CLASS FUN

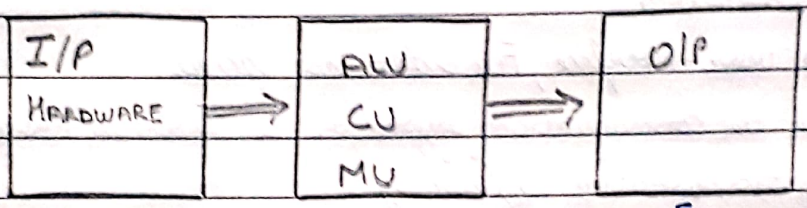
Date | | | | |

Page No.

# UNIT - 1)

## COMPUTER SYSTEMS

- A computer is an electronic device that stores, manipulates & retrieves data.
- A system is a group of several objects with a process.
  Eg. A school involves students / teachers (Objects) and teaching (Process)




ALU → Performs Arithmetic and Logical Operations. [ARITHMETIC LOGIC UNIT]

CONTROL UNIT

CU → Governs operations such as Storing / computing & retrieving data.

MU (MEMORY UNIT) → STORING DATA

### → TYPES OF MEMORY

1) PRIMARY MEMORY

- Read only memory (ROM) stores data even after computer is turned off. It stores BIOS information. Content cannot be modified once written.
- RAM → Stores data & instructions when the computer is turned on. Content can be modified any number of times. Stores programme under execution.

2) • CACHE MEMORY

Stores data & instruction referred by processor.

3) SECONDARY MEMORY

- Magnetic Storage: Stores information that can be read, erased & rewritten a number of times. Eg. Floppy disk, Hard disk, Magnetic tapes
- OPTICAL STORAGE

Uses laser beams to read & write (burn) data.

Eg. CD, DVD

→ COMPUTER SOFTWARE

# SYSTEM SOFTWARE

Programs that manage the hardware resources of a computer & perform required processing tasks. Run in background

1) OPERATING SYSTEM
- Provides user-interface, File & database access.
- Interface to communication systems. ie Internet Protocols.
- Keep system operating efficiently.

2) SYSTEM SUPPORT
- Provides System utilities and other services
- Ex: Sort and Disk format programs.
- Security monitors to protect system & data.

3) SYSTEM DEVELOPMENT SOFTWARE
- Language Translators, debugging tools & Computer-Assisted Software Engineering (CASE)

# APPLICATION SOFTWARE

1) GENERAL PURPOSE
- Can be used for more than one application example.
- Ex. Word processors, DBMS, ~~Computer aided design system~~

2) APPLICATION SPECIFIC
- Can only be used for intended purpose.
- Ex. Console, MATLAB, Tableau

CLASS FUN

Date | Page No.

# COMPUTING ENVIRONMENT

1) PERSONAL / STANDALONE:
- Satisfies needs of a single user, who uses computer for personal tasks.
- Ex. Personal Computer
- No Data Sharing

2) TIME SHARING:
- Sharing of processing of computer by time-sharing.
- Processes are queued in ~~turns~~ priority.

3) CLIENT / SERVER:
- Processing b/w two machines.
- Client Machine → requests processing
- Server Machine → Offers processing

4) DISTRIBUTED COMPUTING:
- Seamless integration of computing functions b/w different servers & clients.
- All clients / Servers share processing tasks.

# COMPUTER LANGUAGES:

→ MACHINE LANGUAGE
- Each machine has its machine code, generally made up of a stream of binary.

* ADVANTAGES
1) High speed execution
2) Computer can understand instruction immediately
3) No translation needed

Class Fun
Date
Page No.

* DISADVANTAGES:

1) Machine dependent
2) Difficult to understand
3) Difficult to write bug-free program
4) Difficult to isolate an error.

