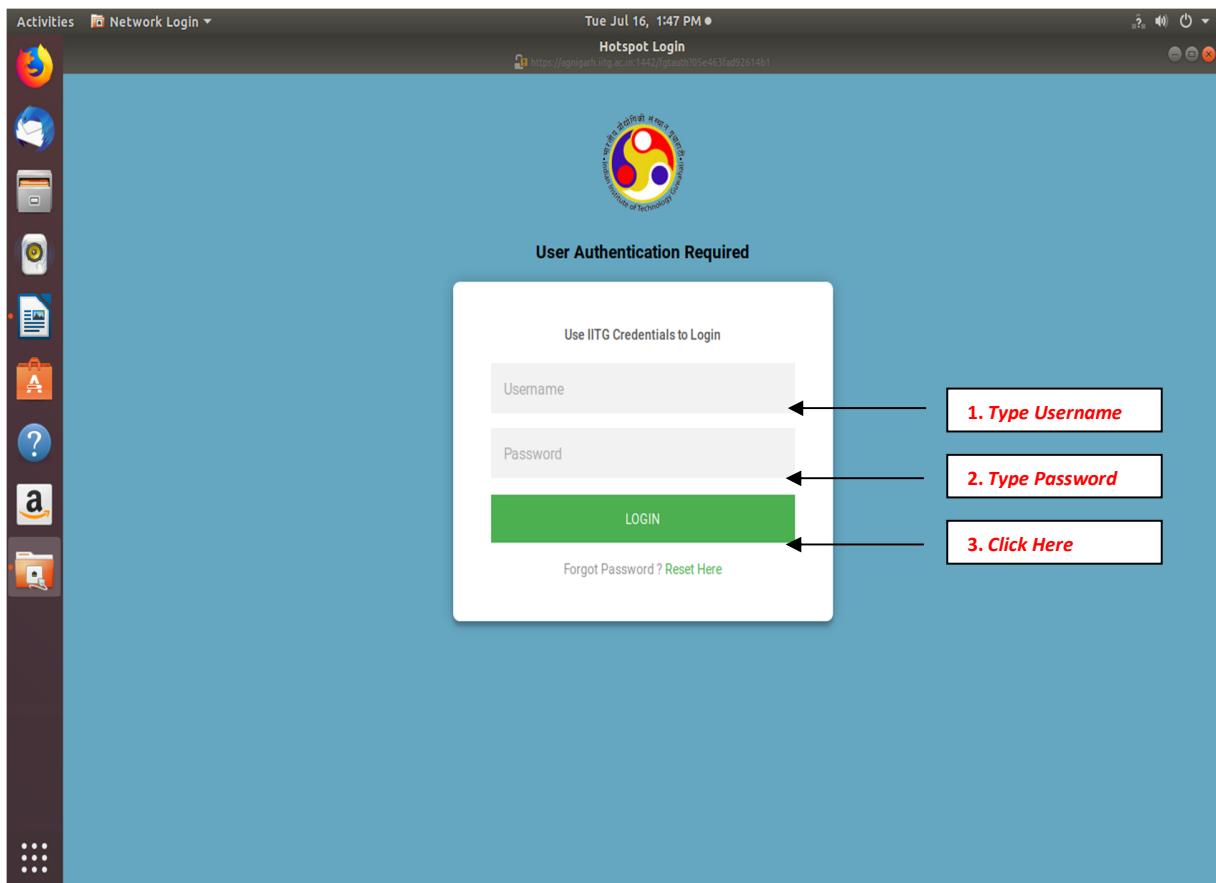


Figure 12(A): Internet access authentication window

If your login credentials are correct then you will get a window as shown below. Open a browser as your choice and access the internet.

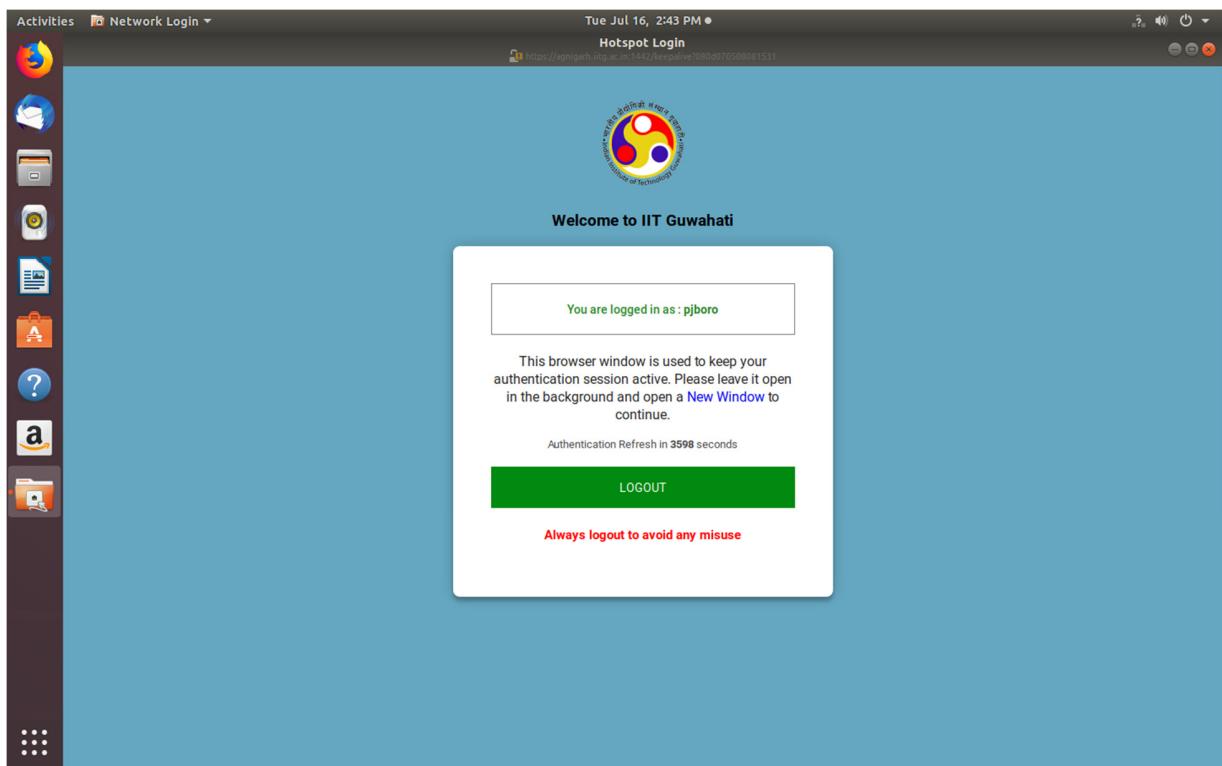


Figure 12(B): Internet access authentication window

METHOD 2

If you don't get any window like shown in Figure 12(A), then open Mozilla Firefox or Google-Chrome browser and click on the **Open Network Login Page** button from the top left corner of the browser.(Screenshot attached)

