CS11001/CS11002
Programming and Data Structures
(PDS) (Theory: 3-0-0)

Class Teacher: Pralay Mitra
Jayanta Mukhopadhyay
Soumya K Ghosh

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Tentative Syllabus

Introduction to digital computers

Basic programming constructs

- Variables and simple data types
- Assignments
- Input/output
- Conditions and branching
- Loops and iteration
- Iterative searching and sorting algorithms
- Programming Examples: Sorting … etc

Advanced programming constructs

- Functions and recursion
- Recursive sorting algorithms
- Arrays and strings
- Structures
- Pointers and dynamic memory allocation
- File Handling

Tentative Syllabus

Performance analysis of programs

Data structures

- Abstract data types
- Ordered lists
- Stacks and queues

Programming Language: C

Course Materials

Do not use books written on specific C compilers like Turbo C, gcc
Use any standard textbook on ANSI C

Some useful text books:

- ✓ Brian W. Kernighan and Dennis M. Ritchie
  The C Programming Language, Prentice Hall of India.
- ✓ E. Balaguruswamy
  Programming in ANSI C, Tata McGraw-Hill
- ✓ Byron Gottfried
  Schaum's Outline of Programming with C, McGraw-Hill
- ✓ Seymour Lipschutz,
  Data Structures, Schaum's Outline Series, Tata McGraw-Hill
- ✓ Ellis Horowitz, Satraj Sahni and Susan Anderson-Freed,
  Fundamentals of Data Strcutures in C, W. H. Freemn and Company

Course Materials

Web references:
http://cse.iitkgp.ac.in/~pds/

Some useful software:
http://cse.iitkgp.ac.in/~pds/software/

Notes:
http://cse.iitkgp.ac.in/~pds/notes/

Course related information and announcements:
http://cse.iitkgp.ac.in/~pds/semester/2016a/

Attendance in the classes is MANDATORY

Students having poor attendance will be penalized in terms of the final grade / deregistration.

Proxy in the attendance will be heavily penalized. Each proxy in the class will result in the deduction of 5 marks from total marks you obtained.

It is your responsibility to check no such attendance marked against you.

Course Facts for section 8,9,10

- **Sections: 8, 9, 10**
- **Class Room: V2**
- **Time Schedule:**
  Monday (8:00-9:55); Tuesday (12:00-12:55)
- **Class Teacher:**
  Pralay Mitra (pralay@cse.iitkgp.ernet.in)
- **Teaching Assistant (TA):**

Anupam Banerjee	(mr.anupambanerjee@gmail.com)
Dipannita Podder	(dip.237@gmail.com)
Susmija Reddy	(susmija.reddy@gmail.com)
Shantanu Ghatak	(shantanu.ghatak@yahoo.in)

Course Facts for sections 11, 12

- **Sections: 11, 12**
- **Class Room: NR121**
- **Time Schedule:**
  Wednesday (12:00-12:55); Thursday (11:00-11:55); Friday (9:00-9:55)
- **Class Teacher:**
  Pralay Mitra (pralay@cse.iitkgp.ernet.in)
  Jayanta Mukhopadhyay (jay@cse.iitkgp.ernet.in)
- **Teaching Assistant (TA):**

Bijju Kranthi Veduruparthi	(bijjuair@gmail.com)
K Sai Ram	(sairam.kasanagottu@gmail.com)
Soumabha Bhowmick	(soumabha.bhowmick@gmail.com)
Romil Roy	(romilroy@gmail.com)

Course Facts for sections 13, 14

- **Sections: 13, 14**
- **Class Room: NR222**
- **Time Schedule:**

  Monday (10:00-10:55); Wednesday (9:00-9:55); Thursday (10:00-10:55)
- **Class Teacher:**

  Soumya K Ghosh (skg@cse.iitkgp.ernet.in)
- **Teaching Assistant (TA):**

Shreya Ghosh	(shreya.cst@gmail.com)
Monidipa Das	(monidipadas@hotmail.com)
Saptarshi Pal	(palsreturn2@gmail.com)
Omprakash Chakraborty	(omchakrabarty@gmail.com)

Course Facts

Distribution of Marks:

Class Test 1:	10
Mid Semester Exam:	30
Class Test 2:	10
End Semester Exam:	50

Important Dates:

Class Test 1:	August 25, 2016, 7:00pm – 8:00pm
Class Test 2:	October 27, 2016, 7:00pm – 8:00pm
Mid-Semester :	September 13-22, 2016 (as per institute schedule)
End-Semester :	November 21-29, 2016 (as per institute schedule)

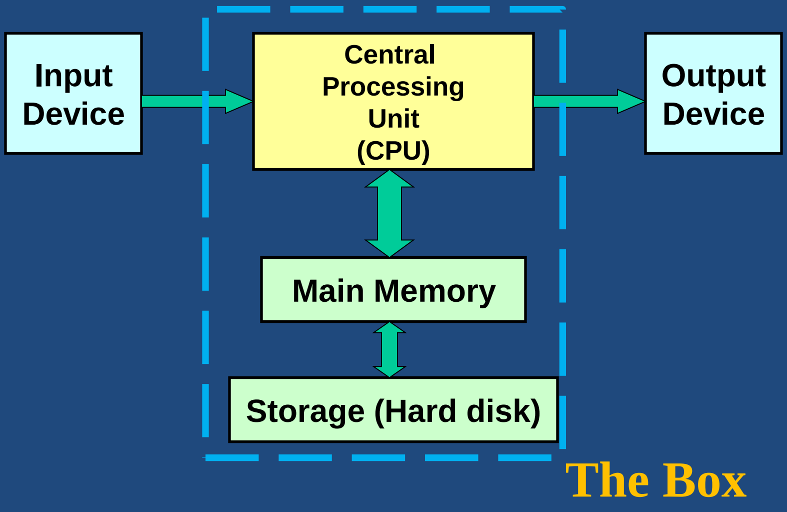
Tentative syllabus of tests:

CT1 syllabus:	Until Arrays and Strings
Mid Sem:	Until Functions, including recursion
CT2:	Until Arrays (2D)
End Sem:	Everything

Let us see

What is a Computer?

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



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 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.

```
while <power is on>
        1. fetch the instruction
                <decode it>
        2. execute the instruction
```

Main Memory

- Uses semiconductor technology
  - Allows direct access
  - RAM – Random Access Memory

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

Input Output (I/O)

- **Input Device**
  - Keyboard, Mouse, Scanner, Digital Camera
- **Output Device**
  - Monitor, Printer
- **Storage Peripherals**
  - Magnetic Disks: hard disk, floppy disk
    - Allows direct access
  - Optical Disks: CDROM, CD-RW, DVD
    - Allows direct access
  - Flash Memory: pen drives
    - Allows direct access
  - Magnetic Tape: DAT
    - Only sequential access

Typical Configuration of a PC

CPU:	Intel(R) Core(TM) i5-4570 CPU, 3.2 GHz
Main Memory:	4 GB
Hard Disk:	500 GB
Floppy Disk:	Not present
CDROM:	DVD RW combo-drive
Input Device:	Keyboard, Mouse
Output Device:	Monitor
Ports:	USB, Firewire, Infrared

Number System

- *Decimal number system*
  - Ten digits : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Every digit position has a weight : power of 10.

- *Example:*

$$234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

$$250.67 = 2 \times 10^2 + 5 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} + 7 \times 10^{-2}$$

Number system in digital computer

- 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 or OFF.
  - This can be represented by 0 (OFF) and 1 (ON).
- This suggests for a binary number system for a digital computer.

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).
- **Kilobyte (KB), MB, GB**
  - ?????
- **Word**
  - Depends on the computer.
  - Typically 4 or 8 bytes (that is, 32 or 64 bits).

Decimal and Binary

- 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.

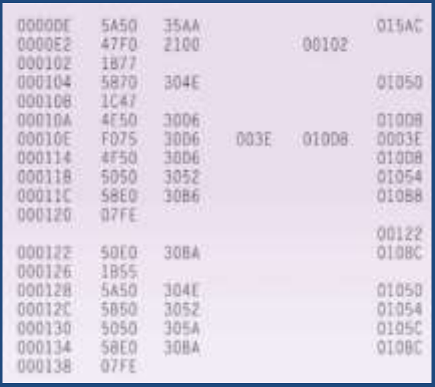
Computer Languages

- **Machine Level Language (MLL)**
  - 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.

