

GE6151 COMPUTER PROGRAMMING

LECTURE 2

Basic features of c

Salient Features of C

C is a Structured programming language.

Programmer can concentrate only on the problem at hand and not about the machine.

C language claim to be machine independent.

C language is flexible, simple and easy to use.

C language is based on modular function concepts.

C is a middle level language.

Preceders of C

C language is derived from ALGOL (ALGOrithmic Language).

C language has both program efficiency and machine efficiency.

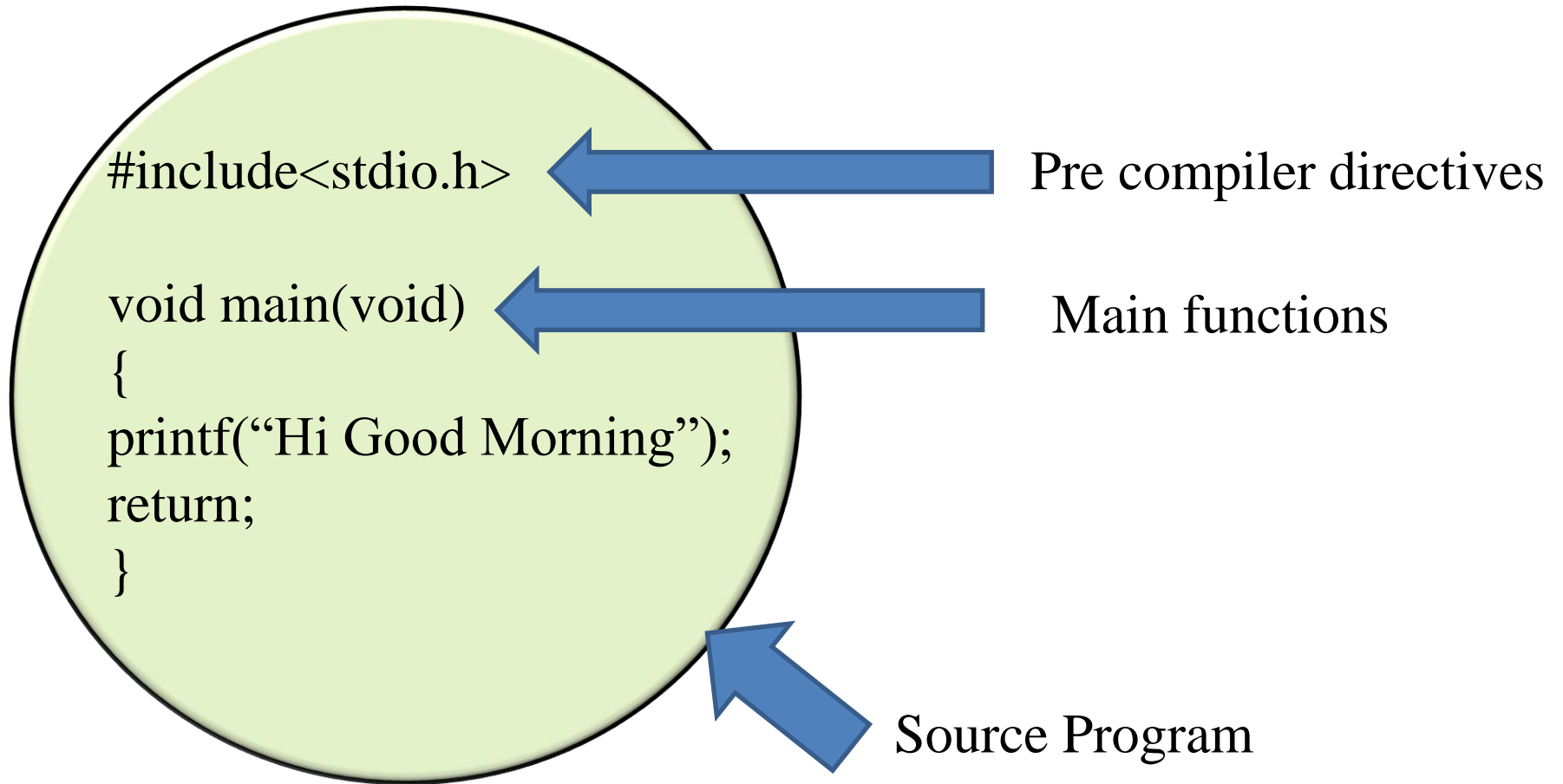
C language precedes 'B' Language which was developed by Ken Thomson.