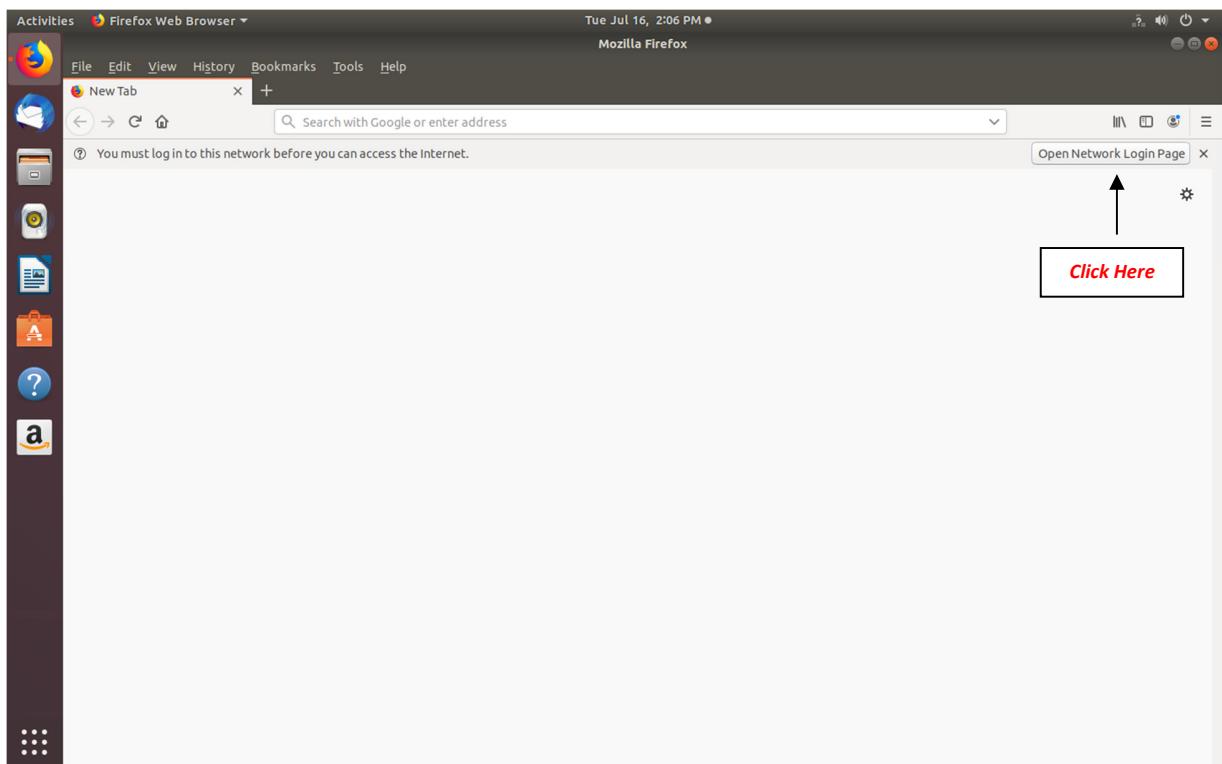
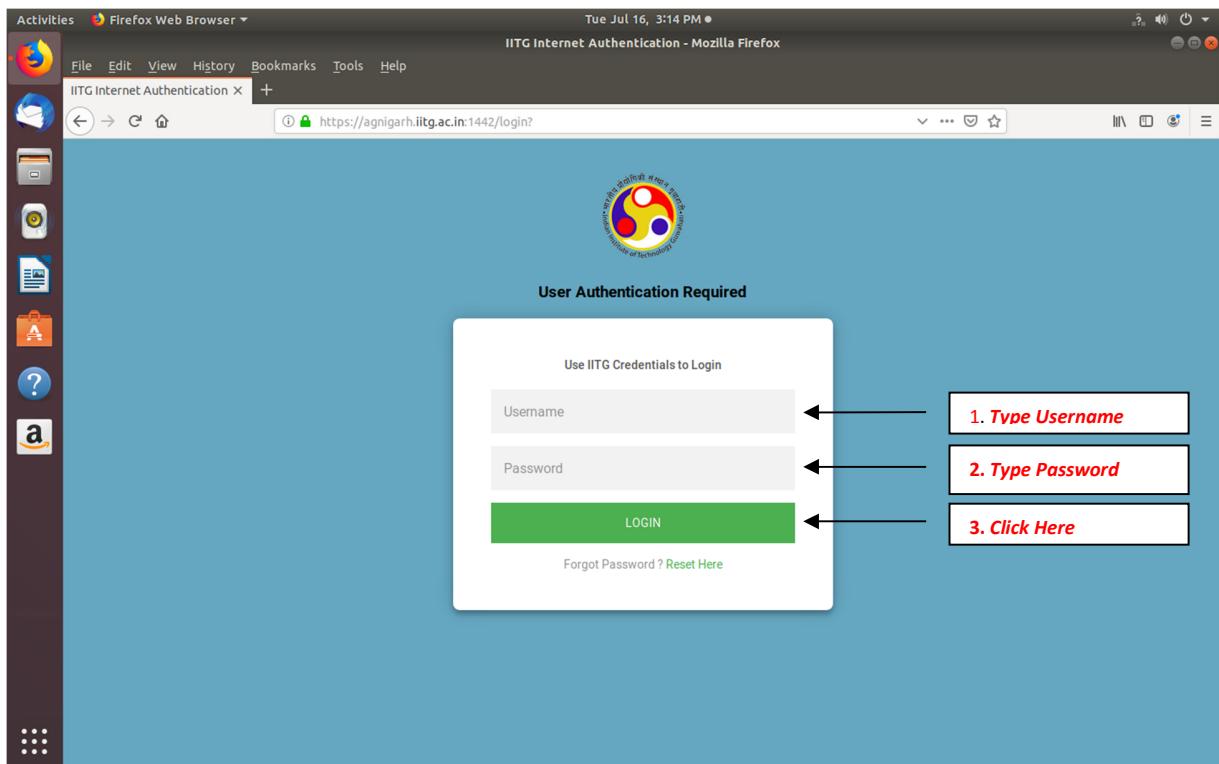


Figure 13

On clicking the **Open Network Login Page** button you will get the below shown window



Type your email ID (without .iitg.ac.in) and password and click on the green color button to access the internet.

LOGIN

Figure 14

METHOD 3

Open Mozilla Firefox or Google-Chrome browser and type the following line

<https://agnigarh.iitg.ac.in:1442/login?> in the address bar of the browser and press the Enter button of the keyboard. Type your user ID and password then click on the LOGIN button. (Screenshot attached)

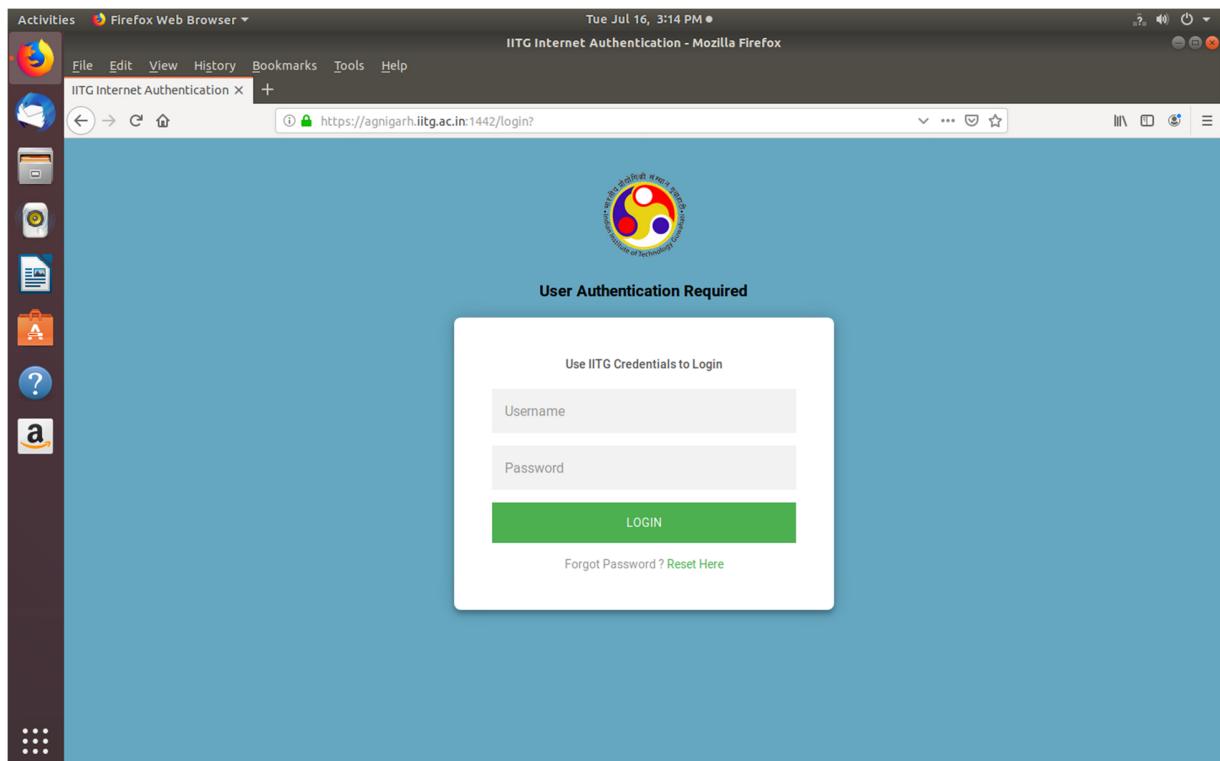


Figure 15

Question set for 02 Aug 2022

1. Take a real number r as input. Write a C program to calculate the area and the perimeter of a circle of radius r . ($\pi = 22/7$)
2. Let there be a circle centered at $(0, 0)$ and of radius r . Take r and a 2D point (a, b) as inputs and write a C program to identify whether (a, b) lies inside the circle or on the circle or outside the circle.

For example, let $r = 1$ and the 2D point is $(1, 0)$. So, the output should be “On the circle”.

3. Let $\{a_n\}$ be a sequence with $a_n = n!/e^n$. Write a C program to calculate the sum of the first 10 even terms of the above sequence. (i.e., $a_2 + a_4 + \dots + a_{20}$)
4. Take two integers a and b as inputs. Convert them into corresponding binary numbers and print them. Now, subtract the two binary numbers and print the result.

Question set for 16th Aug 2022

1. Take an integer n as input and generate a set of n random real numbers (use `rand()`, `srand()`). Write a C program to find the 2nd maximum element of the set.
2. Let $I = [a, b]$ be an interval and f, g be two polynomials on I . Write a C program to compare f and g .

Inputs: a, b , degree and coefficients of f and g .

Output: One of the following:

- a) $f > g$ over I .
- b) $f < g$ over I .
- c) f and g intersect on I .

Hint: Use discretization and divide the interval I into equal subintervals.

3. Take a real number x and ε as inputs. Let $s_n = \sum_{i=0}^n \frac{x^i}{i!}$ Write a C program to calculate n such that $|e^x - s_n| < \varepsilon$

Question set for 23th Aug 2022

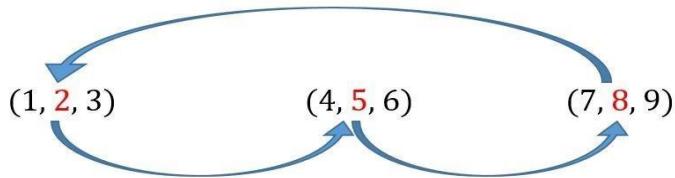
1. Take three 3-Dimensional points (i.e., each point should be a 3-tuple (x,y,z)) as inputs. Do the following:

A. Design a swap function with three variables

```
swap(int number_1, int number_2, int number_3)
```

B. Now, write a C program to swap ONLY the middle coordinates of the input points cyclically. All other coordinates must be unchanged.