Example: Machine Level Language

```
0110101001101010001011011001001010110010101010101001010101
0111110001010111110001101111011100010101001010101001101010100
0101010001001001011010100010100101110001100101010100100110
0011010101010111101011011111010010010001011010101010100000101
0011010100110101000101101100100101011001010101010100101010
1011111000101011111000110111101110001010100101010100110101010
0010101000100100101101010001010010111000110010101010010011
000110101010101111010110111110100100100010110101010100000010
0011010100110101000101101100100101011001010101010100101010
1011111000101011111000110111101110001010100101010100110101010
0010101000100100101101010001010010111000110010101010010011
000110101010101111010110111110100100100010110101010100000010
0011010100110101000101101100100101011001010101010100101010
1011111000101011111000110111101110001010100101010100110101010
0010101000100100101101010001010010111000110010101010010011
000110101010101111010110111110100100100010110101010100000010
0011010100110101000101101100100101011001010101010100101010
1011111000101011111000110111101110001010100101010100110101010
0010101000100100101101010001010010111000110010101010010011
000110101010101111010110111110100100100010110101010100000010
0011010100110101000101101100100101011001010101010100101010
1011111000101011111000110111101110001010100101010100110101010
```

Binary

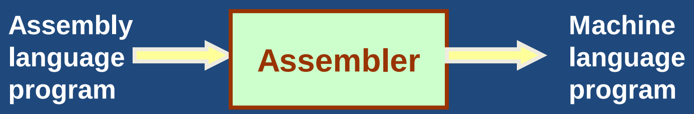


```
0000DE   5A50   35AA                   015AC
0000E2   47F0   2100          00102
000102   1B77
000104   5870   304E                   01050
000108   1C47
00010A   4E50   30D6                   010D8
00010E   F075   30D6   003E   010D8    0003E
000114   4F50   30D6                   010D8
000118   5050   3052                   01054
00011C   58E0   30B6                   010B8
000120   07FE
                                       00122
000122   50E0   30BA                   010BC
000126   1B55
000128   5A50   304E                   01050
00012C   5B50   3052                   01054
000130   5050   305A                   0105C
000134   58E0   30BA                   010BC
000138   07FE
```

Hexadecimal

Computer Languages

- **Assembly Level Language (ALL)**
  - – 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*.**



Assembly language program

Assembler

Machine language program

Example: Assembly Level Language

- 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
```

RES = (X + Y + Z) / 3

High-Level Language

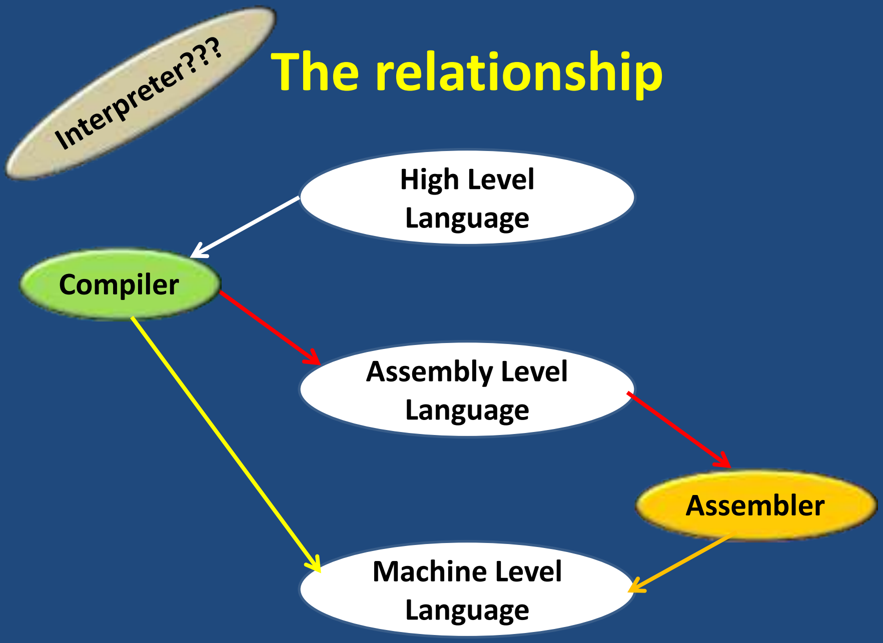
- Machine language and assembly language are called low-level languages.
  - They are closer to the machine.
  - Difficult to use.
- High-level languages are easier to use.
  - They are closer to the programmer.
  - Examples:
    - Fortran, Cobol, C, C++, Java.
  - Requires an elaborate process of translation.
    - Using a software called *compiler*.
  - They are portable across platforms.

Example: High Level Language

- Example: Find the average of three numbers.

RES = (X + Y + Z) / 3

The relationship



Classification of Software

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.

Operating Systems

- A system software to interface between computer hardware and software resources including application programs.
- Categories of operating systems:
  - – Single user
  - – Multi user
    - Time sharing
    - Multitasking
    - Real time

Operating Systems

- Popular operating systems:

– DOS:	single-user
– Windows:	single-user multitasking
– Unix:	multi-user
– Linux:	a free version of Unix

- The laboratory class will be based on Linux.

Programming in C

A complete C program

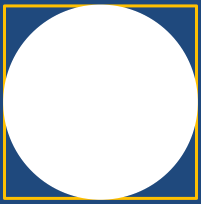
```
#include <stdio.h>

