Introduction to Computing [CSC01]

Ankush Acharyya

CSE Dept., NIT Durgapur

Introduction

Books

- **Programming with C**
  - Byron Gottfried, Schaum's Outlines Series, Tata McGraw-Hill

- **The C Programming Language**
  - Brian W Kernighan, Dennis M Ritchie, Prentice Hall India

- **Programming in ANSI C**
  - E. Balaguruswamy, Tata McGraw-Hill

About the course

Class Timings

- Wednesday [15:30 – 16:20]
- Thursday [14:40 – 15:30]
- Friday [13:50 – 14:40] Tutorial

Necessary Evils

- **Mid-Sem** --- To be announced
- **End-Sem** --- To be announced
- **Assignment 1**: To be announced
- **Assignment 2:** To be announced

Class webpage: **https://sites.google.com/site/acharyyankush/teaching/introduction-to-computing**

Relevant Information

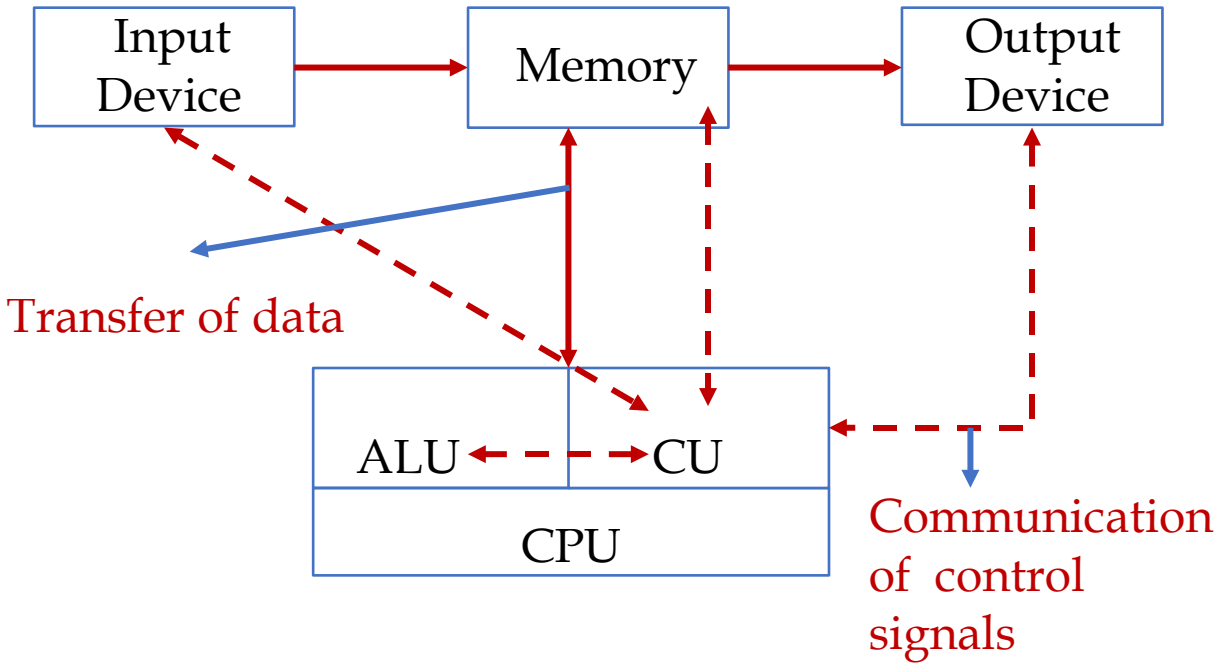
- Updates, announcement, course material will be disseminated through google classroom
  - Students are advised to join both theory and programming courses