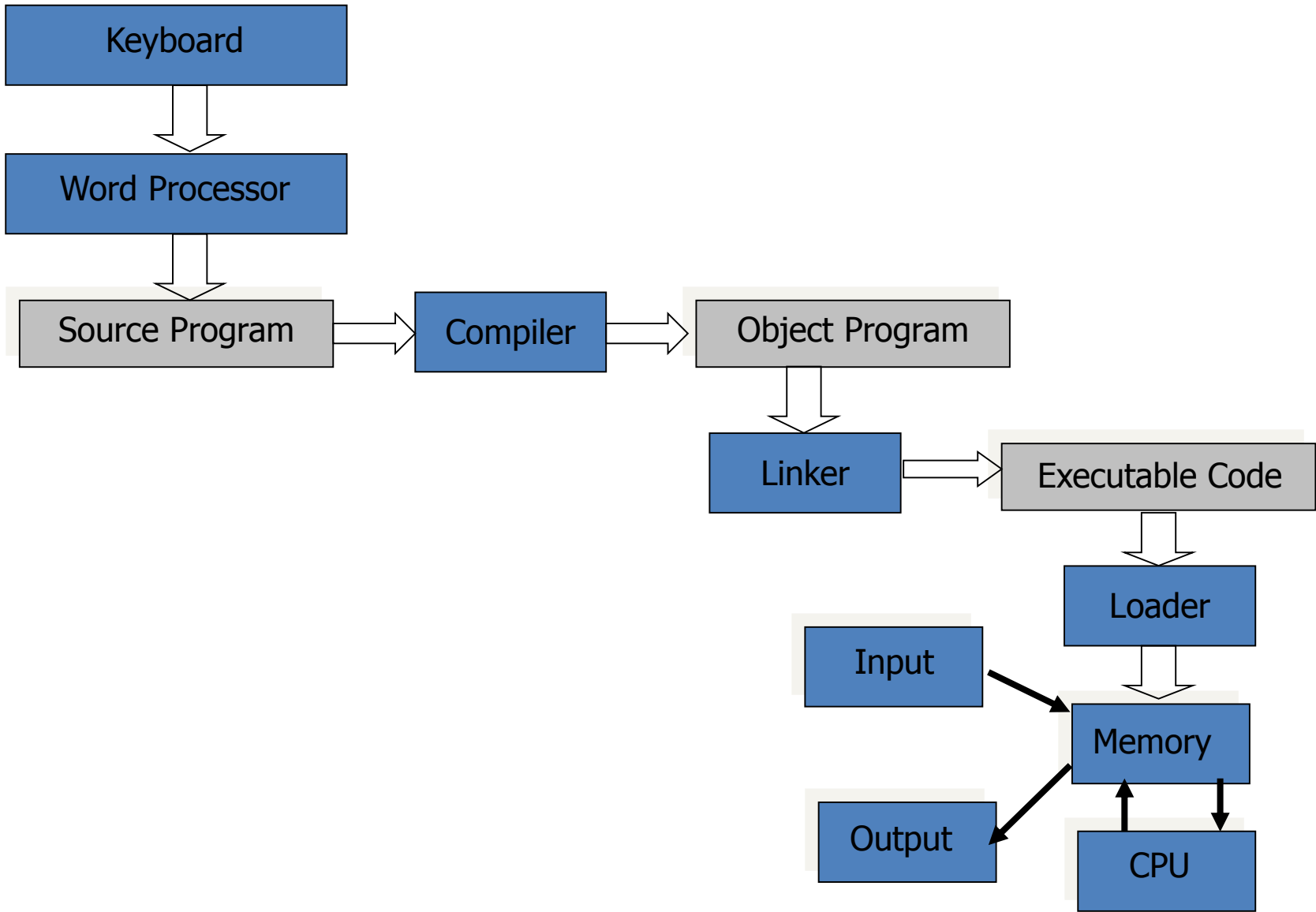
C language was developed by Dennis Ritchie with the main features derived from ALGOL, B and BCPL languages.

Who developed C?

C language was developed by Dennis Ritchie and Ken Thomson at At&T Bell Laboratories.

A Simple program in C





The original program written in a high level language is called the **source program**.

A **compiler** is a program that scans the source program and check whether the source program has followed the high-level language's syntax rule.

If the program is error free, it is converted into machine language instructions called **object program**. If the program contains any syntax error the compilers informs the same to the user.