For example: If input points are (1,2,3), (4,5,6), (7,8,9), then final output should be (1,8,3), (4,2,6), (7,5,9).



Your output should look like:

```
Enter first 3D point: 1 2 3
Enter second 3D point: 4 5 6
Enter third 3D point: 7 8 9
Before swapping the middle coordinates-----
1, 2, 3
4, 5, 6
7, 8, 9
After swapping, the points are-----
1, 8, 3
4, 2, 6
7, 5, 9
```

2. Take an integer n as input. Write a C program to generate two $n \times n$ matrices A and B of random integers greater than 10 using rand function. Print A and B. Further, take an alphabet 'c' as input.

If $c = a$, add A and B and print the resultant matrix.

If $c = s$, subtract A and B and print the resultant matrix.

Otherwise, print "Error".

3. Write a C program to multiply two polynomials.

Inputs: degree and coefficients of the two polynomials.

Your output should look like:

```
Degree of the first polynomial= 1
Degree of the second polynomial= 2
Enter the coefficients of the first polynomial- 2 1
Enter the coefficients of the second polynomial- 6 7 8
The required polynomial (in vector format) is- (12, 20, 23, 8)
```

Question set for 30th Aug 2022

1. Take an alphabet ‘a’ as input. Using switch statement, write a C program for the following two cases.
 - a. If ‘a’ = ‘c’, then your program should concatenate two input strings.

For example, if two input strings are “IIT”, “Guwahati”, then your output should be “IITGuwahati”.

(You are not allowed to use the `strcat()` function for concatenation)

 - b. If ‘a’ = ‘C’, then your program should convert all lower-case alphabet to upper-case and vice versa for an alphanumeric input string. All other symbols should remain unchanged.

For example, if an input string is \$Maths22, then your output should be \$mATHS22.
2. Take a natural number n as input. Write a C program by **using recursive function** to compute the sum of all odd natural numbers less than or equal to n.

Question set for 30th Aug 2022

1. There are n students and each of them has three courses. Every student has the following information.
 - i. Roll number (a two-digit integer)
 - ii. Full name (a string of alphabets)
 - iii. Marks of three courses (an array of three integers)

You are making a database with the above inputs. Now someone has come to you to get the information about a particular student.

First, that person **gave you the name of a student**. Upon searching with that name, you may find that there are multiple students with the same name.

- If the given name is not present in the database, print “No Match Found”.
- If the given name is unique, print all the following details: Roll number, Full name, Highest mark among 3 courses and Total marks of the 3 courses.
- Otherwise, you ask the person **to tell the roll number of that student**. After searching with the Roll number, print all the following details: Roll number, Full name, Highest mark among 3 courses and Total marks of the 3 courses.

Design a C program incorporating the above situations. **You must maintain the order of the occurrence of the incidents. Also, use struct() to build the database of each student.**

```
Enter the number of students: 3
Enter the roll number of the 1th student: 23
Enter the name of the 1th student: Subhajit Pramanick
Enter the marks of 3 courses: 36 89 64
Enter the roll number of the 2th student: 34
Enter the name of the 2th student: Saswata Jana
Enter the marks of 3 courses: 13 21 36
Enter the roll number of the 3th student: 38
Enter the name of the 3th student: Subhajit Pramanick
Enter the marks of 3 courses: 84 53 9
Enter a query name: Subhajit Pramanick
There are 2 students with the given name. Enter the roll number – 38
Details of the student are:
Roll Number - 38
Name – Subhajit Pramanick
Highest marks – 84
Total Marks - 146
```

2. Take two polynomials (a_0, a_1, a_2) and (b_0, b_1, b_2) of degree 2 such that a_2 and b_2 are the leading coefficients and $a_i \neq b_i$ for $i = 0, 1, 2$. Write a C program to find out the point(s) of intersection of these two polynomials **on real plane**. If there is no intersection point on the real plane, print “no intersection”. Your output should be in the form of (x, y) . (Print your answer up to 2 decimal places).

Sample output:

```
Enter the coefficients of the first polynomial: 1 2 1
Enter the coefficients of the second polynomial: 2 -2 2
The points of intersection are (0.26, 1.60), (3.73, 22.39)
```

Question set for 10th Sep 2022

1. Write a C program to reverse a given input string. You must follow the instructions given below.
 - i. You are **not allowed** to use <string.h> header file.
 - ii. You need to create a function **reverse()** that takes a string as its argument. You need to **pass the input string** to the function **reverse()** from the main function **using pointer**.
 - iii. **You need to print the string from the main, NOT from the function reverse().**

Sample Output:

```
Enter the string: IIT Guwahati
```

```
Reverse of the string is: itahawuG TII
```

2. Write a C program to check whether two given input vectors (each element of both vectors are non-zero) of size n (input) is linearly independent or not. You must follow the instructions given below.

- You have to **design a function linear_independent()** that takes **two vectors and an integer as arguments**. You need to **pass the two input vectors and n to the function using pointers**.

Sample Output:

```
Enter the size of the vectors n = 3
```

```
Enter the first vector: 2 4 6
```

```
Enter the second vector: 1 2 3
```

```
The vectors are linearly dependent.
```

3. Now, solve the same problem (Problem no 2) by modifying the function **linear_independent()** where, from now on, the function takes only two vectors as its arguments and there are no restrictions on the elements of the vectors. Also, the function should return “Yes” (or “No”) if the two vectors are linearly independent (or linearly dependent).

Note that “Yes” or “No” are strings. That means your function should return a string when you call the function from main.

Hint: To make a function that returns a string, you need to judiciously use pointers.

Question set for 27th Sep 2022

1. We all know Cayley Hamilton theorem which states that every square matrix satisfies its characteristic equation.

A square matrix M and a polynomial P(x) are given as inputs. Design a C program that verifies the Cayley Hamilton theorem for given M and P(x) i.e., whether $P(M) = \mathbf{0}$ or not. ($\mathbf{0}$ denotes the zero matrix). If $P(M) = \mathbf{0}$, then print YES, otherwise NO.

You need to design two functions. Firstly, design a function **Multiply()** that takes two matrices as arguments and multiplies two matrices. Second function **Scaler_Mult()** that takes a matrix and a scaler as inputs and does scalar multiplication. You need to pass the matrices using pointers from the main function. Also, allocate memory using malloc for the matrix.

```
Enter the degree of the polynomial: 2
Enter the coefficients starting from a_0:
-5
-4
1
Enter the matrix row-wise:
1
2
4
3
YES. Given matrix satisfies the given polynomial.
```

2. Take a 2D array of order $m \times n$ as an input ([allocate the memory using malloc](#)). Print this array as an output. [Write a C program to replace only the \$i\$ -th \(\$i < m\$ \) row of this array with the new row given by the user.](#) Print the modified 2D array after replacing the row.

Sample output:

```
Number of rows: 3
Number of columns: 3
Enter the array row-wise:
1 2 3
4 5 6
7 8 9
The input array is:
1 2 3
4 5 6
7 8 9
Which row you want to replace (< 3): 2
Enter the new elements of the row-2: 0 0 0
The modified array is:
1 2 3
0 0 0
7 8 9
```

MA 511: Computer Programming
Lecture 21: Linked list – create, insert, search

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

How to insert node to a linked list?

1. At the beginning of the list
2. At the middle of the list
3. At the end of the list
- :
- :

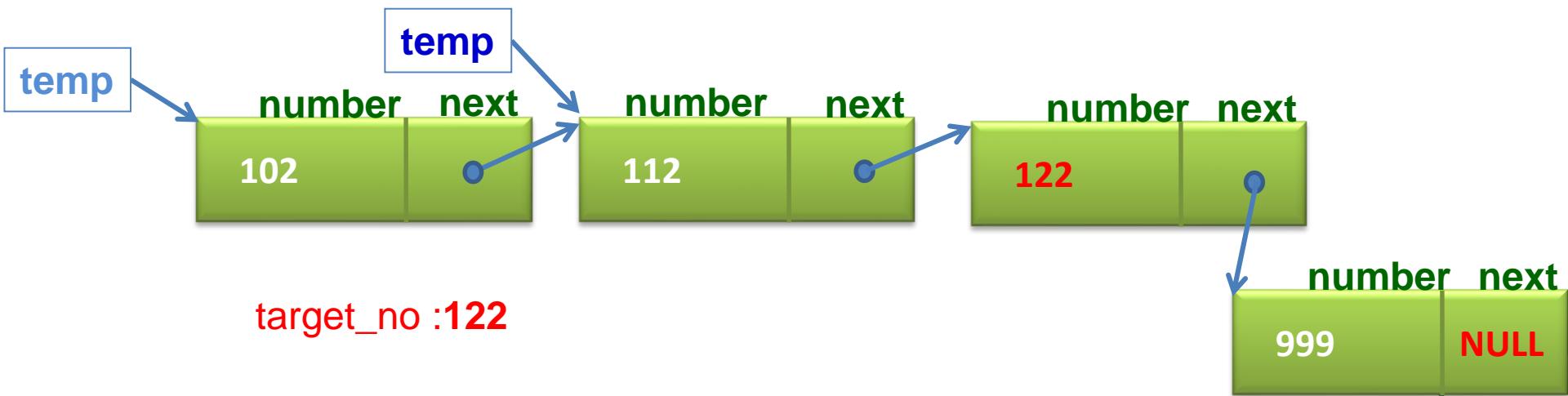
```
node *insert(node *pt);
```

```
main(){  
    node *start;  
    start = (node*)malloc(sizeof(node));  
    create(start);  
    print(start);  
    start=insert(start);  
    print(start);  
}
```

```
node *insert(node *list){  
    node *search(node *, int);  
    node *newnode, *target;  
    int target_no;  
    printf("Insert node before target data, type target ");  
    scanf("%d", &target_no);  
    if(list->data == target_no){ // add at the beginning  
        newnode = (node *) malloc(sizeof(node));  
        printf("type new data to insert");  
        scanf("%d", &(newnode->data));  
        newnode->next= list;  
        list = newnode;  
    }  
    else{ // finder return ptr of the preceding to the target node  
        target = search(list, target_no);  
        if (target ==NULL) printf("target no is not in list");  
        else { // add in the middle of the list  
            newnode = (node *) malloc(sizeof(node));  
            printf("type new data to insert");  
            scanf("%d", &(newnode->data));  
            newnode->next = target->next;  
            target->next= newnode;  
        }  
    }  
    return(list);  
}
```