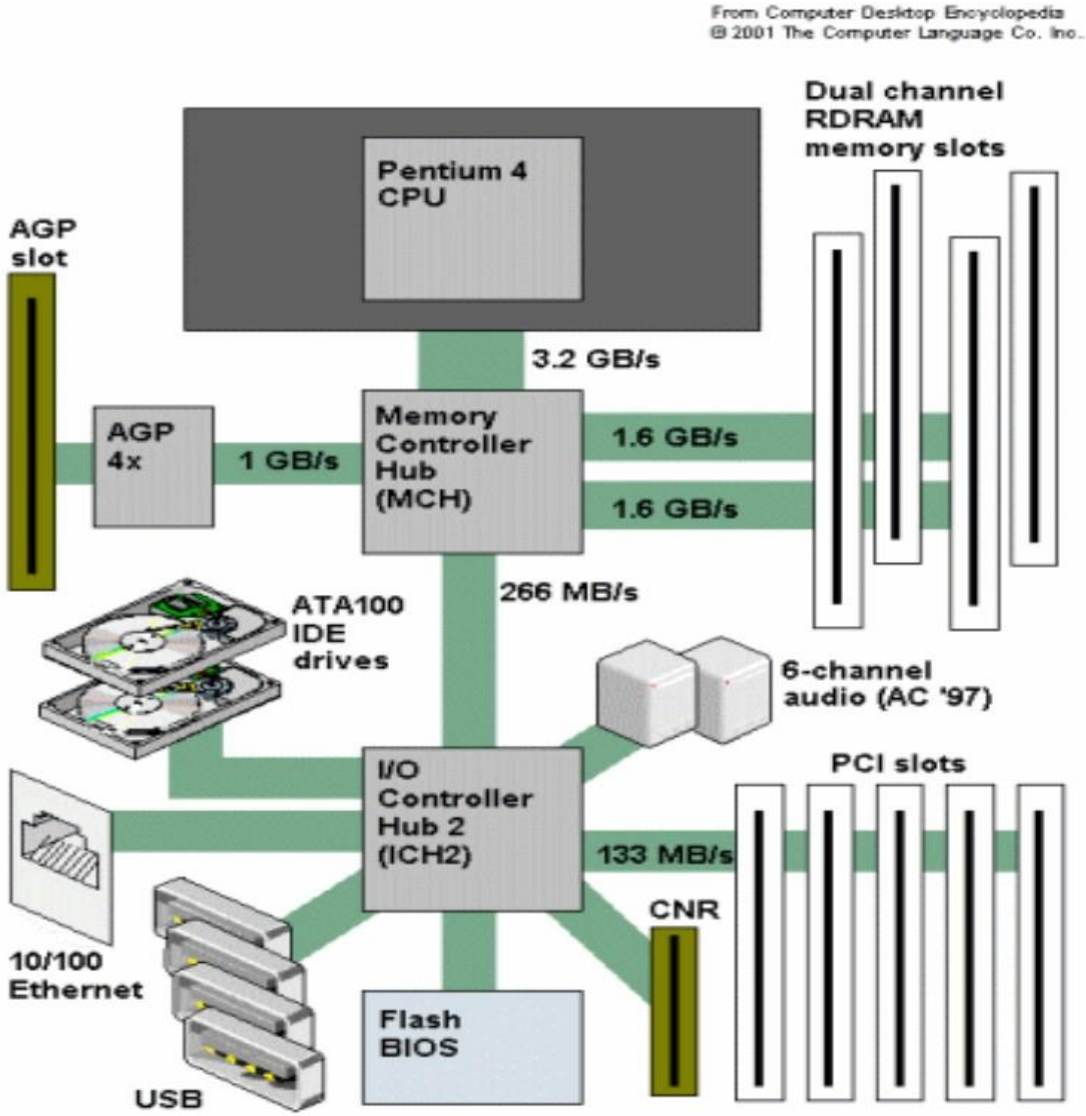
- **For beginners:**
  - Learn C through programming

- **Attendance:**
  - 75% mandatory
  - **Attendance will be based on serial number**

Architecture

- Typical system architecture for a desktop PC





Central Processing Unit (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 functional units (circuitry) to carry out arithmetic and logic operations
- It retrieves instructions from the memory, interprets (decodes) them, and performs the requested operation

Main Memory

- Uses semiconductor technology
  - Allows direct access

- Memory sizes in the range of 256 MegaBytes to 16 GigaBytes are typical today

- Some measures to be remembered
  - 1 K = 2^{10} (= 1024)
  - 1 M = 2^{20} (= one million approx.)
  - 1 G = 2^{30} (= one billion approx.)