→ ASSEMBLY LANGUAGE

- Use of symbolic language to simplify machine code.
- Assembler → Converts Assembly code into machine code

* ADVANTAGE

- Easy to understand
- Easy to modify and isolate errors
- More control on hardware

* DISADVANTAGES

- Machine dependent
- Requires translator/assembler
- Slow development time
- Less efficient

→ HIGH-LEVEL LANGUAGES

- Used to allow programmers to focus on application rather than hardware intricacies.
- C → System Implementation Language

* ADVANTAGE

- Easy to write & understand
- Machine Independent
- Better Portability
- Better readability
- Easier to document
- Easy to maintain

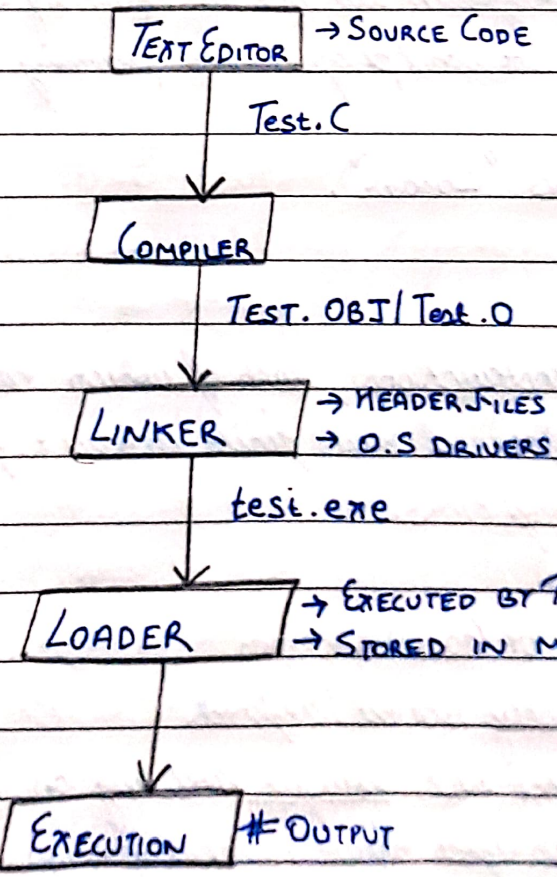
* DISADVANTAGES

- High execution time
- Less efficient
- Poor control on hardware
- Needs translator (Compiler)

→ LANGUAGE TRANSLATORS

| COMPILER | INTERPRETER |
|---|---|
| • Compiles an entire program, generates an executable file. | • Translates each line of code so it is executed |
| • Stored in disk for further use. | • Executable program is generated on RAM & requires Interpreter for each execution |
| • Run faster | • Run slower |
| • Most languages use compiler Ex. C, C++, Java | • very few languages use interpreter. Ex. Python, Perl |

# CREATING & RUNNING PROGRAMS



CLASS FUN

Date | Page No.

# COMPILATION PROCESS

→ PREPROCESSOR

- Reads source code and prepares it for translator.
- Scans for preprocessor commands, ie #include, #define, #undef

→ TRANSLATOR

- After preprocessing of code, translator reads translation unit & writes resulting object module to a file ie. Machine code.

→ LINKER

- Assembles all functions, into one executable program.

# EXECUTION OF A PROGRAMME

- To execute a program we use an operating system command, such as "run", to load program into primary memory and execute it.
- This is known as "Loader".

# ALGORITHMS

- Finite set of instructions, each of which has a clear meaning & can be performed with finite amount of effort in a finite length of time.

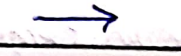
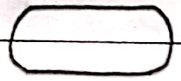
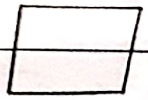
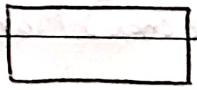
→ CHARACTERISTICS OF GOOD ALGORITHM

- Steps are precisely stated / defined.
- Result of each step are unique and rely on input & output of previous step
- Algorithm stops after finite number of steps are executed.

CLASS FUN

Date | Page No.

