What is a Computer?

It is a machine which can accept data, process them, and output results.

Central Processing Unit (CPU)

Input Device → Central Processing Unit (CPU) → Output Device

Main Memory

Storage Peripherals
• CPU
  – All computations take place here in order for the computer to perform a designated task.
  – It has a large number of registers which temporarily store data and programs (instructions).
  – It has circuitry to carry out arithmetic and logic operations, take decisions, etc.
  – It retrieves instructions from the memory, interprets (decodes) them, and perform the requested operation.
• Main Memory
  – Uses semiconductor technology
    • Allows direct access
  – Memory sizes in the range of 256 Mbytes to 4 Gbytes are typical today.
  – Some measures to be remembered
    • $1 \text{ K} = 2^{10} (= 1024)$
    • $1 \text{ M} = 2^{20} (= \text{one million approx.})$
    • $1 \text{ G} = 2^{30} (= \text{one billion approx.})$

• **Input Device**
  – Keyboard, Mouse, Scanner, Digital Camera

• **Output Device**
  – Monitor, Printer

• **Storage Peripherals**
  – Magnetic Disks: hard disk, floppy disk
    • Allows direct (semi-random) access
  – Optical Disks: CDROM, CD-RW, DVD
    • Allows direct (semi-random) access
  – Flash Memory: pen drives
    • Allows direct access
  – Magnetic Tape: DAT
    • Only sequential access
Typical Configuration of a PC

• CPU: Pentium IV, 2.8 GHz
• Main Memory: 512 MB
• Hard Disk: 80 GB
• Floppy Disk: Not present
• CDROM: DVD combo-drive
• Input Device: Keyboard, Mouse
• Output Device: 17” color monitor
• Ports: USB, Firewire, Infrared
How does a computer work?

• Stored program concept.
  – Main difference from a calculator.

• What is a program?
  – Set of instructions for carrying out a specific task.

• Where are programs stored?
  – In secondary memory, when first created.
  – Brought into main memory, during execution.
Number System :: The Basics

• We are accustomed to using the so-called decimal number system.
  – Ten digits :: 0,1,2,3,4,5,6,7,8,9
  – Every digit position has a weight which is a power of 10.

• Example:
  234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0

  250.67 = 2 \times 10^2 + 5 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} \\
  \quad + 7 \times 10^{-2}
• A digital computer is built out of tiny electronic switches.
  – From the viewpoint of ease of manufacturing and reliability, such switches can be in one of two states, ON and OFF.
  – A switch can represent a digit in the so-called binary number system, 0 and 1.

• A computer works based on the binary number system.
Concept of Bits and Bytes

• Bit
  – A single binary digit (0 or 1).

• Nibble
  – A collection of four bits (say, 0110).

• Byte
  – A collection of eight bits (say, 01000111).

• Word
  – Depends on the computer.
  – Typically 4 or 8 bytes (that is, 32 or 64 bits).
• A $k$-bit decimal number
  – Can express unsigned integers in the range $0$ to $10^k - 1$
    • For $k=3$, from 0 to 999.

• A $k$-bit binary number
  – Can express unsigned integers in the range $0$ to $2^k - 1$
    • For $k=8$, from 0 to 255.
    • For $k=10$, from 0 to 1023.
Classification of Software

• Two categories:
  
1. Application Software
   • Used to solve a particular problem.
   • Editor, financial accounting, weather forecasting, etc.

2. System Software
   • Helps in running other programs.
   • Compiler, operating system, etc.
Computer Languages

• Machine Language
  – Expressed in binary.
  – Directly understood by the computer.
  – Not portable; varies from one machine type to another.
    • Program written for one type of machine will not run on another type of machine.
  – Difficult to use in writing programs.
Contd.

• Assembly Language
  – Mnemonic form of machine language.
  – Easier to use as compared to machine language.
    • For example, use “ADD” instead of “10110100”.
  – Not portable (like machine language).
  – Requires a translator program called assembler.

![Diagram showing the process of translating assembly language to machine language using an assembler.](image-url)
• Assembly language is also difficult to use in writing programs.
  – Requires many instructions to solve a problem.

• Example: Find the average of three numbers.