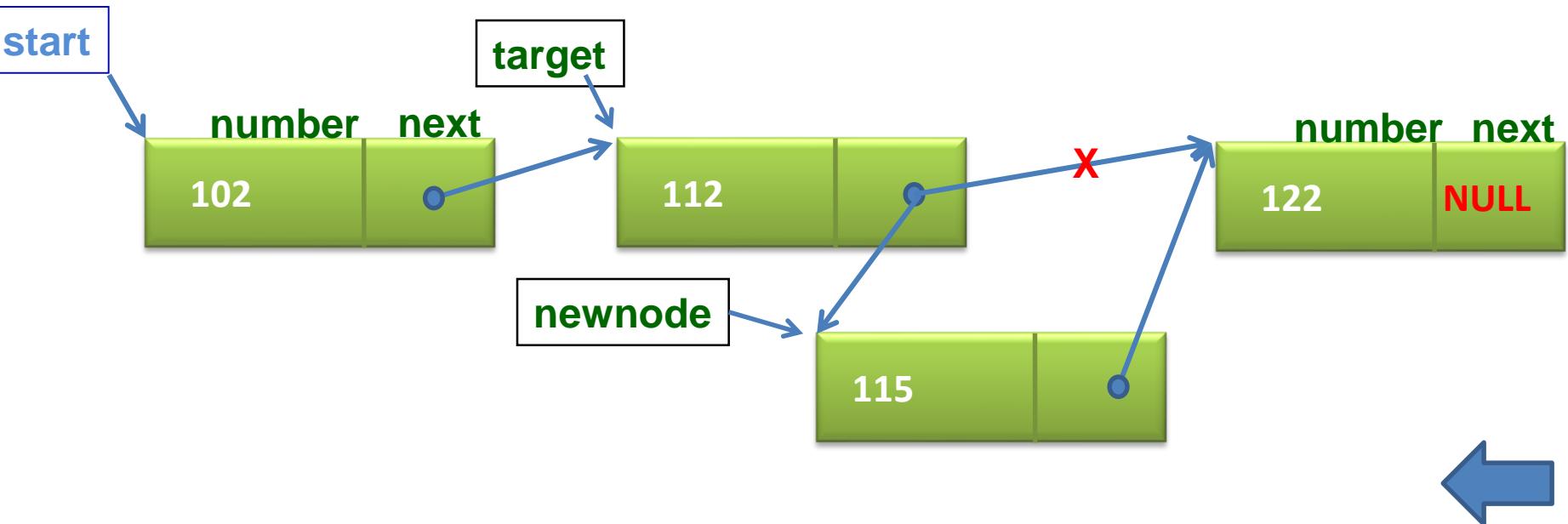
Searching the target node

```
node *search(node *temp, int target_no){  
    //return a ptr to the node before the target node  
    while(1){  
        if(temp->next->data==target_no)  
            return(temp);  
        else if(temp->next->next==NULL)  
            return(NULL);  
        else temp=temp->next;  
    }  
}
```



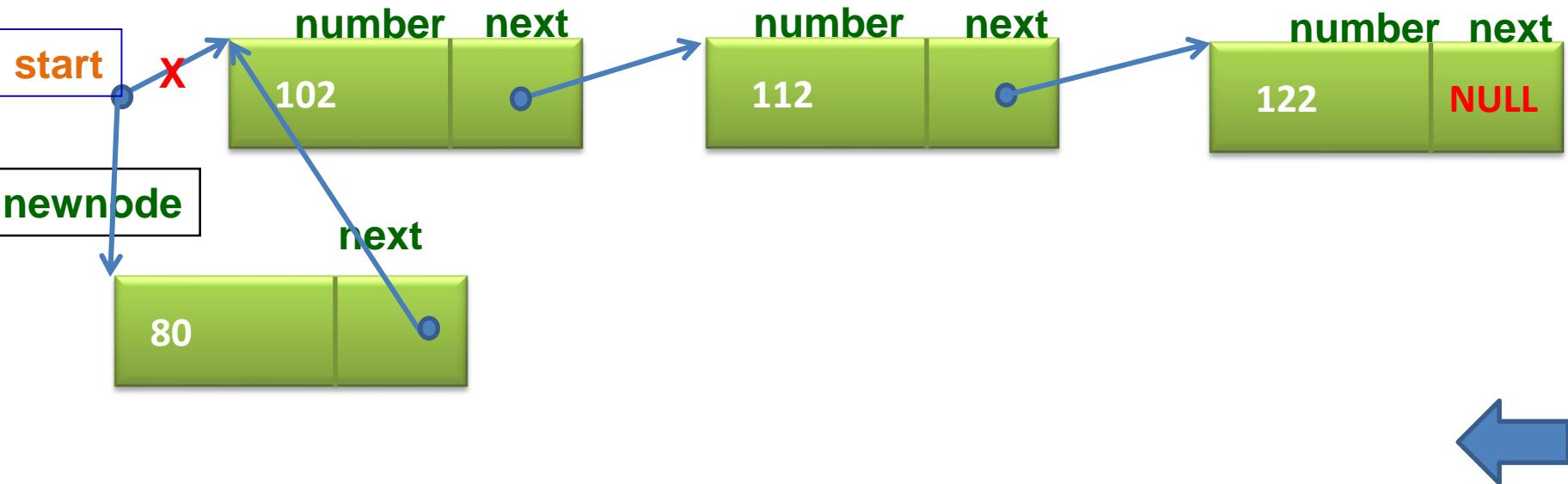
How to insert node in middle of the list?

```
newnode = (node *) malloc(sizeof(node));  
scanf("%d", &(newnode->data));  
newnode->next = target->next;  
target ->next= newnode;
```



How to **insert** node at the **beginning** of the list?

```
newnode = (node *) malloc(sizeof(node));  
scanf("%d", &(newnode->data));  
newnode->next= start;  
start = newnode;
```



Assignment

- Write a C-Program for inserting a new node in a linked list after the target data (or key). The key value of a new node and target key should be given from the terminal.
- Write a C-Program for inserting a new node in a linked list maintaining order with respect to the key value of nodes. The key value of a new node should be given from the terminal.

MA 511: Computer Programming

Lecture 22: Linked list – Deletion

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

How to delete node from a linked list?

```
node *delete(node *start)
node *search(node*, int)
node *target, *temp;
int target_no;

main(){
    node *start;
    start = (node*)malloc(sizeof(node));
    create(start);
    print(start);
    start=delete(start);
    print(start);
}
```

```
node *delete(node *list){
    node *search(node *, int);
    node *temp, *target;
    int target_no;
    printf("Type target key to be deleted");
    scanf("%d", &target_no);
    if(list->data == target_no){ delete from beginning
        temp= list->next;
        free(list);
        list = temp;
    }
    else{ finder return ptr of the preceding to the target node
        target = search(list, target_no);
        if (target ==NULL) printf("target key is not in list");
        else { delete from the middle of the list
            temp = target->next->next;
            free(target->next);
            target->next= temp;
        }
    }
    return(list);
}
```

Assignment

- Write a C-Program for deleting a node located just before the target node. The key value of the target node should be entered from the terminal.
- Write a C-Program for deleting a node from the linked list, the following node of the target node with a given key value. The key value should be given from the terminal.