Secondary Memory

- Provides permanent storage
- Data is stored in the form of sectors, tracks, cylinders
- Memory sizes in the range of 500 GB to 2 TB (TeraBytes) are typical today

I/O and Peripherals

- Input Device
  - Keyboard, Mouse, Scanner
- Output Device
  - Monitor, Printer

Computer Languages

- A **programming language** is a language specifically designed to express computations that can be performed on a computer

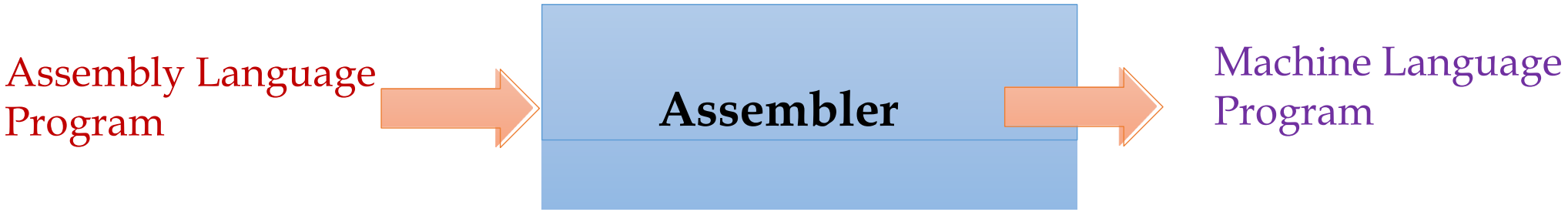
- Programming languages are used to ***express algorithms*** or ***as a mode of human communication***

Computer Languages

- **Machine Language [1st Gen]**
  - Expressed in binary
    - All the commands and data values are expressed using 1s and 0s
  - Directly understood by the computer
    - The code can run very fast and efficiently, since it is directly executed by the CPU
  - 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

Computer Languages

- **Assembly Language [2nd Gen]**
  - 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)
    - Assembly language is machine dependent
  - Requires a translator program called *assembler*