The compiled program must be linked with operation codes provided by the high-level language developer using a linker program. When the program is **linked**, the object program receives the run time information such as memory addresses where variables and codes will be stored. The linker program provides a complete machine language program called executable code.

The **executable code** can be now loaded into the computer memory and a direction must be given to the CPU to begin the execution. As the program is executed, it takes the one or more **input** from the user and sends the **result** to the user.

PRE PROCESSOR DIRECTIVES

These are special instructions to the compiler.

They instruct the compiler how to prepare the program for compilation.

The most important pre processor directive is **#include** statement.

The **#include** command is used to include the **header files** at the time of compilation.

A **header file** contains declarations and macro definitions being shared between several source files. Header files are included in a source program to supply the definitions and declarations we need to invoke the system calls and libraries.

PRE PROCESSOR DIRECTIVES

You can create your own header files which may contain declarations for interfaces between the source files of your program. In C, the usual convention is to give header files names that end with `.h`.

Both user and system header files are included using the pre-processing directive `#include`. It has two variants:

`#include <file>`

This variant is used for system header files. It searches for a file named `file` in a standard list of system directories. You can prefix directories to this list with the `-I` option

`#include "file"`

This variant is used for header files of your own program. It searches for a file named `file` first in the directory containing the current file, then in the same directories used for `<file>`.

The standard C library is defined by the ANSI C standard and is composed of the functions, definitions and macros that are declared in fifteen header files. Some of the header files are:

```
#include<stdio.h>  
#include<stdlib.h>  
#include<math.h>  
#include<limits.h>  
#include<float.h>  
#include<string.h>  
#include<ctype.h>  
#include<time.h>
```

GLOBAL DECLARATIONS

This section defines the variables, functions that are visible or accessible to the whole program.

MAIN FUNCTION

Every C program should have a function named main.

Execution of a program begins only at main (It is the starting point).

The main function has two divisions namely: Local definition section and statement section

The definition section should be at the beginning of the main function.

The variables declared inside the main function are local to the main function and they are not visible to other functions.

The statement section contains the instructions to the computer to perform various data manipulation operations.

The key word main must be followed by parentheses.

The set of statements belonging to the main function are enclosed within a pair of braces.

CHARACTER SET OF C

Alphabets:

A,B,...,Z, a,b,...,z

Digits

0,1,2,3,4,5,6,7,8,9

Special Symbols

~	apostrophe	—	Under score
!	Exclamation	-	Minus
@	at the rate of	+	Plus
#	Hash	=	Equal to
%	Percentage	‘	Single Quote
^	Caps Mark	“	Double Quote
&	ampersand		Vertical bar
*	Star	/	Slash
(C open bracket	\	Back slash
)	C Closing bracket	:	Colon
{	Left curl bracket	;	Semi colon
}	Right curl bracket	.	Period
[left square bracket	,	Comma
]	Right Square Bracket	<	Less than
		>	Greater Than

Note: Symbols other than these are not allowed in C.

IDENTIFIERS

Identifiers are used to name data, variables and other objects in the program.

Good identifier names should be short and descriptive.

The identifier name must start with an **alphabetic character** or **underscore character**.

Must consist only alphabetic characters, digits or underscore.

First 31 characters of an identifier are significant.

The identifier must not be a C keyword.

C distinguishes between upper and lower case characters.

The system uses identifiers that have an underscore as the first character. For this reason it is advisable not to use the underscore as the first character.

Examples for Identifiers:

total
TOTAL
sum
SUM
member
cost
mark
student_id
i
j
K
sum_123
s1234
_total

The following are not
valid identifiers

total\$
1apple
apple=2
+apple
distance in km
long
short

Reserved words (32)

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

CONSTANTS

Data values that cannot be changed during program execution.

DATA TYPES

- ❖ The programs manipulate data. Data are stored in the computer's internal memory called "**cell**".
- ❖ A cell is a section of memory that can store one data item of a particular type at time.

STANDARD DATA TYPES OF C

void

int

char

float

Void Data Type

This data type has no value.

Used for assignment operation.

Can be used as a generic data type (in the case of pointers).

int DATA TYPE

- A number without a decimal point
- C supports three different sizes of int data type :
short int, int, long int
- The type also defines the size of the field in which the data is stores.
- The size of any data type can be found using the “sizeof” operator.
- For any machine the following relation is always true.
- $\text{sizeof}(\text{short int}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long})$
- Each integer type can be further defined as signed integer or unsigned integer.
- The signed integer can have positive or negative number while the unsigned integer can have only positive number.