#define  PI    3.1416

double area_of_circle(float);

double area_of_circle (float   radius)
{
            return  PI*radius*radius;
}

int main()
{
            int   squareSide;
            double  area;
            scanf(“%d”, &squareSide);
            area= area_of_circle(squareSide/2);
            printf(“Area of the circle enclosing the square of side %d is: %f\n”,  squareSide, area);
            return 0;
}
```

Include header files

Declare global variables, constants and function prototypes

Function bodies

There must be a **main** function in any C program.



Universal starting point

```
#include <stdio.h>

int main()
{
            printf(“Hello World\n”);
            return 0;
}
```

A program must have an output.

Three steps to follow

1. Write a program and save it.
2. Compile the program using the correct compiler.
3. Execute the program

```
1. vi  hello.c
#include <stdio.h>

int main()
{
      printf("Hello World\n");
      return 0;
}

2. $ cc hello.c
$

3. $ ./a.out
Hello World
```

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)

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.

Structure of a C program

- 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 */

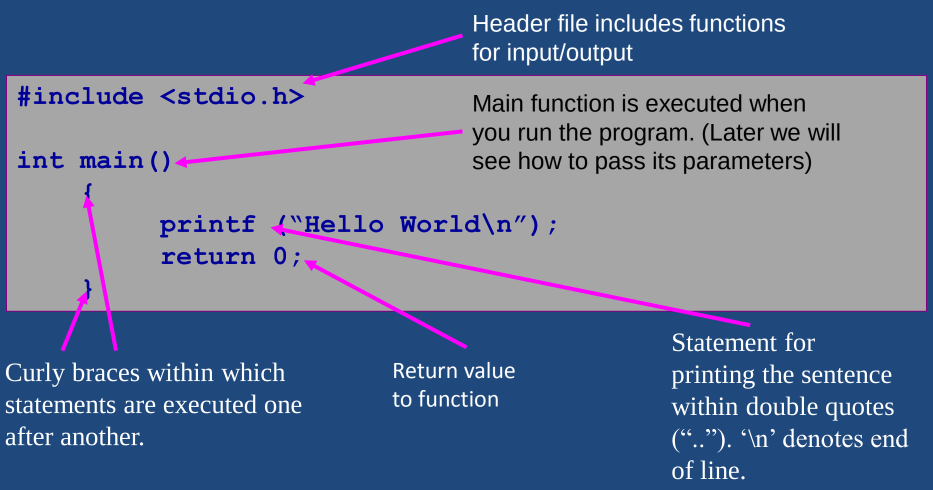
In and Out only

```
#include <stdio.h>

int main()
{
        int  n;
        scanf(“%d”,&n);
        printf(“%d”,n);
        return 0;
}
```

```
#include <stdio.h>

int main()
{
        int  n;
        scanf(“%d”,&n);
        printf(“%d”,n+n);
        return 0;
}
```

```
#include <stdio.h>

int main()
{
        int  n,m;
        scanf(“%d”,&n);       /* Read the value of n */
        m=n+n;
        printf(“%d”,m);
        return 0;
}
```

Universal starting point