Computer Languages

- 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

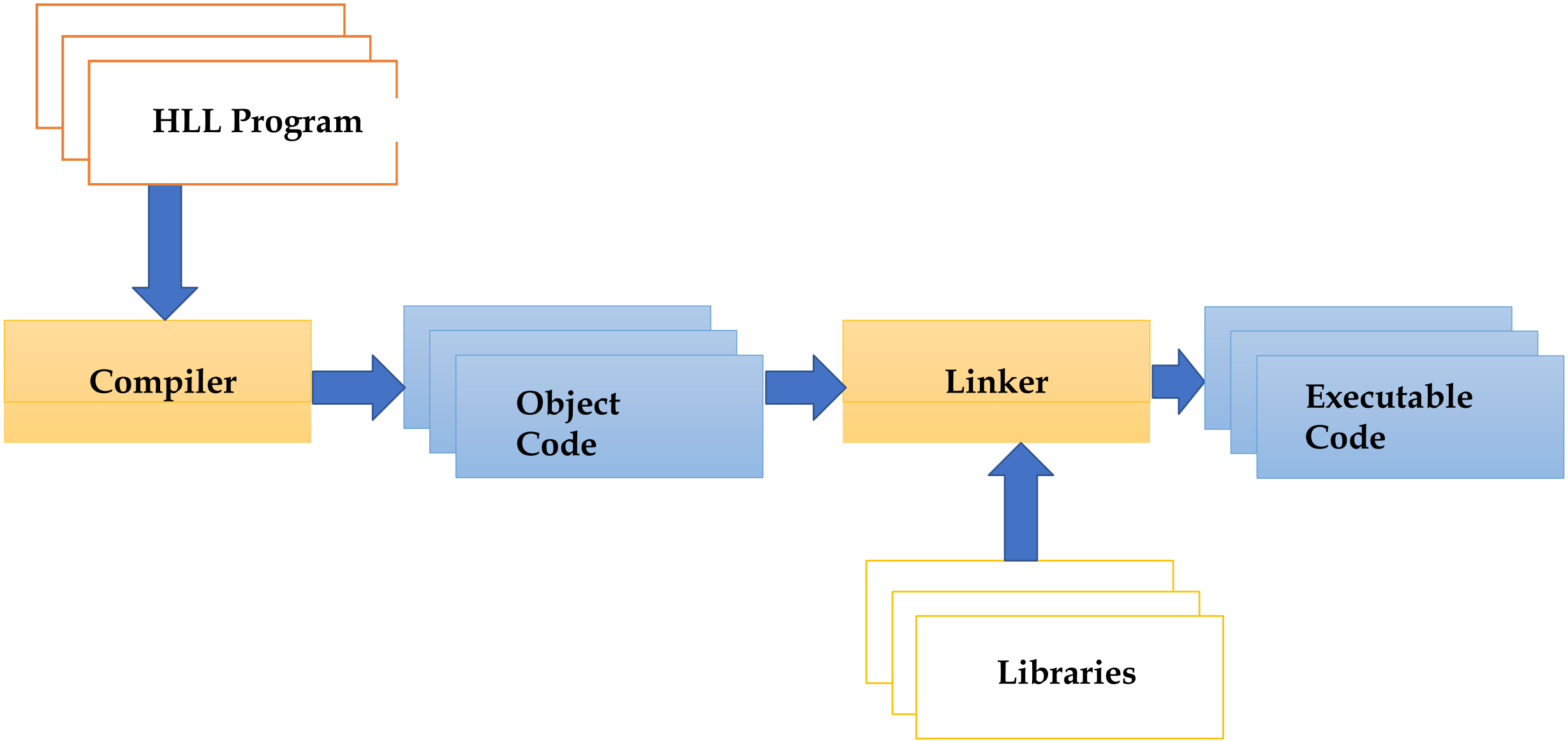
RES = (X + Y + Z)/3

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
  - Portable across machines
- 3rd Gen Languages
  - FORTRAN, PASCAL, COBAL, **C**, C++
- 4th and 5th Gen Languages [Non-procedural]
  - Prolog, SQL, Ruby
  - Needs to worry about what problems need to be solved and what conditions need to be met, *without worrying about how to implement a routine or algorithm to solve them*
- Requires an elaborate process of translation
  - Using a software called ***compiler/interpreter***

Programmer jargon

- **Some words that will be used a lot:**
  - **Source code:** The stuff you type into the computer. The program you are writing
  - **Compile(build):** Taking source code and making a program that the computer can understand
  - **Executable:** The compiled program that the computer can run
  - **Language:** The core part of C, central to writing C code
  - **Library:** Added functions for C programming which are bolted on to do certain tasks
  - **Header file:** Files ending in '.h' which are included at the start of source code


HLL Program
Compiler
Object Code
Linker
Executable Code
Libraries

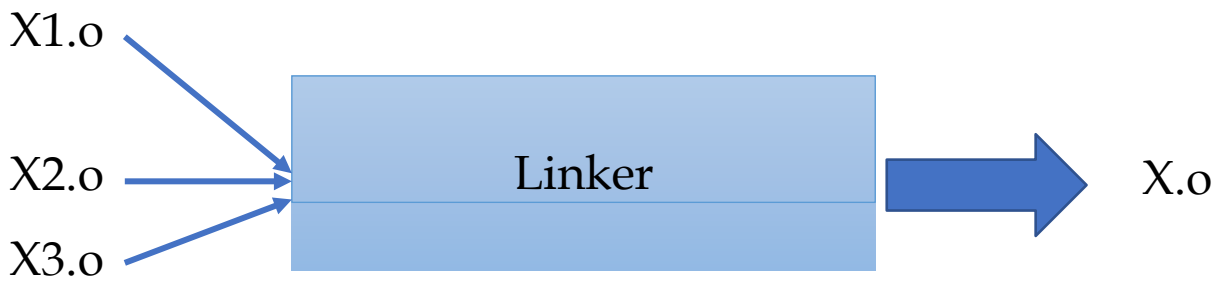
Linker and Loader

- **Compiler** converts source code written in a programming language (the *source language*) into machine language
- If code contains error compiler fails
  - Syntax Error
  - Logic Error

- **Interpreter** operates line by line
- If any line contains error execution halts there

Linker and Loader

- **Linker** is a program that combines object modules to form an executable



- **Loader** is a special type of program that copies programs from a storage device to main memory, where they can be executed
  - Transparent to user

User writes a program in C language (high-level language). The C compiler, compiles the program and translates it to assembly program (low-level language). An assembler then translates the assembly program into machine code (object). A linker tool is used to link all the parts of the program together for execution (executable machine code). A loader loads all of them into memory and then the program is executed

- **Interpreter**

An interpreter translates high-level language into low-level machine language. The difference lies in the way they read the source code or input. A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code, executes the whole program and may involve many passes. In contrast, an interpreter reads a statement from the input, converts it to an intermediate code, executes it, then takes the next statement in sequence. If an error occurs, an interpreter stops execution and reports it. Whereas a compiler reads the whole program even if it encounters several errors.