  MOV A,X ; A = X
  ADD A,Y ; A = A + Y
  ADD A,Z ; A = A + Z
  DIV A,3 ; A = A / 3
  MOV RES,A ; RES = A

  In C,
  
  RES = (X + Y + Z) / 3
High-Level Language

• Machine language and assembly language are called low-level languages.
  – They are closer to the machine.
  – Difficult to use.

• High-level languages are easier to use.
  – They are closer to the programmer.
  – Examples:
    • Fortran, Cobol, C, C++, Java.
  – Requires an elaborate process of translation.
    • Using a software called *compiler*.
  – They are portable across platforms.
Contd.
To Summarize

• Assembler
  – Translates a program written in assembly language to machine language.

• Compiler
  – Translates a program written in high-level language to machine language.
Operating Systems

• Makes the computer easy to use.
  – Basically the computer is very difficult to use.
  – Understands only machine language.

• Operating systems make computers easy to use.

• Categories of operating systems:
  – Single user
  – Multi user
    • Time sharing
    • Multitasking
    • Real time
Contd.

• Popular operating systems:
  – DOS: single-user
  – Windows 2000/XP: single-user multitasking
  – Unix: multi-user
  – Linux: a free version of Unix

• The laboratory class will be based on Linux.

• Question:
  – How multiple users can work on the same computer?
Contd.

• Computers connected in a network.
• Many users may work on a computer.
  – Over the network.
  – At the same time.
  – CPU and other resources are shared among the different programs.
    • Called time sharing.
    • One program executes at a time.
Multiuser Environment

Computer Network


User 1 User 2 User 3 User 4 User 4

Printer
Basic Programming Concepts
Some Terminologies

• Algorithm / Flowchart
  – Should be independent of the programming language.

• Program
  – A translation of the algorithm/flowchart into a form that can be processed by a computer.
  – Typically written in a high-level language like C, C++, Java, etc.
Variables and Constants

• Most important concept for problem solving using computers.
• All temporary results are stored in terms of variables and constants.
  – The value of a variable can be changed.
  – The value of a constant do not change.
• Where are they stored?
  – In main memory.
• How does memory look like (logically)?
  – As a list of storage locations, each having a unique address.
  – Variables and constants are stored in these storage locations.
  – Variable is like a *house*, and the name of a variable is like the *address* of the house.
    • Different people may reside in the house, which is like the *contents* of a variable.
Every variable is mapped to a particular memory address.
Variables in Memory

Instruction executed

Time

X = 10
X = 20
X = X + 1
X = X * 5

Memory location allocated to a variable X

10
20
21
105
Variables in Memory (contd.)

Instruction executed

<table>
<thead>
<tr>
<th>Time</th>
<th>Variable</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 20</td>
<td></td>
<td>20</td>
<td>?</td>
</tr>
<tr>
<td>Y = 15</td>
<td></td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>X = Y + 3</td>
<td></td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>Y = X / 6</td>
<td></td>
<td>18</td>
<td>3</td>
</tr>
</tbody>
</table>
Data types

• Three common data types used:
  – Integer :: can store only whole numbers
    • Examples: 25, -56, 1, 0
  – Floating-point :: can store numbers with fractional values.
    • Examples: 3.14159, 5.0, -12345.345
  – Character :: can store a character
Data Types (contd.)

• How are they stored in memory?
  – Integer ::
    • 16 bits
    • 32 bits
  – Float ::
    • 32 bits
    • 64 bits
  – Char ::
    • 8 bits (ASCII code)
    • 16 bits (UNICODE, used in Java)

Actual number of bits varies from one computer to another
Problem solving

• Step 1:
  – Clearly specify the problem to be solved.

• Step 2:
  – Draw flowchart or write algorithm.

• Step 3:
  – Convert flowchart (algorithm) into program code.

• Step 4:
  – Compile the program into object code.

• Step 5:
  – Execute the program.
Flowchart: basic symbols

Computation

Input / Output

Decision Box

Start / Stop
Contd.

Flow of control

Connector
Example 1: *Adding three numbers*

START

READ A, B, C

S = A + B + C

OUTPUT S

STOP
Example 2: Larger of two numbers

Start

Read X, Y

Is X > Y?