```
#include <stdio.h>

int main()
    {
          printf (“Hello World\n”);
          return 0;
    }
```

Header file includes functions for input/output

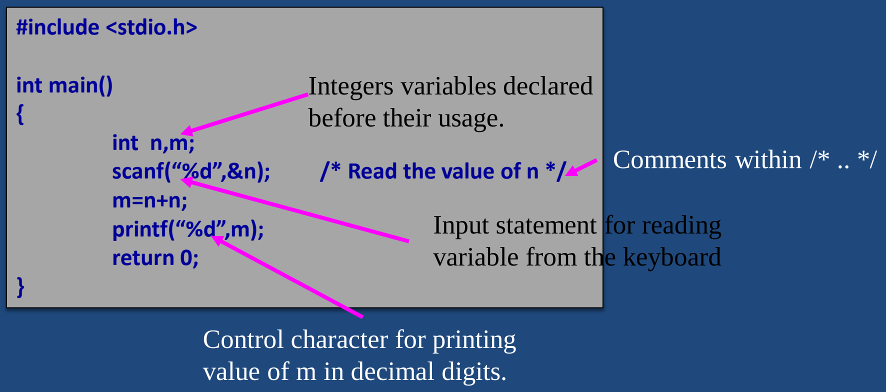
Main function is executed when you run the program. (Later we will see how to pass its parameters)

Curly braces within which statements are executed one after another.

Return value to function

Statement for printing the sentence within double quotes (“..”). ‘\n’ denotes end of line.

In and Out only



```
#include <stdio.h>

int main()
{
          int  n,m;
          scanf(“%d”,&n);      /* Read the value of n */
          m=n+n;
          printf(“%d”,m);
          return 0;
}
```

Integers variables declared before their usage.

Comments within /* .. */

Input statement for reading variable from the keyboard

Control character for printing value of m in decimal digits.

A complete C program

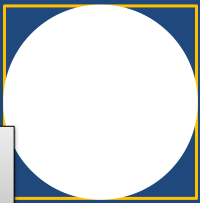
```
#include <stdio.h>

#define  PI    3.1416

double area_of_circle(float);