- **Assembler**

An assembler translates assembly language programs into machine code. The output of an assembler is called an object file, which contains a combination of machine instructions as well as the data required to place these instructions in memory.

- **Linker**

Linker is a computer program that links and merges various object files together in order to make an executable file. All these files might have been compiled by separate assemblers. The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

- **Loader**

Loader is a part of operating system and is responsible for loading executable files into memory and execute them. It calculates the size of a program (instructions and data) and creates memory space for it. It initializes various registers to initiate execution.

Operating System

- 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
    - Timesharing
    - Multitasking
    - Realtime

Operating System

- Popular Operating System
  - Windows 2000/XP/11 --- single user multitasking
  - Unix --- multiuser
  - Linux --- a free version of Unix

How does a program 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

Why to study C?

- C is *small* (only 32 keywords)
- C is *common* (lots of C code available)
- C is *stable* (the language doesn't change much)
- C is *quick running*
- C is the *basis for many other languages* (Java, C++, awk, Perl)
- *It may not feel like it but C is one of the easiest languages to learn*

C Tokens

C tokens are the basic buildings blocks in C language which are constructed together to write a C program.

Each and every smallest individual units in a C program are known as C tokens

```
#include<stdio.h>
void main()
{
int x, y, sum;
x = 6, y = 6;
sum = x + y;
printf ("Total = %d \n", sum);
}
```

main – identifier

{,}, (,) – delimiter

int – keyword

x, y, sum – identifier

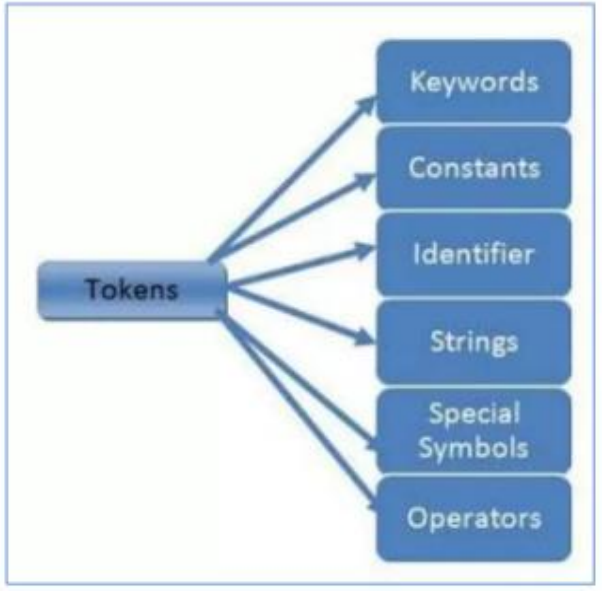
main, {, }, (,), int, x, y, sum – tokens

output

```
Total = 12
Press any key to continue . . .
```

In C Programming tokens are of six types. They are,

I. Keywords (ex: int, while,float),
II. Identifiers (ex: sum, total),
III. Constants (ex: 10, 20),
IV. Strings (ex: “ram”, “hello”),
V. Special symbols (ex: (), {}),
VI. Operators (ex: +, /,-,*)



Keywords of C [32]

Flow Control (6)	if, else, return, switch, case, default
Loops (5)	for, do, while, break, continue
Common types (5)	int, float, double, char, void
Structures (3)	struct, typedef, union
Counting and Sizing things (2)	enum, sizeof
Rare but useful types (7)	extern, signed, unsigned, long, short, static, const
Evil keywords which should be avoided (1)	goto
Wierdies (3)	auto, register, volatile

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:**

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

How to write a program?

- **Algorithm / Flowchart**
  - A step-by-step procedure for solving a particular problem
  - 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