MA 511: Computer Programming
**Lecture 23: Linked list – Circular linked list
and link reversal**

Partha Sarathi Mandal

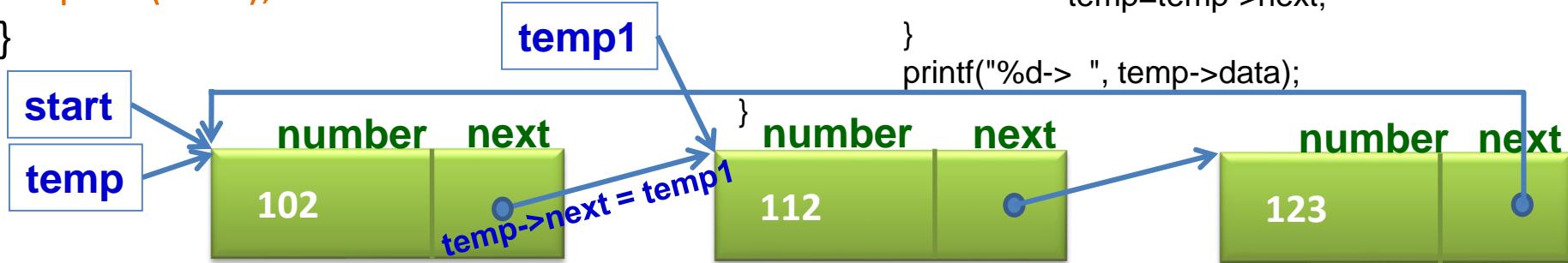
psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

How to create a Circular linked list ?

```
struct list_of_no {  
    int data;  
    struct list_of_no *next;  
};  
typedef struct list_of_no node;  
void create(node *pt);  
void print(node *pt);  
main(){  
    node *start;  
    start = (node *) malloc(sizeof(node));  
    scanf("%d", &start->data);  
    create(start);  
    print(start);  
}
```

```
void create(node *head){  
    node *temp=head, *temp1;  
    char c_value;  
    while(1){  
        printf("Type 'Y' to continue Type 'N' to stop: ");  
        scanf(" %c", &c_value);  
        if(c_value=='N'){ temp->next = head; break; }  
        else{ temp1 = (node *) malloc(sizeof(node));  
              scanf("%d", &(temp1->data));  
              temp->next = temp1;  
              temp=temp->next; } }  
void print(node *head){  
    node *temp=head;  
    printf("%d-> ", temp->data);  
    temp=temp->next;  
    while(!(temp==head)){  
        printf("%d-> ", temp->data);  
        temp=temp->next; } }  
printf("%d-> ", temp->data);
```



Linked reversal

```
struct list_of_no {  
    int data;  
    struct list_of_no *next;  
};  
typedef struct list_of_no node;  
void create(node *pt);  
void print(node *pt);  
node *ReverseLink(node *pt);  
main(){  
    node *start;  
    start = (node *) malloc(sizeof(node));  
    scanf("%d", &start->data);  
    create(start);  
    print(start);  
    start=ReverseLink (start);  
    print(start);  
}
```

```
node *reverseLinked(node *head) {  
    node *previous, *current, *next;  
    previous = NULL;  
    current = head;  
    while (current != NULL) {  
        next = current->next;  
        current->next = previous;  
        previous = current;  
        current = next;  
    }  
    return previous;
```

Assignments

1. Write a C-Program for **inserting** a new node in a **circular** linked list (i) before a target key, (ii) after a target key.
2. Write a C-Program for **deleting** a node from the **circular** linked list (i) preceding to give key (ii) after a give key (iii) node containing the key value.
 - where target key, key value of the inserting node and deleting node should enter in from the terminal

Assignments

1. Write a c program for creating a **doubly linked list** with a **Loop** or **recursive** function.
2. Write a C-Program for **inserting** a new node in a **doubly** linked list (i) before a target key, (ii) after a target key.
3. Write a C-Program for **deleting** a node from the **doubly** linked list (i) preceding to give key (ii) after a give key (iii) node containing the key value.
 - where target key, key value of the inserting node and deleting node should enter in from the console.

MA 511: Computer Programming

Lecture 24: Doubly Linked list

Partha Sarathi Mandal

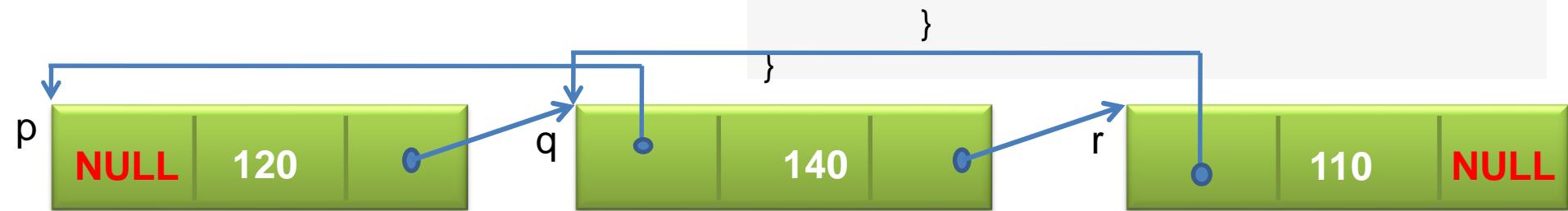
psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

How to create a Doubly linked list?

```
struct list_of_no {  
    int data;  
    struct list_of_no *prev;  
    struct list_of_no *next;  
};  
typedef struct list_of_no node;
```

```
main(){  
    node *p, *q, *r;  
    p = (node *) malloc(sizeof(node));  
    q = (node *) malloc(sizeof(node));  
    r = (node *) malloc(sizeof(node));  
    p->data = 120;  
    q->data = 140;  
    r->data = 110;  
    p->prev = NULL;  
    p->next = q;  
    q->prev = p;  
    q->next = r;  
    r->prev = q;  
    r->next = NULL;  
    while(p!=NULL){  
        printf(" %d", p->data);  
        p=p->next;  
    }  
}
```



How to create a Doubly linked list?

```
struct list_of_no {  
    int data;  
    struct list_of_no *prev;  
    struct list_of_no *next;  
};  
typedef struct list_of_no node;  
  
void create(node *pt);  
void print(node *pt);  
  
int main(void){  
    node *start;  
    start = (node *) malloc(sizeof(node));  
    printf("Type data for first node: ");  
    scanf("%d", &start->data);  
    start->prev = NULL;  
    create(start);  
    print(start);  
    return 0;  
}
```

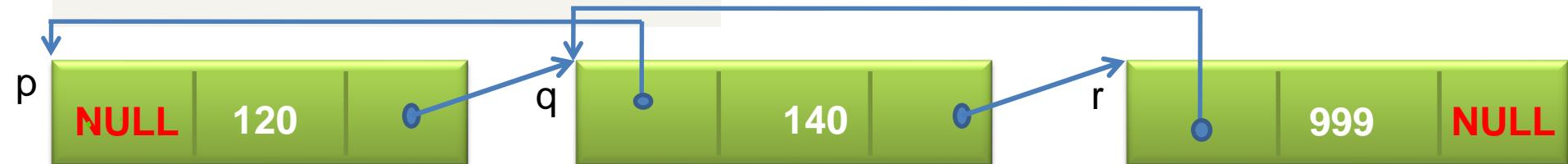
```
void create(node *head){  
    node *temp=head, *temp1;  
    char c_value;  
    while(1){  
        printf("Type Y to continue, Type N to stop: ");  
        scanf(" %c", &c_value);  
        if(c_value=='N'){  
            temp->next = NULL;  
            break;  
        }  
        else{  
            temp1 = (node *) malloc(sizeof(node));  
            printf("Type data for a new node: ");  
            scanf("%d", &(temp1->data));  
            temp->next = temp1;  
            temp1->prev = temp;  
            temp=temp->next;  
        }  
    }  
}
```



How to create a Doubly linked list?

```
struct list_of_no {  
    int data;  
    struct list_of_no *prev;  
    struct list_of_no *next;  
};  
typedef struct list_of_no node;  
  
void create(node *pt);  
void print(node *pt);  
  
int main(void){  
    node *start;  
    start = (node *) malloc(sizeof(node));  
    printf("Type data for first node: ");  
    scanf("%d", &start->data);  
    start->prev = NULL;  
    create(start);  
    print(start);  
    return 0;  
}
```

```
void print(node *temp){  
    node *head;  
    head = temp;  
    while(temp->next){  
        printf("|%d |->", temp->data);  
        temp=temp->next;  
    }  
    printf("|%d |->", temp->data);  
  
    printf("print using prev pointer\n");  
    while(temp!=head){  
        printf("|%d |->", temp->data);  
        temp=temp->prev;  
    }  
    printf("|%d |->", temp->data);  
}
```



Assignments

1. Write a c program for creating a **doubly linked list** with a **Loop or recursive** function.
2. Write a c program for creating a **circular doubly linked list** with a **Loop or recursive** function.
3. Write a C-Program for **inserting** a new node in a **doubly** linked list (i) before a target key, (ii) after a target key.
4. Write a C-Program for **deleting** a node from the **doubly** linked list (i) preceding to give key (ii) after a give key (iii) node containing the key value.
 - where target key, key value of the inserting node and deleting node should enter in from the console.