Rules for Constructing Integer constant

1. An integer constant must have at least one digit.
2. It must not have a decimal point.
3. It may be positive or negative.
4. If no sign precedes an integer constant is assumed to be positive.
5. No commas or blanks are allowed within an integer constant.

EXAMPLE:

12 -12 2341 -12321 -98709L

76543LU

An integer constant can be represented as an octal or a hexadecimal or as a decimal integer constant

Example:

Decimal Integer Constants:

-12 123 123L 134LU

Octal Integer Constants:

An Octal integer is an integer constant formed with the numerals 0-7. A zero should prefix the octal integer constant .

Example:

-012 0123 0123L 0134LU

Hexadecimal Integer Constants:

An Octal integer is an integer constant formed with the numerals 0-9 and alphabets A to F. A zero and the letter 'x' or "X" should prefix the hexadecimal integer constant .

Example:

-0x12 0x123 0x123L 0x134LU

Type	Bytes	Minimum value	Maximum value
signed short int	2	-32768	32767
unsigned short int	2	0	65535
signed int (16 bit)	2	-32768	32767
unsigned int (16 bit)	2	0	65535
signed int (32 bit)	4	-2147483648	2147483647
unsigned int (32 bit)	4	0	4294967295
signed long int	4	-2147483648	2147483647
unsigned long int	4	0	4294967295

Note: To provide flexibility across different hardware platforms, C has a library function `limits.h` and `float.h` that contain the size information of integers and floats.

FLOATING POINT

A number with a decimal point.

C supports three floating point data type

float (Byte size 4) (f)

double (Byte size 8)

long double (Byte size 10) (L)

The default representation is double.

Example:

0.0	7.231	3.1416	Double
-2.6f	3.232f	7.23123f	float
3.121314123L	1.234234L		long double

CHARACTER CONSTANT

Symbols are given numerical values.

Symbols represented in ASCII form

Numerical values range is 0 to 127.

Enclose between two single quotes

Special characters will have a back slash

(known as escape character).

'A' = 65; 'Z' = 90;

'a' = 97; 'z' = 122;

'0' = 48; '1' = 49;

ESCAPE CHARACTERS

null character ‘\0’

alert ‘\a’

back space ‘\b’

horizontal tab ‘\t’

new line ‘\n’

vertical line ‘\v’

single quote ‘\’

double quote ‘\”’

back slash ‘\\’

ASCII Character Code

0	ct1	15	ctO	30	ct=	45	-	60	<	75	K	90	Z	105	i	120	x
1	ctA	16	ctP	31	ct-	46	.	61	=	76	L	91	[106	j	121	y
2	ctB	17	ctQ	32	Sp	47	/	62	>	77	M	92	\	107	k	122	z
3	ctC	18	ctR	33	!	48	0	63	?	78	N	93]	108	l	123	{
4	ctD	19	ctS	34	“	49	1	64	@	79	O	94	^	109	m	124	/
5	ctE	20	ctT	35	#	50	2	65	A	80	P	95	_	110	n	125	}
6	ctF	21	ctU	36	\$	51	3	66	B	81	Q	96	`	111	o	126	~
7	ctG	22	ctV	37	%	52	4	67	C	82	R	97	a	112	p	127	del
8	ctH	23	ctW	38	&	53	5	68	D	83	S	98	b	113	q		
9	ctI	24	ctX	39	'	54	6	69	E	84	T	99	c	114	r		
10	ctJ	25	ctY	40	(55	7	70	F	85	U	100	d	115	s		
11	ctK	26	ctZ	41)	56	8	71	G	86	V	101	e	116	t		
12	ctL	27	Esc	42	*	57	9	72	H	87	W	102	f	117	u		
13	Ret	28	ct<	43	+	58	:	73	I	88	X	103	g	118	v		
14	ctN	29	ct/	44	,	59	;	74	J	89	Y	104	h	119	w		

STRING CONSTANTS

A string constant is a sequence of zero or more characters enclosed in double quotes.