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

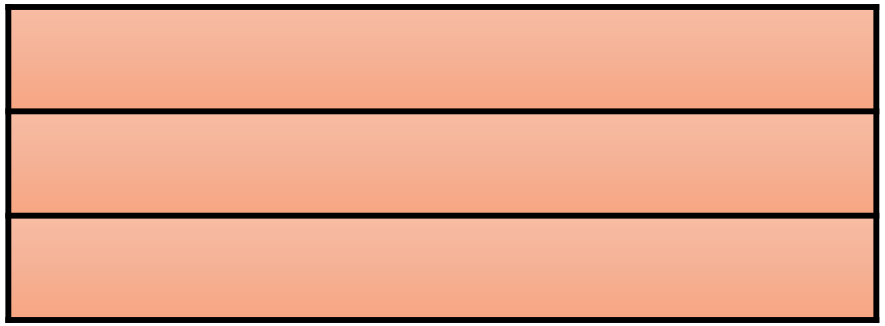
Variables and Constants

- **Most important concept for problem solving using computers**
- **All temporary results are stored in terms of variables**
  - The value of a variable can be changed
  - The value of a constant **do not change**
- **Where are they stored?**
  - In main memory

Variables and Constants

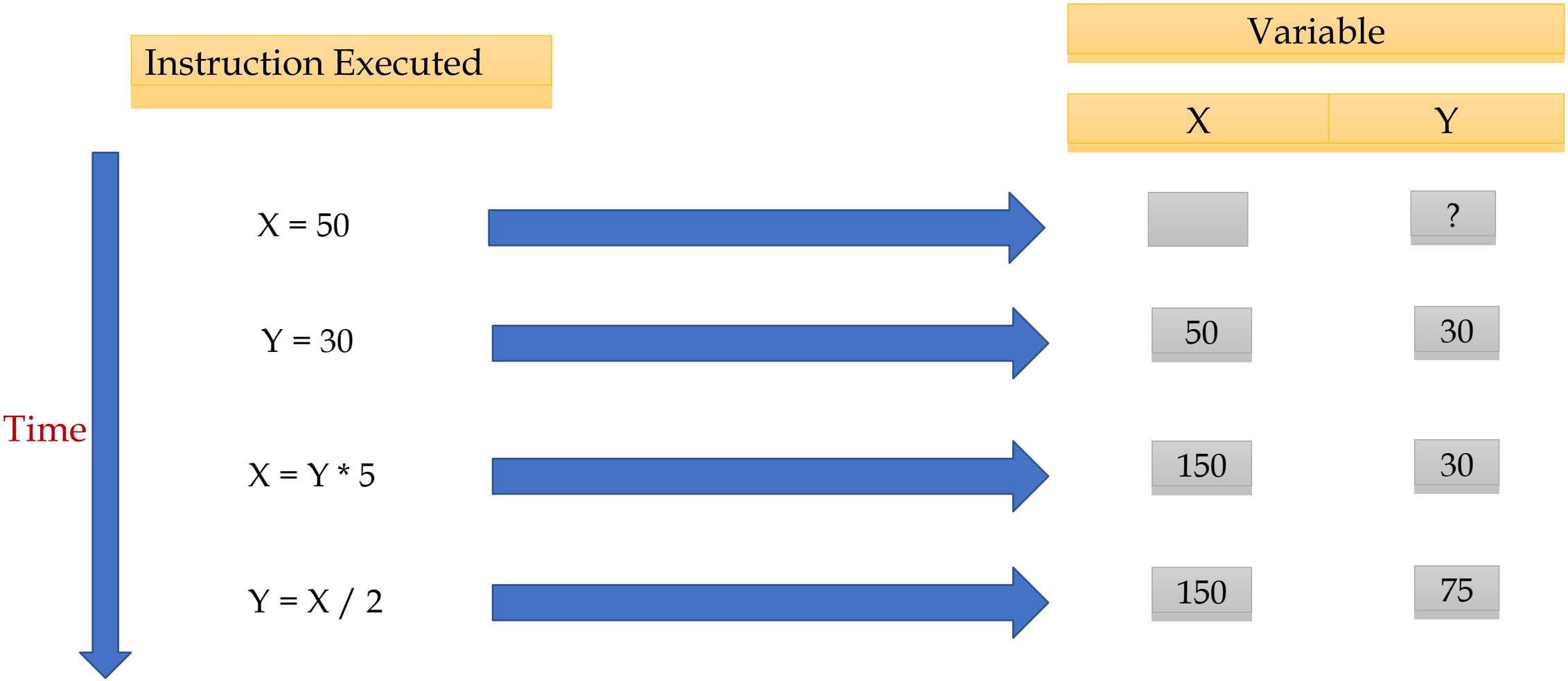
- **How does memory look like (logically)?**
  - As a list of storage locations, each having a unique address
  - Variables and constants are stored in these storage locations