double area_of_circle (float   radius)
{
            return  PI*radius*radius;
}

int main()
{
            int   squareSide;
            double  area;
            scanf(“%d”, &squareSide);
            area= area_of_circle(squareSide/2);
            printf(“Area of the circle enclosing the square of side %d is: %f\n”,  squareSide, area);
            return 0;
}
```

Preprocessor statement. Replace PI by 3.1416 before compilation.

Example of a function called as per need from main programme.

main() is also a function

Function called



The *C* Character Set

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

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

Identifiers

- 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.

Keywords

- Keywords
  - Reserved words that have standard, predefined meanings in ***C***.
  - Cannot be used as identifiers.
  - OK within comments.
  - Standard ***C*** keywords:

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**

Valid and Invalid Identifiers

- Valid identifiers

  X
  abc
  simple_interest
  a123
  LIST
  stud_name
  Empl_1
  Empl_2
  avg_empl_salary

- Invalid identifiers

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

Basic 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

Precision refers to the number of significant digits after the decimal point.

Augmented Data Type

- 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

Integer type

Type	Storage size (in byte)	Value range
char	1	-128 to 127 or 0 to 255
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2 or 4	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4	0 to 65,535 or 0 to 4,294,967,295
short	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
long	4	-2,147,483,648 to 2,147,483,647
unsigned long	4	0 to 4,294,967,295

Integer type

unsigned char → 1 byte → 8 bits
→ 00000000 to 11111111 → 0 to 255

11111111 → $1\times2^7+ 1\times2^6+ 1\times2^5+ 1\times2^4+ 1\times2^3+ 1\times2^2+ 1\times2^1+ 1\times2^0$

signed char → 1 byte → 8 bits
→ 00000000 to 11111111 → -128 to 127

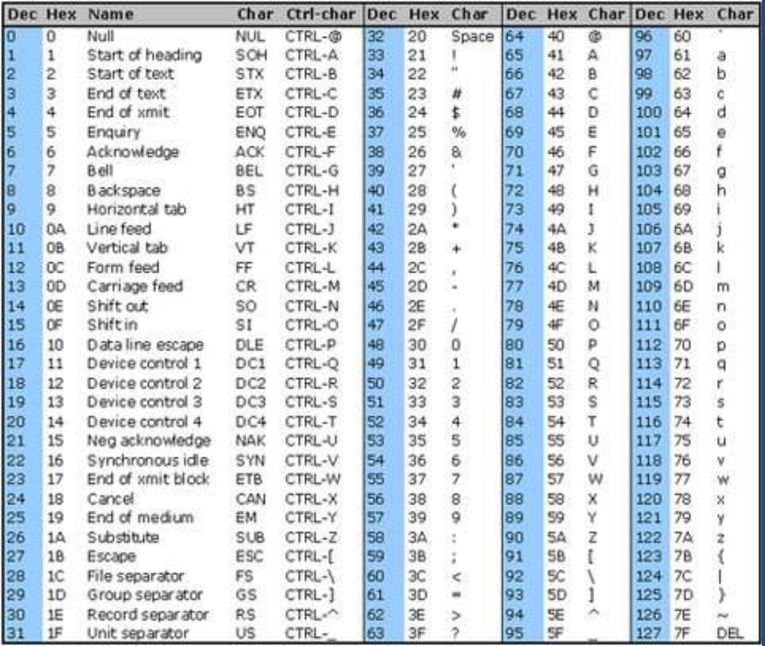
1111111 → $1\times2^6+ 1\times2^5+ 1\times2^4+ 1\times2^3+ 1\times2^2+ 1\times2^1+ 1\times2^0$

Floating-point type

Type	**Storage size (in byte)**	**Value range**	**Precision**
float	4	1.2E-38 to 3.4E+38	6 decimal places
double	8	2.3E-308 to 1.7E+308	15 decimal places
long double	10	3.4E-4932 to 1.1E+4932	19 decimal places

E or e means "10 to the power of"

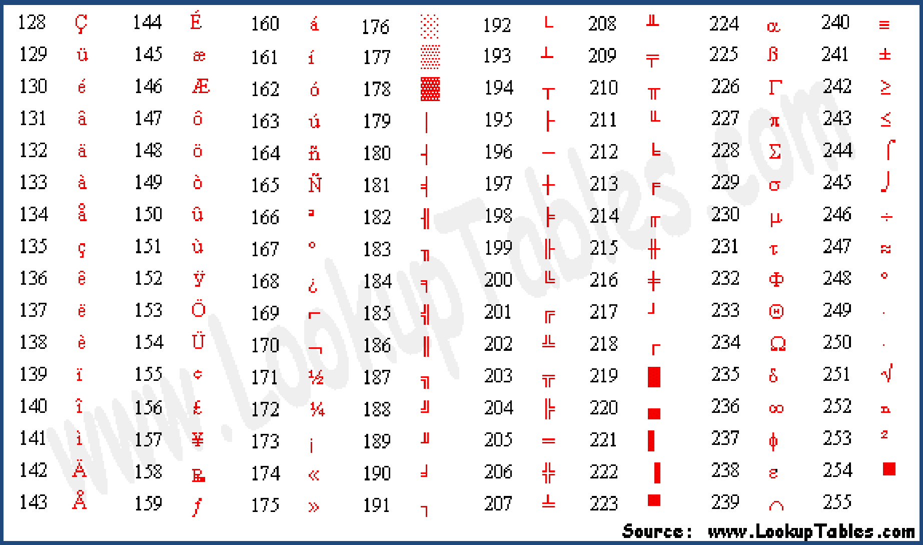
ASCII Table



Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Extended ASCII Table

(American Standard Code for Information Interchange)



128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	·
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Some Examples of Data Types

- int

  0, 25, -156, 12345, –99820
- char

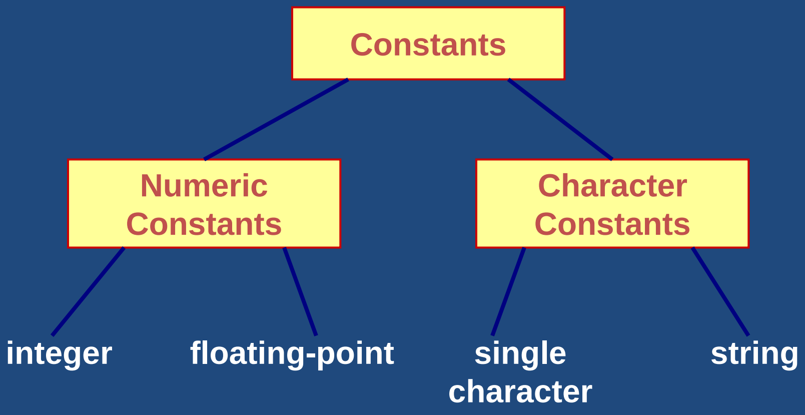
  ‘a’,   ‘A’,   ‘*’,   ‘/’,   ‘ ’
- float

  23.54, –0.00345, 25.0

  2.5E12, 1.234e-5

E or e means “10 to the power of”

Constants



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.

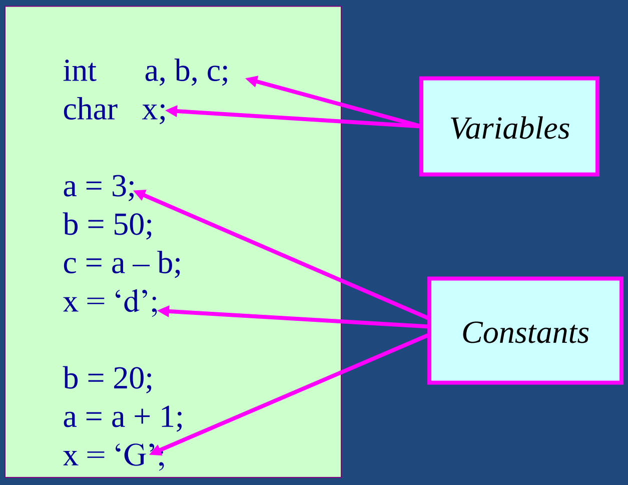
Variables

- 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.

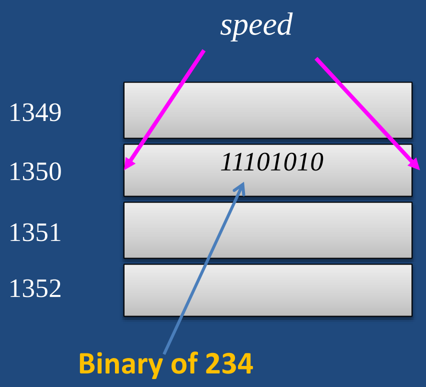
- General syntax:

```
data-type  variable-list;
```

- Examples:

```
int   velocity, distance;
int   a, b, c, d;
float  temp;
char  flag, option;
```

Address and Content



```
int speed;
speed=234;
```

speed → 234

&speed → 1350

Every variable has an address (in memory), and its contents.

Address and Content

- In *C* terminology, in an expression

  speed refers to the contents of the memory location.

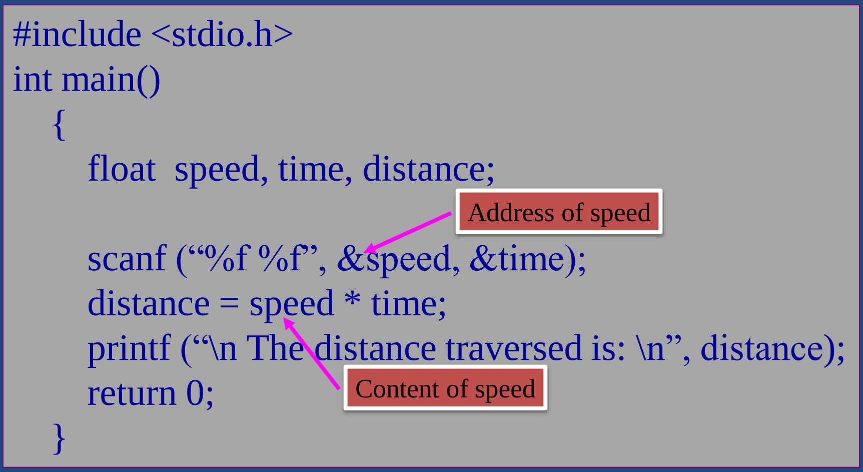
  &speed refers to the address of the memory location.

- Examples:

  printf (“%f %f %f”, speed, time, distance);

  scanf (“%f %f”, &speed, &time);

An Example