# FLOWCHART

| | SYMBOL | PURPOSE | DESCRIPTION |
|---|---|---|---|
| 1) |  | FLOW LINE | Indicates flow of logic |
| 2) |  | TERMINAL (START/STOP) | Represents start/stop of flowchart |
| 3) |  | INPUT/OUTPUT | Used for Input & Output operation |
| 4) |  | PROCESSING | Used for arithmetic operations & data manipulation |
| 5) |  | DECISION | Used to represent operation in which there are two alternative (T/F) |
| 6) |  | ON PAGE CONNECTOR | Used to join different flow lines |

# INTRODUCTION TO C-LANGUAGE

- High purpose level general purpose language

## → FEATURES

- Easy to learn
- Structured language
- Produces efficient programs
- Can handle low-level activities
- Can be compiled on variety of computers

## → FACTS

- ~~Used to~~ Invented to write operating system "UNIX".
- Successor of B-language
- Formalized in 1988 by AMERICAN NATION STANDARD INSTITUTE (ANSI)
- Most widely used system programming language.
- By 1973 ~~Ken~~ UNIX written almost only in C.

CLASS FUN

Date | Page No.

→ USES:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print spoolers
- Databases
- Interpreter
- Utilities
- Modern Programs.

# C- TOKENS

Keywords → int, while

Identifiers → main, total [letter or _ as first char] Max 31 length

Constants → 10, 20

Strings → "Hello!"

Special Symbols → {}, (), ""

Operators → +, -, /, *

→ VARIABLES

| | TYPE | DESCRIPTION |
|---|---|---|
| 1) | Char (Signed) | 1 Byte (-128, 127) |
| | (Unsigned) | 1 Byte (0, 255) |
| 2) | Int (Signed) | 2 Byte (-32,768, 32767) |
| | (unsigned) | 2 Byte (0, 65,535) |
| 3) | Long Int | 4 Byte (-2,147,483,648 to 2,147,483,647) |
| 4) | Float | 4 Bytes |
| 5) | Double | 8 bytes |
| 6) | Long double | 16 bytes |

Class Fun

Date | Page No.

# Operators

| Arithmetic | → Increment | → Relational |
|---|---|---|
| + → addition | ++ → Increase value by 1 | == → Checks if values are = |
| - → Subtraction | -- → Decrease value by 1 | != → Not equal to |
| * → Multiplication | | > or < → G.T or L.T |
| / → Division | | <= or >= → GTE or LTE |
| % → Remainder | | |

## Assignment

= → assigns value
+= → Add AND
-= → Subtract AND
*= → Multiply AND
/= → Divide AND
<<= → Left shift AND
>>= → Right shift AND
&= → Bitwise AND
^= → Bitwise Exclusive AND
|= → Bitwise Inclusive AND

## Bitwise Operators

& → Binary AND
| → Binary OR
^ → Bitwise XOR (If & only if) (Only one should be true)
~ → Binary Ones Complement
<< → Binary Left Shift
>> → Binary Right Shift

* We solve (*, /, %) from Left to right & then will solve (+, -) from Left to Right

## Logical Operator

&& → Logical AND
|| → Logical OR
! → Logical NOT

## Conditional Operators (Ternary)

(Condition ? True_Value : False_Value);

# BIT CONVERSION

→ BINARY TO DECIMAL

$(1100\mathring{0}1)_2 \rightarrow (\ )_{10}$

$\Rightarrow (2^0 \cdot 1) + (2^5 \cdot 1) + (2^6 \cdot 1) = 1 + 32 + 64$

$= 97$

→ DECIMAL TO BINARY

$(50)_{10} \rightarrow (\ )_2$

$\Rightarrow$

| | |
|---|---|
| 2 | 50 |
| 2 | 25 – 0 |
| 2 | 12 – 1 |
| 2 | 6 – 0 |
| 2 | 3 – 0 |
| 2 | 1 – 1 |
| 2 | 0 – 1 |

# VARIABLES

- Name given to a storage area that a user can write in.

- GLOBAL VARIABLE → Variables declare outside a function
- LOCAL VARIABLE → Variable declared inside a function.

- FORMAL PARAMETER → Variable declared in function brackets.
  ACTUAL PARAMETER → Variable provided during function call.

| | |
|---|---|
| %d → Integer | %d → Bool |
| %c → Character | %s → String |
| %f → Float | |
| %lf → Double | |

Class Fun
Date
Page No.