Memory Map



Every variable is mapped to a particular memory address

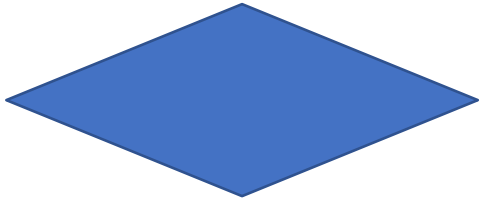
Variables in Memory



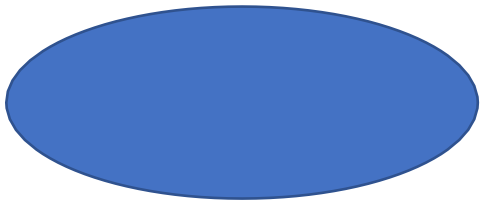
Flowchart: Basic Symbols

Computation/ Processing Statements

Input/ Output

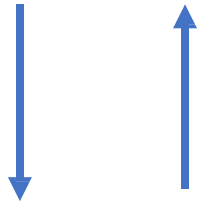


Decision Box



Start/ Stop

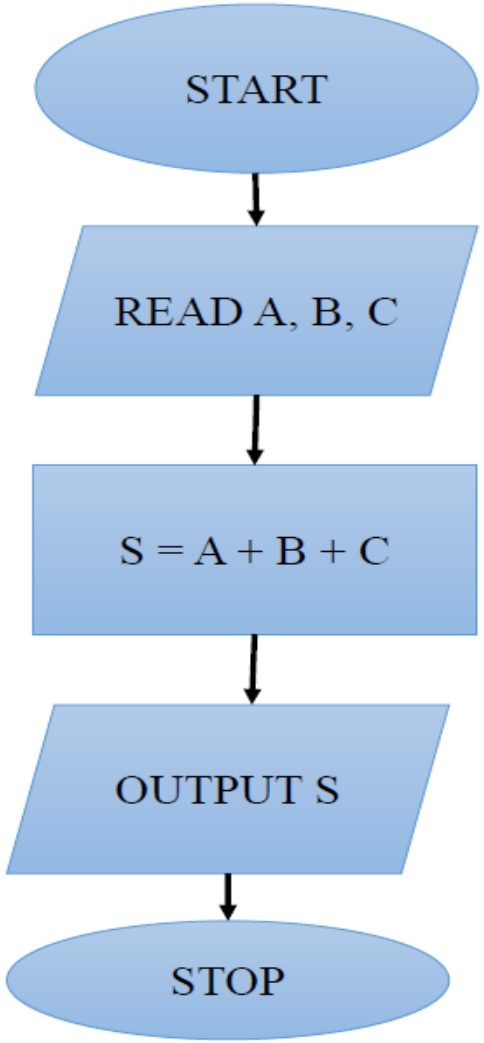
Flowchart: Basic Symbols



Flow of Control

Connector [*For longer flow chart*]

Example 1: Adding three numbers



Types of variable

- We must *declare* the *type* of every variable we use in C
- Every variable has a *type* (e.g. int) and a *name*
- This prevents some bugs caused by spelling errors (misspelling variable names)
- **Declarations of types should always be together at the top of main or a function (see later)**
- Other types are char, signed, unsigned, long, short and const (see later)

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 or underscore
- There **cannot be two successive underscores**
- Keywords **cannot be** used as identifiers
- **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

Valid and Invalid Identifiers

- **Valid Identifiers**

- X
- Abc
- simple_interest
- _emp_1
- a123
- LIST
- stud_name Empl_3
- Empl_2avg_empl_salary

- **Invalid Identifiers**

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

Datatypes

- **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**
  - Typically occupies 8 bytes (64 bits) of memory

- Represents basic data types
- This is for 64bit machines
- However, specifics will vary with different hardware and machine

Datatypes: Qualifier

- **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
  - unsigned char

Datatype – Memory [64 bit machine]