MA 511: Computer Programming

Lecture 25: Binary Search Trees

Partha Sarathi Mandal

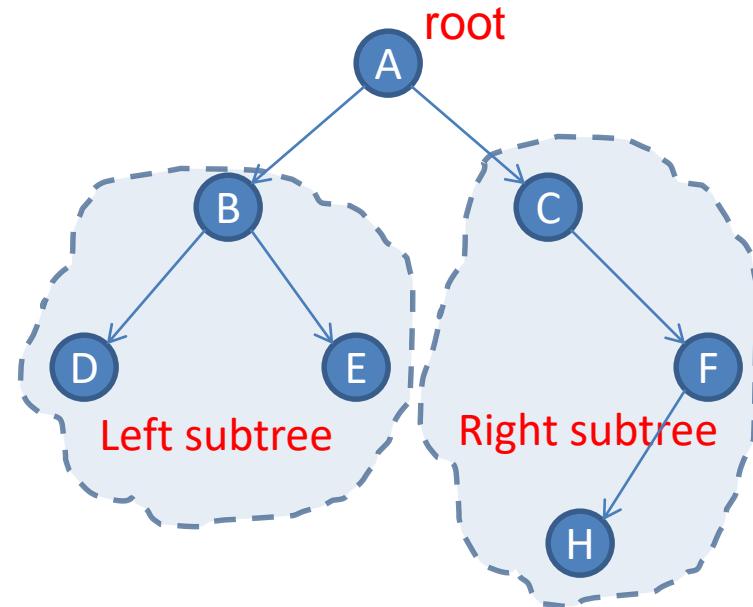
psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Trees

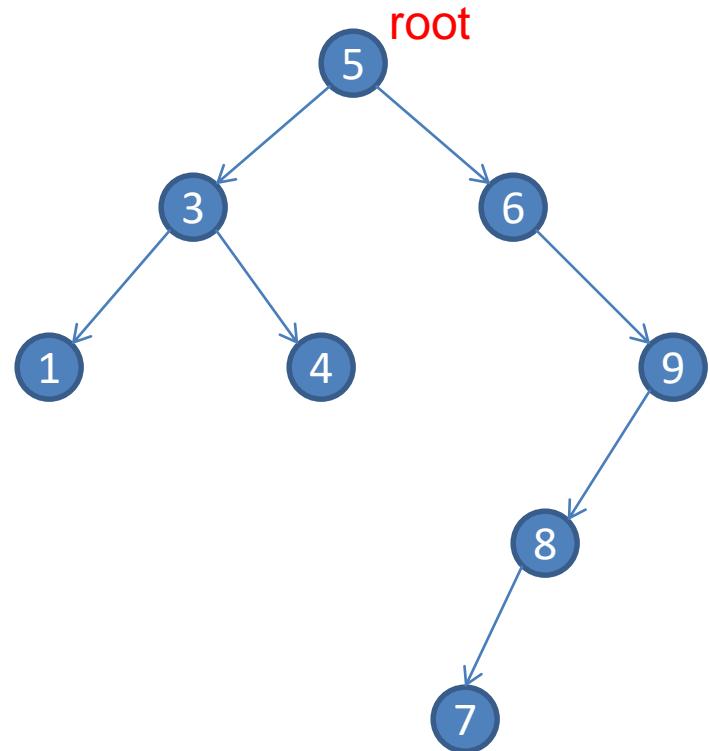
- **Binary Trees**

- A Binary tree is a finite set of elements that is either empty or is partitioned into at most three disjoint subsets.
- The first subset contains a single element called the **root** of the tree.
- The other two subsets are themselves binary trees, called the **left** and **right subtrees** of the original tree.



Binary Search Trees

- Its a Binary Trees with following property
- All elements in the left subtree of a node **n** are less than the contents of **n**, and all elements in the right subtree of **n** are grater than or equal to the contents of **n**.

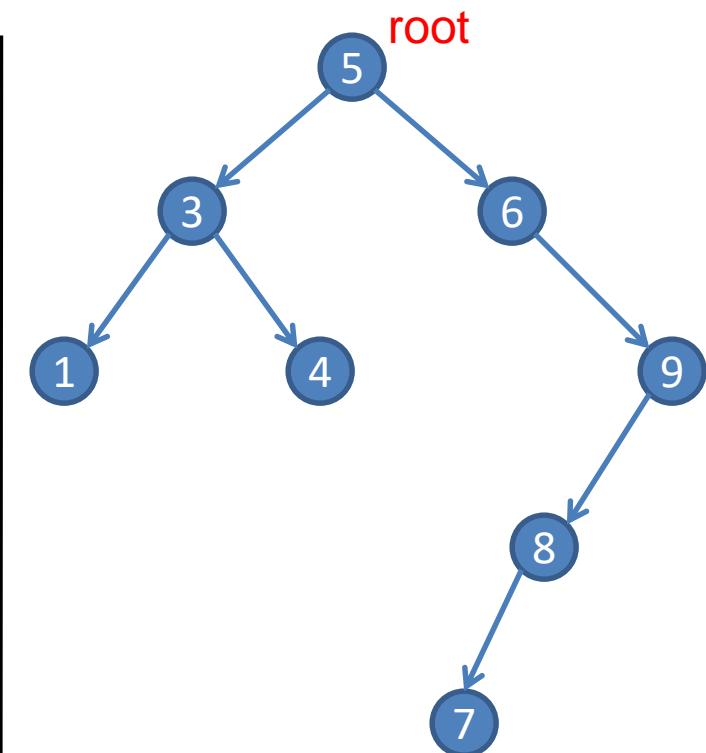
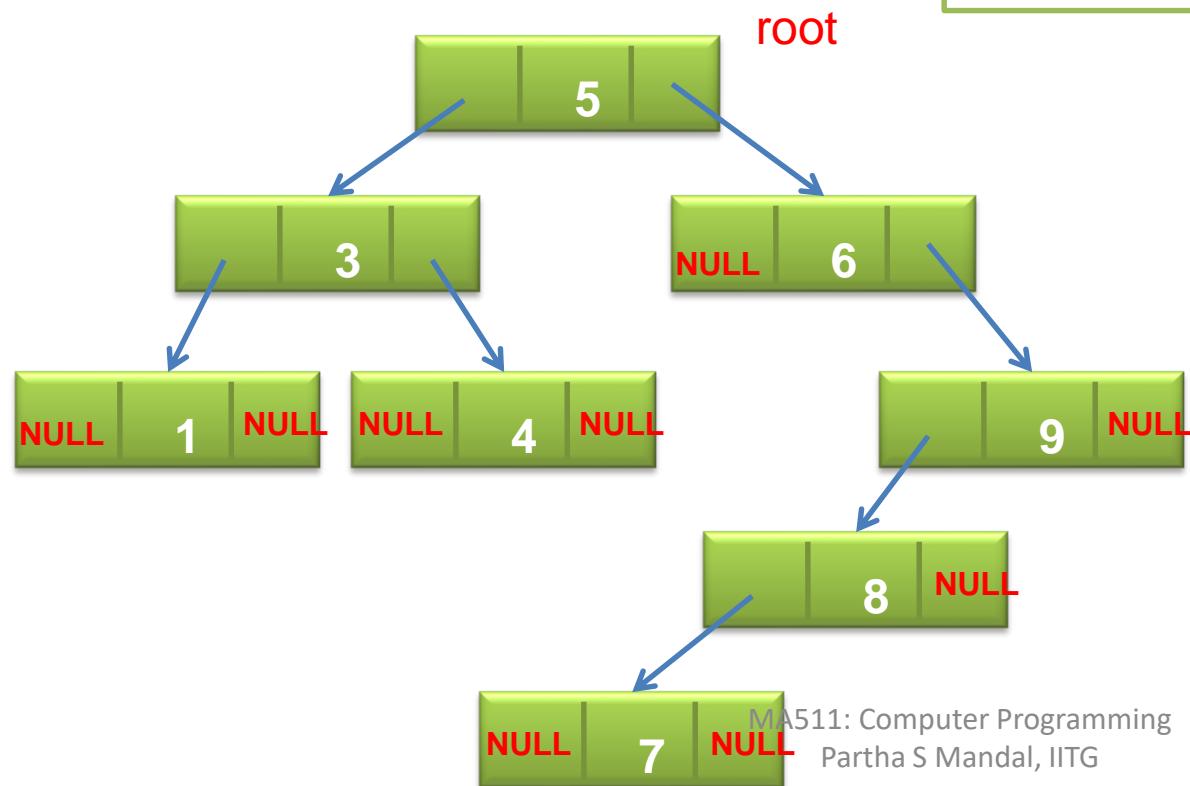


Binary search tree using linked list

node



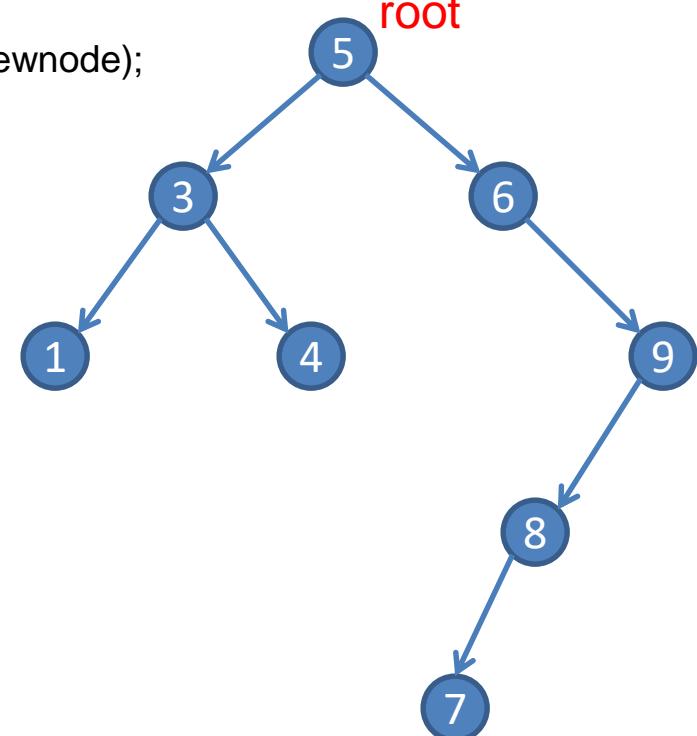
```
typedef struct BinaryTreeNode {  
    struct BinaryTreeNode *leftchild;  
    int data;  
    struct BinaryTreeNode *rightchild;  
} node;
```