EXAMPLE:

“”

“A”

“UniMAP”

“GOOD DAY”

Note: ‘A’ and “A” are different

Data Type	Memory (Bytes)	Range	Format Specifier	Constant
Char	1	-128 to 127	%c	'c'
Signed Char	1	-128 to 127	%c	'c'
Unsigned Char	1	0 to 255	%c	'c'
Short int	2	-32768 to 32767	%d	12
Unsigned short int	2	0 to 65535	%d	12U
Int	4	-2147483648 to 2147483647	%ld	12
Unsigned int	4	0 to 4294967295	%lu	12U
Long int	4	-2147483648 to 2147483647	%ld	1234L
Unsigned long int	4	0 to 4294967295	%lu	1234LU
float	4	-3.4e-38 to 3.4 e38	%f %e	12.3f
double	8	-1.7e-308 to 1.7e308	%le %LG	1.23e-23
Long double	8	-1.7e-308 to 1.7e308	%le %LG	12.3e-303L
	10	-1.7e-4932 to 1.7 e 4932	%le %LG	12.3e-1303L

To know the memory size of a data type, sizeof operator can be used.

Example: sizeof(char) will give the size of a character data
sizeof(int) will give the size of a integer data

To know the maximum and minimum data values use the named constants available in the file limits.h and float.h

Example: THE max and min values are available in the following named constants.
CHAR_MAX, CHAR_MIN, UCHAR_MAX, SHRT_MAX, SHRT_MIN, USHRT_MAX,
LONG_MAX, LONG_MIN, ULONG_MAX, INT_MAX, INT_MIN, UINT_MAX, FLT_MAX,
FLT_MIN, DBL_MAX, DBL_MIN

LOGICAL DATA TYPE

- ❖ Represents only two values namely **True** and **False**.
- ❖ In 'C' zero represents false and any non zero value represents true.