Yes

Output X

Stop

No

Output Y

Stop
Example 3: *Largest of three numbers*

```
START
READ X, Y, Z

IS X > Y?
YES LAR = X
NO IS LAR > Z?
YES OUTPUT LAR
NO OUTPUT Z

STOP
```
Example 4: *Sum of first N natural numbers*

```
START
READ N
SUM = 0
COUNT = 1
SUM = SUM + COUNT
COUNT = COUNT + 1
IS
COUNT > N?
YES
OUTPUT SUM
NO
SUM = SUM + COUNT
COUNT = COUNT + 1
STOP
```
Example 5: \( \text{SUM} = 1^2 + 2^2 + 3^2 + N^2 \)

```
START
READ N
SUM = 0
COUNT = 1
SUM = SUM + COUNT*COUNT
COUNT = COUNT + 1
IS COUNT > N?
NO
YES
OUTPUT SUM
STOP
```
Example 6: \( \text{SUM} = 1.2 + 2.3 + 3.4 + \text{to N terms} \)

```
START
READ N
SUM = 0
COUNT = 1
SUM = SUM + COUNT * (COUNT+1)
COUNT = COUNT + 1
IS COUNT > N?
YES OUTPUT SUM
NO
STOP
```
Example 7: Computing Factorial

```
START
READ N
PROD = 1
COUNT = 1
PROD = PROD * COUNT
COUNT = COUNT + 1
IS COUNT > N?
YES
OUTPUT PROD
STOP
NO
```

Example 8: Computing $e^x$ series up to $N$ terms

START

READ $X$, $N$

TERM = 1
SUM = 0
COUNT = 1

SUM = SUM + TERM
TERM = TERM * X / COUNT

COUNT = COUNT + 1

IS COUNT $>$ N?

NO

YES

OUTPUT SUM

STOP

START

READ $X$, $N$

TERM = 1
SUM = 0
COUNT = 1

SUM = SUM + TERM
TERM = TERM * X / COUNT

COUNT = COUNT + 1

IS COUNT $>$ N?

NO

YES

OUTPUT SUM

STOP
Example 9: Computing $e^x$ series up to 4 decimal places

```
START
READ X
TERM = 1
SUM = 0
COUNT = 1
SUM = SUM + TERM
TERM = TERM * X / COUNT
COUNT = COUNT + 1
IS TERM < 0.0001?
YES
OUTPUT SUM
STOP
NO
```

Flowchart:
- START
- READ X
- TERM = 1, SUM = 0, COUNT = 1
- SUM = SUM + TERM
- TERM = TERM * X / COUNT
- COUNT = COUNT + 1
- IS TERM < 0.0001?
- YES
- OUTPUT SUM
- STOP
- NO

Example 10: Roots of a quadratic equation

\[ ax^2 + bx + c = 0 \]

TRY YOURSELF
### Example 11: Grade computation

| MARKS ≥ 90 | 89 ≥ MARKS ≥ 80 | 79 ≥ MARKS ≥ 70 | 69 ≥ MARKS ≥ 60 | 59 ≥ MARKS ≥ 50 | 49 ≥ MARKS ≥ 35 | 34 ≥ MARKS | Ex | A | B | C | D | P | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |
Grade Computation (contd.)

START

READ MARKS

MARKS ≥ 90? NO

YES

OUTPUT “Ex” STOP

MARKS ≥ 80? NO

YES

OUTPUT “A” STOP

MARKS ≥ 70? NO

YES

OUTPUT “B” STOP

A
MARKS ≥ 60? NO
YES OUTPUT “C” STOP
MARKS ≥ 50? NO
YES OUTPUT “D” STOP
MARKS ≥ 35? NO
YES OUTPUT “P” STOP
NO OUTPUT “F” STOP
Programming in C
Introduction to C

• C is a general-purpose, structured programming language.
  – Resembles other high-level structured programming languages, such as Pascal and Fortran-77.
  – Also contains additional features which allow it to be used at a lower level.

• C can be used for applications programming as well as for systems programming.

• There are only 32 keywords and its strength lies in its built-in functions.

• C is highly portable, since it relegated much computer-dependent features to its library functions.
History of C

• Originally developed in the 1970’s by Dennis Ritchie at AT&T Bell Laboratories.
  – Outgrowth of two earlier languages BCPL and B.

• Popularity became widespread by the mid 1980’s, with the availability of compilers for various platforms.

• Standardization has been carried out to make the various C implementations compatible.
  – American National Standards Institute (ANSI)
  – GNU
Structure of a C program

• Every C program consists of one or more functions.
  – One of the functions must be called *main*.
  – The program will always begin by executing the main function.