Code for Binary Search Trees

```
typedef struct BinaryTree {  
    struct BinaryTree *leftchild;  
    int data;  
    struct BinaryTree *rightchild;  
}node;  
node *root=NULL;  
main(){  
    node *newNode;  
    int i, n, key;  
    printf("Type no of nodes: ");  
    scanf("%d", &n);  
    for(i = 0; i < n; i++){  
        printf("Type %dth data :", i);  
        scanf("%d", &key);  
        newNode = createNode(key);  
        add_node(newNode);  
    }  
    print_inorder(root);  
}
```

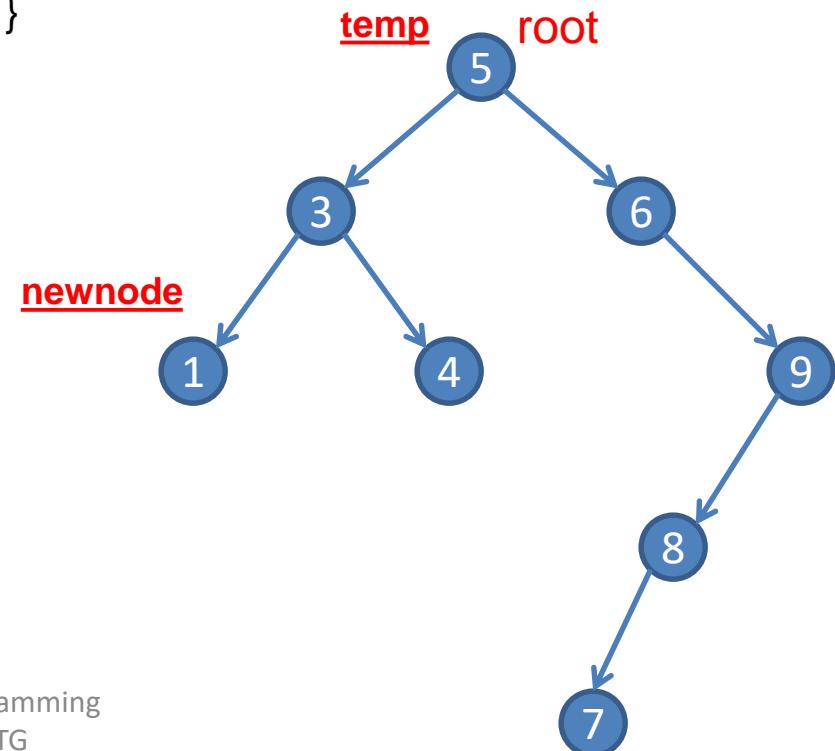
```
node *createNode(int value){  
    node *newnode;  
    newnode = (node *)malloc(sizeof(node));  
    if (newnode) {  
        newnode->data = value;  
        newnode->leftchild = NULL;  
        newnode->rightchild = NULL;  
    }  
    return(newnode);  
}
```



Code for Binary Search Trees

```
void add_node(node *newnode){  
    node *temp;  
    if (!root) root = newnode;  
    else {  
        temp = root;  
        while(1){  
            if(newnode->data < temp->data){  
                if(!temp->leftchild){  
                    temp->leftchild = newnode; break;  
                }  
                else temp = temp->leftchild;  
            }  
            else{  
                if(!temp->rightchild){  
                    temp->rightchild = newnode; break;  
                }  
                else temp = temp->rightchild;  
            }  
        }  
    }  
}
```

```
void print_inorder(node *node) {  
    if(node == NULL) return;  
  
    print_inorder(node->leftchild);  
    printf("%d ", node->data);  
    print_inorder(node->rightchild);  
}
```



Searching target in a given Tree

```
/* Given a binary tree, return 1 if a node with the
target data is found in the tree otherwise return 0.
Recurrs down the tree, chooses the left or right
branch by comparing the target to each node.*/

int search(struct node* temp, int target) {
    if (temp == NULL) return(0);           //not found
    else {
        if (target == temp->data) return(1); //found here
        else { //recur down to the correct subtree
            if (target < temp->data)
                return(search(temp->left, target));
            else
                return(search(temp->right, target));
        }
    }
}
```

```
main{
    :
    :
    printf("Type Target: ");
    scanf("%d", &target);
    j = search(root, target);
    if(!j) printf("Target NOT in tree");
    else printf("Target IN the Tree");
    :
    :
}
```

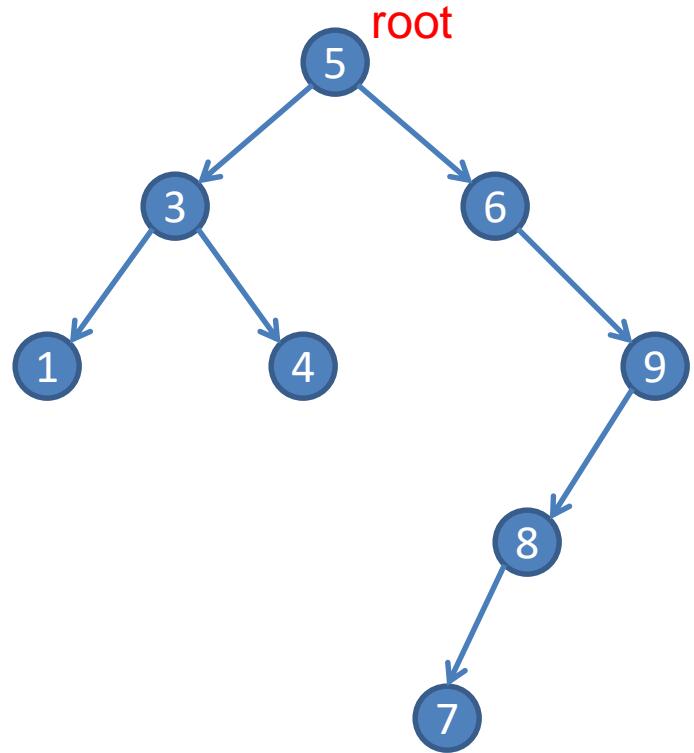
Traversal of a Binary Tree

- Preorder
 - Visit the root
 - Traverse the left subtree in preorder
 - Traverse the right subtree in preorder

Ex. 5 3 1 4 6 9 8 7
- Inorder
 - Traverse the left subtree in inorder
 - Visit the root
 - Traverse the right subtree in inorder

Ex. 1 3 4 5 6 7 8 9
- Postorder
 - Traverse the left subtree in postorder
 - Traverse the right subtree in postorder
 - Visit the root

Ex. 1 4 3 7 8 9 6 5



Assignments

- Delete a node from a Binary Search Tree for given target key.

MA 511: Computer Programming

Lecture 26: File read and write

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

I/O

- Buffer
 - A segment of memory used to hold data while a program is being processed.
 - In a program, buffers are created to hold some amount of data from each of the files that will be read or written.
- File
 - A bunch of blocks of information written on the hard-drive
 - OS knows the location and order of blocks and will present you with a continuous view of file.
- FILE (as in File *)
 - A data structure that holds file information
 - What file
 - Current position in file
 - Permissions (read, write)

Data Files

Example:

```
#include <stdio.h>
main(){
    FILE *fp;
    int i;
    fp = fopen("output.dat", "w");
    if(fp==NULL)
        printf("Error for opening a file\n");
    for(i=65; i<90; i++)
        fprintf(fp, "%c\n", i);
    fclose(fp);
}
```

output.dat

A
B
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y

Data Files

Example:

```
#include <stdio.h>
main(){
    FILE *fp;
    int i;
    fp = fopen("output.dat", "w");
    if(fp==NULL)
        printf("Error for opening a file\n");
    for(i=65; i<90; i++)
        fprintf(fp, "%c\n", i);
    fclose(fp);
}
```

FILE: spl structure type that establishes the buffer area

fp : pointer variable pointing to the beginning of the buffer area.

fp=fopen(file-name, file-type);

“r” : open a existing file for reading

“w”: opening a new file for writing, overwrite if file exist.

“a” : opening an existing file for appending, create if not exist.

“r+”: opening an existing file for read and write.

“w+”: opening a new file for both reading & writing, overwrite if file exist.

“a+”: opening a new file for reading and appending, create if not exist.

Reading a Data File

Example:

```
#include <stdio.h>
main(){
    FILE *fp1;
    char ch;
    fp1 = fopen("output.dat", "r");
    if(fp1==NULL)
        printf("Error for opening a file\n");
    else{

        while(!feof(fp1)){
            fscanf(fp1, "%c", &ch);
            printf("%c",ch);
        }
    }
    fclose(fp1);
}
```

Assignments

- Write a C program for inserting an integer before a given target value in a given array of integer.
- Write a C program for deleting a given value in a given array of integer.
- Write a C program for (i) writing integers 1-100 to a file (fwrite.dat) (ii) read data from the file fwrite.dat, identify numbers which are perfect square and write over a new file (newfile.dat).