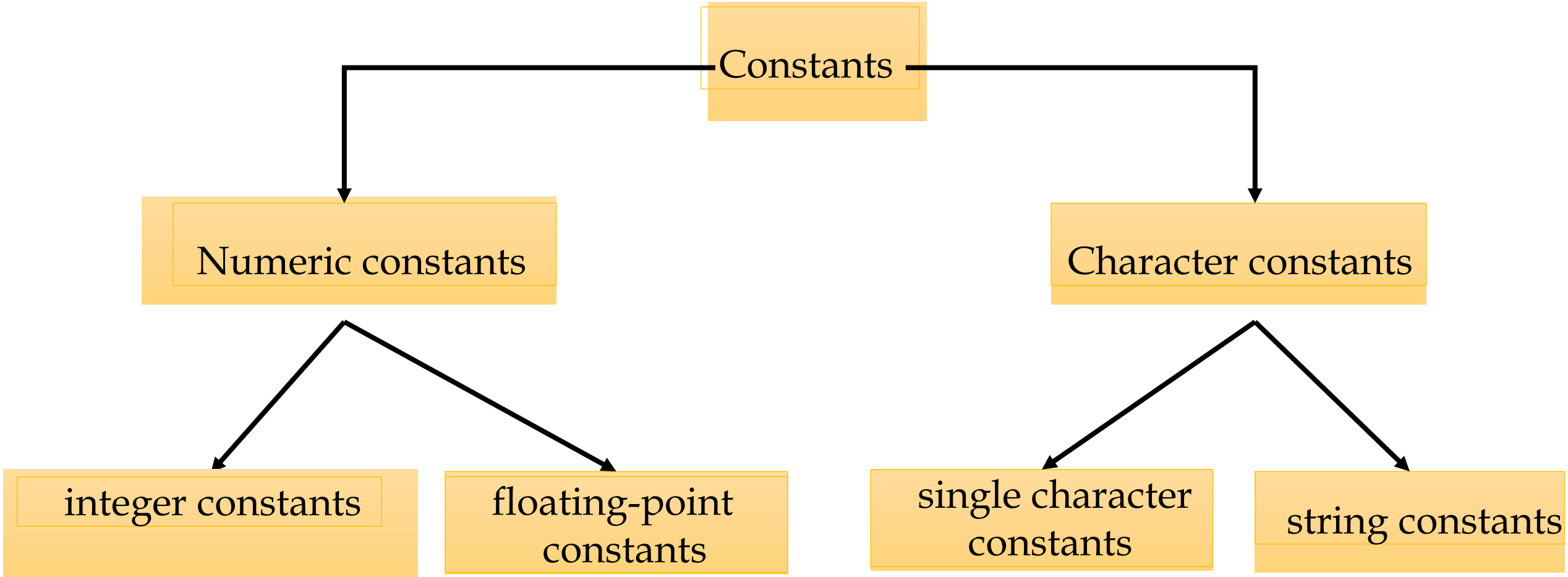
Qualifier	Data-type	Bytes
	int	4
short	int	2
long	int	8
	float	4
short	float	X
long	float	X
	double	8
short	double	X
long	double	16
	char	1
short	char	X
long	char	X

Datatype – Values [64 bit machine]

Qualifier	Data-type	Bytes
	int	-2^{31} to $2^{31}-1$
unsigned	int	0 to $2^{32}-1$
	char	-2^{7} to $2^{7}-1$
unsigned	char	0 to $2^{8}-1$
unsigned	float	X
unsigned	double	X

Constants

- Four basic types of constants in C



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
  - 25000000 can be represented as 2.5e7

e means "10 to the power of"

- Two different notations:
  - **Decimal notation**
    - 25.0, 0.0034, .84, -2.234
  - **Exponential (scientific) notation**
    - 3.45e23, 0.123e-12, 123E2

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

- Each Character constant has an associated ASCII value
  - ‘a’ :: 97
  - ‘A’:: 65

Escape Sequence

- Certain nonprinting characters, including backslash (\) and apostrophe (‘), can be expressed in terms of escape sequences

- An escape sequence always **begins with a backward slash** and is ***followed by one or more special characters***
  - **‘\n’** --- new line
  - **‘\t’** --- horizontal tab
  - **‘\’’** --- single quote
  - **‘\”’** --- double quote
  - **‘\\’** --- backslash
  - **‘\0’** --- null

Escape Sequence

Character	Escape Sequence	ASCII value
bell (alert)	\a	7
backspace	\b	8
horizontal tab	**\t**	**9**
vertical tab	\v	11
newline	**\n**	**10**
form feed	\f	12
carriage return	\r	13
double quote(“)	**\”**	**34**
apostrophe/single quote	**\’**	**39**
backslash	\\	92
null	**\0**	**0**

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