• Each function must contain:
  – A function *heading*, which consists of the function *name*, followed by an optional list of *arguments* enclosed in parentheses.
  – A list of argument *declarations*.
  – A *compound statement*, which comprises the remainder of the function.
Contd.

• Each compound statement is enclosed within a pair of braces: ‘{‘ and ‘}’
  – The braces may contain combinations of elementary statements and other compound statements.

• Comments may appear anywhere in a program, enclosed within delimiters ‘/*’ and ‘*/’.
  – Example:
    
    ```
    a = b + c;  /* ADD TWO NUMBERS */
    ```
Sample C program #1

```c
#include <stdio.h>
main()
{
    printf ("\n Our first look at a C program \n");
}
```

- Header file includes functions for input/output
- Main function is executed when you run the program. (Later we will see how to pass its parameters)
- Curly braces within which statements are executed one after another.
- Statement for printing the sentence within double quotes ("..."). ‘\n’ denotes end of line.

Our first look at a C program
#include <stdio.h>
main()
{
    int   a, b, c;
    a = 10;
    b = 20;
    c = a + b;
    printf ("\n The sum of %d and %d is %d\n", 
             a, b, c);
}

Integers variables declared before their usage.

Control character for printing value of a in decimal digits.

The sum of 10 and 20 is 30
```c
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if  ((a>b) && (a>c))  /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if  (b>c) /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```

Comments within /* .. */

Input statement for reading three variables from the keyboard

Conditional statement
Sample C program #4

Preprocessor statement.
Replace PI by 3.1415926 before compilation.

Example of a function
Called as per need from Main programme.

```c
#include <stdio.h>
#define PI 3.1415926

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a);  /* return result */
}

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);
    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("Area is %f \n", area);
}
```

Function called.
main() is also a function