MA 511: Computer Programming

Lecture 27: Unformatted data files

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

```
typedef struct {
    int acct_no;
    char acct_type;
    char name[80];
    float balance;
}record;
record readData(record cust);
void writeToFile(record cust);
FILE *fp1;
```

```
main(){
    int i;
    record customer;
    fp1 = fopen("file.dat", "w");
    if(fp1==NULL)
        printf("Error for opening a file\n");
    else{
        while(1){
            printf("Type 0 (zero) to stop ");
            scanf("%d", &i);
            if(i==0) break;
            customer = readData(customer);
            writeToFile(customer);
        }
    }
    fclose(fp1);
}
```

Example: Data files

```
record readData(record cust){
    scanf(" %[^\n]", cust.name);
    scanf(" %d", &cust.acct_no);
    scanf(" %c", &cust.acct_type);
    scanf(" %f", &cust.balance);
    return(cust);
}

void writeToFile(record cust){
    fprintf(fp1, "%s\n", cust.name);
    fprintf(fp1, "%d\n", cust.acct_no);
    fprintf(fp1, "%c\n", cust.acct_type);
    fprintf(fp1, "%.2f\n", cust.balance);
}
```

Unformatted data files

- `fread` `fwrite`: are called unformatted read write functions follows:
 - Read an entire block from data file or write the entire block to a data file.
 - Function required four arguments:
 - A pointer to the data block
 - The size of the data block
 - No of the data block being transferred
 - Stream pointer (File pointer)

```
fwrite(&customer, sizeof(record), 1, fp1);
```

```
fread(&customer, sizeof(record), 1, fp1);
```

```
typedef struct {
    int acct_no;
    char acct_type;
    char name[80];
    float balance;
}record;
record readData(record cust);
void writeToFile(record cust);
FILE *fp1;
```

```
main(){
    int i;
    record customer;
    fp1 = fopen("file.dat", "w");
    if(fp1==NULL)
        printf("Error for opening a file\n");
    else{
        while(1){
            printf("Type 0 (zero) to stop ");
            scanf("%d", &i);
            if(i==0) break;
            customer= readData(customer);
            // writeToFile(customer);
            fwrite(&customer, sizeof(record), 1, fp1);
            strset(customer.name, ' '); //erase strings
            strset(customer.acct_type, ' ');
        }
    }
    fclose(fp1);
}
```

Unformatted data files

```
record readData(record cust){
    scanf(" %[^\n]", cust.name);
    scanf(" %d", &cust.acct_no);
    scanf(" %c", &cust.acct_type);
    scanf(" %f", &cust.balance);
    return(cust);
}

void writeToFile(record cust){
    fprintf(fp1, "%s\n", cust.name);
    fprintf(fp1, "%d\n", cust.acct_no);
    fprintf(fp1, "%c\n", cust.acct_type);
    fprintf(fp1, "%.2f\n", cust.balance);
}
```

Binary Files

- We learned how to handle text file in last class, which is a default mode.
- All machine language files are **binary files**
Ex: .com, .exe, .obj, .dll etc.
- File mode has to be mentioned as “rb” & “wb” in fopen command for opening a binary file [“rt” & “wt” for text file]
`fptr=fopen("file.dat", "rb/wb")`
- Text files can also be stored and processed as binary files but not vice versa.
- Binary files differ from text files in two ways mainly:
 - The storage of newline characters (\n)
 - The eof character

the storage of newline characters

- in text files ‘`\n`’ is stored as a single character by user, it takes 2 bytes of storages inside memory since it’s a collection of two characters.
- In binary file it takes 1 bytes of storages inside memory.
- If we count the number of characters of a text file, each newline character contributes by one.
- If we store 10 newline characters in a text file but try to count characters of this file by opening in binary mode.
- The count will be 20.

the **eof** character

- The **eof** corresponds to the character having ASCII code 26 for text file.
- In binary files there is no such explicit **eof** character, and do not store any special character at the end of the file and their file-end is verified by using their size itself.

Storage of number in binary format

- **fprintf** stores numbers as sequence of alphabets
 - storage of 1001 in a file (text and binary both) done as sequence of 4 alphabets ‘1’, ‘0’, ‘0’, ‘1’. 4-digit number will take 4 bytes.

```
for(i=10001; i<=10100; i++)  
    fprintf(fp, "%d", i);
```

- **fwrite** will store every integer value by taking 2 bytes (independent of the number of digits).

```
for(i=10001; i<=10100;i++)  
    fwrite(&i, sizeof(int), 1, fp);  
    [200 bytes (2 bytes for each 100 integer)]
```

Storage number in binary format

```
main(){                                //here sizeof(int) is 4 bytes
    FILE *fp1, *fp2;
    int i;
    fp1 = fopen("fp.dat", "wb");
    fp2 = fopen("fw.dat", "wb");
                    //printf("Size of Integer = %d\n", sizeof(int));
    if(fp1==NULL) printf("Error for opening file fp2\n");
    else if (fp2==NULL) printf("Error for opening file fp2\n");
    else{
        for(i=10001; i<=10100; i++){
            fprintf(fp1, "%d", i); //fprintf(fp1, "%d\n", i);
            fwrite(&i, sizeof(int), 1, fp2);
        }
    }
    fcloseall();
}
```

Verify the results with different data types and different
file types and check file size in each case

adv. and disadvantage in Binary file format

- File storing in binary form save a lot of space.
- Any editor / word processor cannot read the file in binary format.
- Special program is required to read file in binary format.

MA 511: Computer Programming
Lecture 28: command line parameters

Partha Sarathi Mandal

psm@iitg.ac.in

Dept. of Mathematics, IIT Guwahati

Command line parameters

- Without recompiling a program its possible to pass different starting values (special arguments) in an iterations through the empty parentheses of the **main** i.e., **main()**.
- Following two arguments are generally allow most of the C version.
- The parameters to be passed to **main** from OS.

```
main(int argc, char *argv[ ]) {
```

```
}
```

- argc** : an integer variable.
- argv** : an array of pointer to characters i.e., an array of strings.
- Each **string** in this array will represent a parameter and that passes to **main**.

Command line parameters

mainArg.c

```
#include <stdio.h>

main(int argc, char *argv[ ]){

    int i;
    printf("argc = %d\n", argc);
    for(i = 0; i<argc; ++i)
        printf("argv[%d] = %s\n", i, argv[i]);
}
```

```
$ cc mainArg.c -o mainArg
$ ./mainArg my name is Rana
argc = 5
argv[0] = ./mainArg
argv[1] = my
argv[2] = name
argv[3] = is
argv[4] = Rana
```

Command line parameters

mainArg1.c

```
#include <stdio.h>
main(int argc, char *argv[ ]){
    int i, n;
    float sum=0.0, x, term = 1.0;
    sscanf(argv[1], "%f", &x);
    sscanf(argv[2], "%d", &n);
    for(i = 1; i<=n; ++i){
        term *= x/(float)i;
        sum = sum + term;
    }
    printf("x = %f, n = %d, sum = %f\n", x,n,sum);
}
```

```
$ cc mainArg1.c -o mainArg1
$ ./mainArg1 0.3 10
x = 0.300000, n = 10, sum = 0.349859
```

Command line parameters

mainArg2.c

```
#include <stdio.h>
main(int argc, char *argv[ ]){
    FILE *fp;
    char ch;
    fp = fopen(argv[1], "r");
        if(fp == NULL) printf("Error for opening a
file\n");
    else{
        while(!feof(fp)){
            fscanf(fp, "%c\n", &ch);
            printf("%c\n", ch);
        }
    }
    fclose(fp);
}
```

output.dat

A
B
C
D
E
F

```
$ cc mainArg2.c -o mainArg2
$ ./mainArg2 output.dat
```

A
B
C
D
E
F

Command line parameters

mainArg.c

```
#include <stdio.h>
main(int argc, char *argv[]){
    FILE *fp1;
    int n, i, j=0;
    float x, sum=0.0, term = 1.0;
    fp1 = fopen(argv[1], "r");
        if(fp1==NULL) printf("Error for opening a file\n");
    else{ while(j++!=7){
            fscanf(fp1, "%d %f\n", &n, &x);
            for(i = 1; i<=n; ++i){
                term *= x/(float)i;
                sum = sum + term;
            }
            printf("n = %d, x = %f, sum = %f\n", n, x,sum);
        }
    }
    fclose(fp1);
}
```

inputFile.dat

```
2 0.200000
1 0.800000
3 0.900000
2 0.100000
3 0.700000
4 0.100000
2 0.500000
```

```
$ cc mainArg.c -o mainArg
$ ./mainArg inputFile.dat
n = 2, x = 0.200000, sum = 0.220000
n = 1, x = 0.800000, sum = 0.236000
n = 3, x = 0.900000, sum = 0.258824
n = 2, x = 0.100000, sum = 0.259028
n = 3, x = 0.700000, sum = 0.259038
n = 4, x = 0.100000, sum = 0.259038
n = 2, x = 0.500000, sum = 0.259038
```