```
#include <stdio.h>
int main()
   {
      float  speed, time, distance;

      scanf (“%f %f”, &speed, &time);
      distance = speed * time;
      printf (“\n The distance traversed is: \n”, distance);
      return 0;
   }
```

Address of speed

Content of speed

Assignment Statement

- Used to assign values to variables, using the assignment operator (=).
- General syntax:

```
variable_name  =  expression;
```

- Examples:

```
velocity = 20;
b = 15;  temp = 12.5;
A = A + 10;
v = u + f * t;
s = u * t + 0.5 * f * t * t;
```

Advanced Assignment Statement

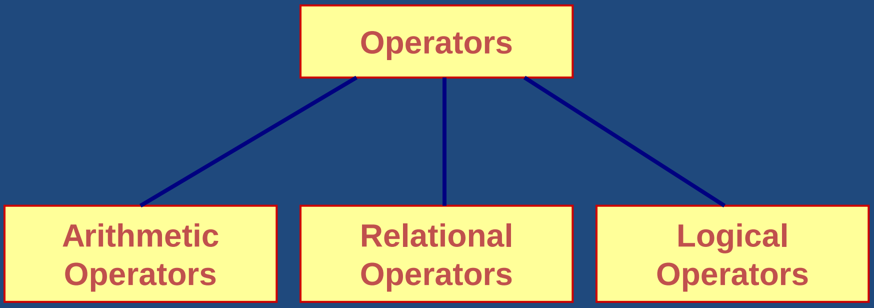
- Assignment during declaration

```
int   speed = 30;
char  flag = ‘y’;
```

- Multiple variable assignment

```
a = b = c = 5;
flag1 = flag2 = ‘y’;
speed = flow = 20.0;
```

Operators in Expressions



Operators

Arithmetic Operators

Relational Operators

Logical Operators

Arithmetic Operators

- Addition :: +
- Subtraction :: –
- Division :: /
- Multiplication :: *
- Modulus :: %

X= 25; Y=23;

X + Y	48
X – Y	2
X * Y	575
X / Y	?
X % Y	??

Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ( )
  2. Unary minus :: –5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction
- For operators of the same priority, evaluation is from left to right as they appear.
- Parenthesis may be used to change the precedence of operator evaluation.

Examples: Arithmetic expressions

v = u + f * t; → v = u+(f*t);

X = x * y / z → X = (x*y)/z

A = a + b – c * d / e → A = ((a+b)-((c*d)/e))

A = -b * c + d % e → A = (((-b)*c)+(d%e))

Integer Arithmetic

- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.
- Integer arithmetic always yields integer values.

Real Arithmetic

- Involving only real or floating-point operands (including double, long double).
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.

  A = 22/7*7*7 = (((22/7)*7)*7) = 153.86
  =(((22*7)/7)*7) = 154

- The modulus operator cannot be used with real operands.

Arithmetic – integer /real

- An expression contains only integer operands → Integer arithmetic will be performed.
- An expression contains only real operands → Real arithmetic will be performed.
- An expression contains integer and real both the operands → Real arithmetic will be performed.

Type casting

- A faulty reciprocal finder

```
#include <stdio.h>
int main ()
{
        int n;
        scanf("%d",&n);
        printf("%d\n",1/n);
        return 0;
}
```

The division 1/n is of integers (quotient).
The format %d is for printing integers.

Type casting

```
#include <stdio.h>
int main ()
{
        int n;
        scanf("%d",&n);
        printf("%f\n",1.0/n);
        return 0;
}
```

```
#include <stdio.h>
int main ()
{
        int n;
        float x;
        scanf("%d",&n);
        x=(float)1/n;
        printf("%f\n",x);
        return 0;
}
```

Type casting

Integer to real

```
        int a=10;
        float b;
        b=(float)a;
```

Real to integer

```
        int a;
        float b=3.14;
        a=(int)b;
```

Real to real

```
        float b;
        double c=3.14;
        b=(float)c;
```

Real to real

```
        float b;
        double c;
        c=22.0/7.0;
        b=(float)c;
```