```c
#include <stdio.h>
main()
{
    int a, b, c;
    a = 10;
    b = 20;
    c = a + b;
    printf("\n The sum of %d and %d is %d\n", a, b, c);
}
```
Desirable Programming Style

• Clarity
  – The program should be clearly written.
  – It should be easy to follow the program logic.

• Meaningful variable names
  – Make variable/constant names meaningful to enhance program clarity.
    • ‘area’ instead of ‘a’
    • ‘radius’ instead of ‘r’

• Program documentation
  – Insert comments in the program to make it easy to understand.
  – Never use too many comments.
Contd.

• Program indentation
  – Use proper indentation.
  – Structure of the program should be immediately visible.
#include <stdio.h>
define PI 3.1415926
/* Compute the area of a circle */

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}

main()
{
    float radius, area;
    float myfunc (float radius);

    scanf("%f", &radius);
    area = myfunc (radius);
    printf("\n Area is %f \n", area);
}
#include <stdio.h>
#define PI 3.1415926
/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);
    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a);  /* return result */
}
/include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c)) /* Composite condition check */
        printf("\n Largest is %d", a);
    else
        if (b>c) /* Simple condition check */
            printf("\n Largest is %d", b);
        else
            printf("\n Largest is %d", c);
}
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int   a, b, c;
    scanf("%d %d %d", &a, &b, &c);
    if  ((a>b) && (a>c))   /* Composite condition check */
        printf("\n Largest is %d", a);
    else
        if  (b>c)     /* Simple condition check */
            printf("\n Largest is %d", b);
        else
            printf("\n Largest is %d", c);
}
The C Character Set

• The C language alphabet:
  – Uppercase letters ‘A’ to ‘Z’
  – Lowercase letters ‘a’ to ‘z’
  – Digits ‘0’ to ‘9’
  – Certain special characters:
Identifiers and Keywords

• Identifiers
  – Names given to various program elements (variables, constants, functions, etc.)
  – May consist of *letters*, *digits* and the *underscore* (‘_’) character, with no space between.
  – First character must be a letter.
  – An identifier can be arbitrary long.
    • Some C compilers recognize only the first few characters of the name (16 or 31).
  – Case sensitive
    • ‘area’, ‘AREA’ and ‘Area’ are all different.
Contd.

• Keywords
  – Reserved words that have standard, predefined meanings in C.
  – Cannot be used as identifiers.
  – OK within comments.

<table>
<thead>
<tr>
<th>auto</th>
<th>break</th>
<th>case</th>
<th>char</th>
<th>const</th>
<th>continue</th>
<th>default</th>
<th>do</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>else</td>
<td>enum</td>
<td>extern</td>
<td>float</td>
<td>for</td>
<td>goto</td>
<td>if</td>
</tr>
<tr>
<td>int</td>
<td>long</td>
<td>register</td>
<td>return</td>
<td>short</td>
<td>signed</td>
<td>sizeof</td>
<td>static</td>
</tr>
<tr>
<td>struct</td>
<td>switch</td>
<td>typedef</td>
<td>union</td>
<td>unsigned</td>
<td>void</td>
<td>volatile</td>
<td>while</td>
</tr>
</tbody>
</table>
Valid and Invalid Identifiers

• Valid identifiers
  X
  abc
  simple_interest
  a123
  LIST
  stud_name
  Empl_1
  Empl_2
  avg_empl_salary

• Invalid identifiers
  10abc
  my-name
  “hello”
  simple interest
  (area)
  %rate
Data Types in C

int :: integer quantity
     Typically occupies 4 bytes (32 bits) in memory.

char :: single character
       Typically occupies 1 byte (8 bits) in memory.

float :: floating-point number (a number with a decimal point)
        Typically occupies 4 bytes (32 bits) in memory.

double :: double-precision floating-point number
Some of the basic data types can be augmented by using certain data type qualifiers:
- short
- long
- signed
- unsigned

Typical examples:
- short int
- long int
- unsigned int
Some Examples of Data Types

- **int**
  - 0, 25, -156, 12345, -99820
- **char**
  - 'a', 'A', '*', '/', '
- **float**
  - 23.54, -0.00345, 25.0
  - 2.5E12, 1.234e-5

E or e means “10 to the power of”
Constants

- Numeric Constants
  - integer
  - floating-point

- Character Constants
  - single character
  - string
Integer Constants

• Consists of a sequence of digits, with possibly a plus or a minus sign before it.
  – Embedded spaces, commas and non-digit characters are not permitted between digits.
• Maximum and minimum values (for 32-bit representations)
  Maximum :: 2147483647
  Minimum :: –2147483648
Floating-point Constants

• Can contain fractional parts.
• Very large or very small numbers can be represented.
  23000000 can be represented as 2.3e7
• Two different notations:
  1. Decimal notation
     25.0, 0.0034, .84, -2.234
  2. Exponential (scientific) notation
     3.45e23, 0.123e-12, 123E2

*e means “10 to the power of”*
Single Character Constants

• Contains a single character enclosed within a pair of single quote marks.
  – Examples :: ‘2’, ‘+’, ‘Z’

• Some special backslash characters
  ‘\n’    new line
  ‘\t’    horizontal tab
  ‘\’     single quote
  ‘\”’    double quote
  ‘\\’    backslash
  ‘\0’    null
String Constants

• Sequence of characters enclosed in double quotes.
  – The characters may be letters, numbers, special characters and blank spaces.

• Examples:
  “nice”, “Good Morning”, “3+6”, “3”, “C”

• Differences from character constants:
  – ‘C’ and “C” are not equivalent.
  – ‘C’ has an equivalent integer value while “C” does not.
Variables

• It is a data name that can be used to store a data value.

• Unlike constants, a variable may take different values in memory during execution.

• Variable names follow the naming convention for identifiers.
  – Examples :: temp, speed, name2, current
Example

```
int a, b, c;
char x;

a = 3;
b = 50;
c = a - b;
x = 'd';

b = 20;
a = a + 1;
x = 'G';
```

Variables

Constants
Declaration of Variables

• There are two purposes:
  1. It tells the compiler what the variable name is.
  2. It specifies what type of data the variable will hold.

• General syntax:
  
  data-type variable-list;

• Examples:
  
  int velocity, distance;
  int a, b, c, d;
  float temp;
  char flag, option;
A First Look at Pointers

• A variable is assigned a specific memory location.
  – For example, a variable speed is assigned memory location 1350.
  – Also assume that the memory location contains the data value 100.
  – When we use the name speed in an expression, it refers to the value 100 stored in the memory location.
    distance = speed * time;

• Thus every variable has an address (in memory), and its contents.
Adress and Content

```c
int speed;
speed = 100;
```

```
speed  \rightarrow  100
&speed \rightarrow  1350
```