# If - Else Statements

## Syntax

```
if (condition){
    (statement);

}
else if (condition){
    (statement);

}
else {
    Statement ;
}
```

## * Ternary

```
(cond") ? (Code when True) : (Code when False);
```

## Switch Statements

```
Switch (Expression) {
    case (value1):
        Break;          # Executes if Expression value = Value1
    case (value2):
        Break;
    default :           // Executes if none of cases are
        break;          executed
```

Class Fun

Date | | | | |

Page No. | |

# Loops

## → For Loop

- Syntax

~~for (int i;~~

```
for (Initialization; Condition; Increment) {

    Statement;
}
```

- Flow of Control

→ Init statement is executed first & only once.

→ Condition is evaluated

→ Loop variable is updated according to increment.

→ Condition is evaluated again

## → While Loop

- Syntax

```
while (cond^n) {
    Statement;
}
```

## → Do-While

- Syntax

```
do                          // One iteration of loop runs, then
{                           condition is evaluated.
  Statements;
}
while (Cond^n);
```

CLASS FUN

Date | Page No.

# STATEMENTS

→ **BREAK**

Terminates loops and exits

→ **CONTINUE**

Skips to next iteration

→ **LABEL**

Label:

```
Label_Name:
        // code to be executed
goto Label_Name;
```

# FUNCTIONS:

- A block of code consisting of statements to be run.

→ **SYNTAX**

```
returntype Function_name(Parameters) {

    Body of Function
}
```

→ **CALL BY VALUES:**

A copy of variable is passed through function

→ **CALL BY REFERENCE**

An adress of variable is passed to func".

→ **BENEFITS OF FUNCTION**

- Reusability
- Modularity
- Efficient & easy to understand

→ **BUILT-IN FUNCTIONS**

- strcat(), gets(), puts(), malloc(), pow(), sqrt()

# STORAGE CLASSES

- Storage classes are used to specify scope & lifetime of a variable.

## 1) AUTO

- Default storage class for variables in function or block.
- Memory is allocated on the stack and deallocated when func^n exits.
- Default value is garbage.

## 2) REGISTER

- Used to store variable in register memory.
- Improves access time
- Stores limited data depending on processor.

## 3) STATIC

- Retain their value until the end of program.

## 4) EXTERN

- Used to declare global variable which can be defined in another file.

# RECURSION

- Technique in which function calls itself to solve a problem.

→ FACTORIAL OF A NUMBER

```
int factorial(int n){
    if (n==0){
        return 1;
    }
    else{
        return n* factorial(n-1);
    }
}
```

→ ADVANTAGE
- Reduces time complexity
- Makes function writing easier

→ DISADVANTAGE
- Uses alot of stack memory
- Generally slower

CLASS FUN

Date

Page No.

# ARRAYS

- Collection of homogenous datatypes
- Size must be a constant value
- Contiguous memory allocation
- First element of array is address of array.
- 2D arrays are stored in Row Major fashion.

→ SYNTAX

1) 1D ARRAY

```
datatype array_name[size];
array_name[index] = (*value); // Array value assignment
```

2) 2D ARRAY

```
datatype array_name[Row][Column]; // Declaration
```

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

→ PASSING ARRAY IN FUNCTION

```
Void func(arr[]); // Array must be mentioned during func declaration
func(arr); // Only name to passed when calling function
```

Class Fun

Date | Page No.

# STRINGS

- Array of characters
- Recognized in double quotes
- Terminated with ~~to~~ \0 [Null character].

## → SYNTAX

- #include <String.h>
- Char str_name[size] = "Value";
- Char str_name[size] = {'a', 'b', 'c', '\0'}; // \0 is null character

## → STRING FUNCTIONS

- strlen() → Computes length of string
- strcat() → Concats given string
- sizeof() → gives size of string
- strcpy(1,2) → copies content of second string into first
- str.cmp() → Returns '1' if strings are equal and '-1' if strings are unequal.