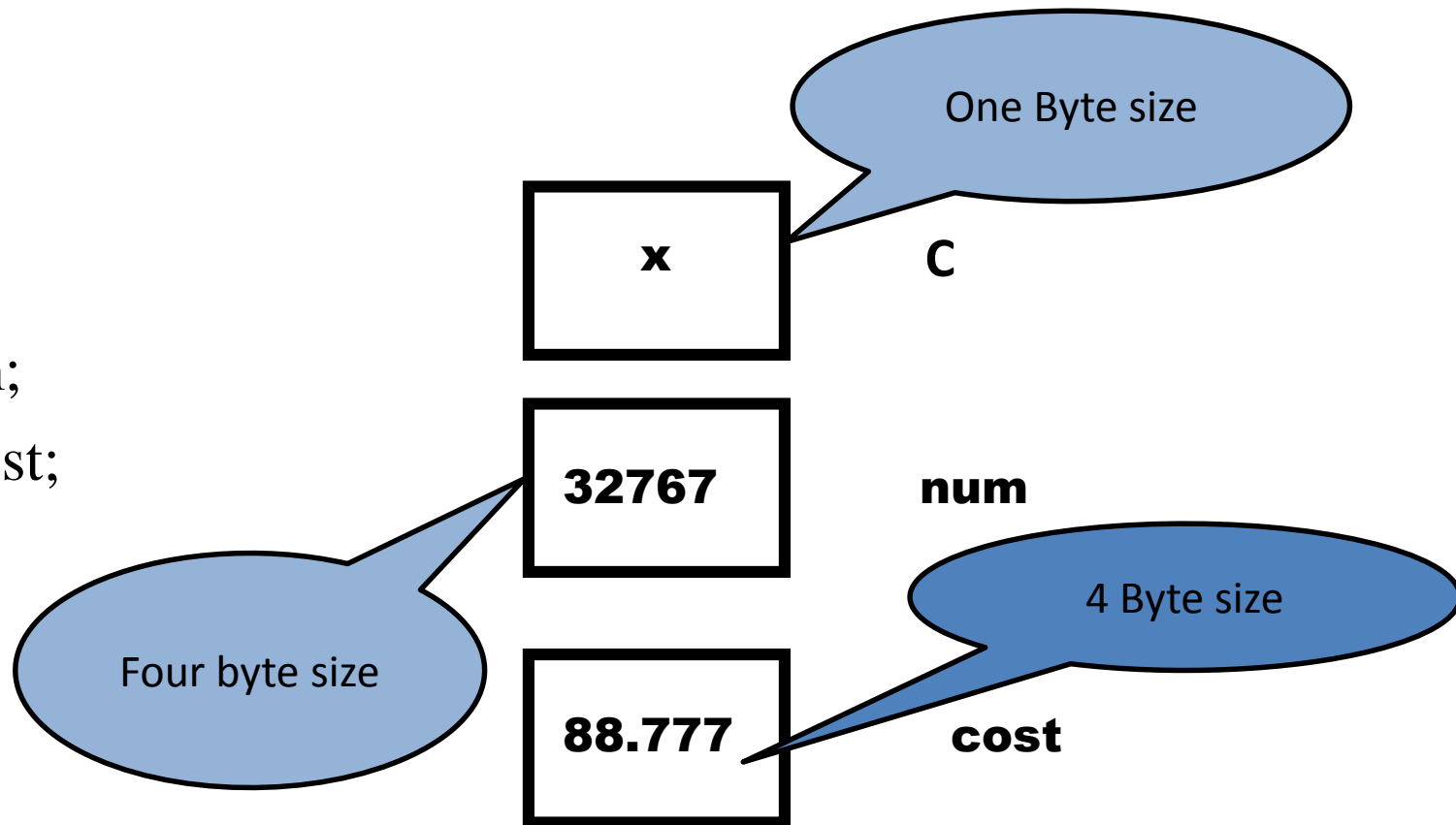
VARIABLES

Variable are named memory locations, which stores values that are going to change during the execution of a program.

Every variable must be declared before assigning a value.

EXAMPLE

```
char c;  
int num;  
float cost;
```



NOTE A variable can not be of type void.

Variable initialization:

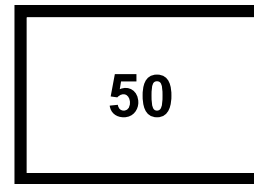
A variable can be assigned a value while declaring them

EXAMPLE

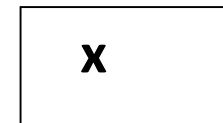
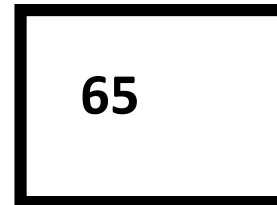
```
int i = 50;
```

```
char c = 'A';
```

```
float x = 20.1;
```



i



Type declaration:

General Syntax of a Type Declaration Statement:

```
data_type data_list;
```



signed short int
unsigned short int
short int
signed int
int
unsigned int
signed long int
unsigned long int
long int
float
double
long double
char
void

Variable names
separated by
commas

Example:

```
int length, width, height;
```



data_type



data list

Example 1:

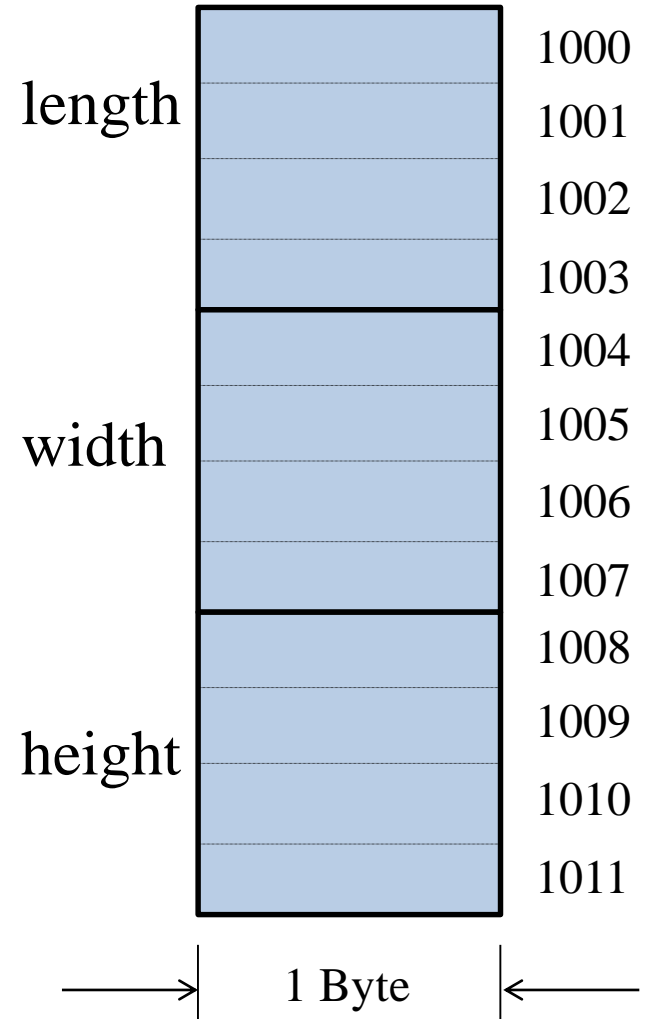
`int length, width, height;`



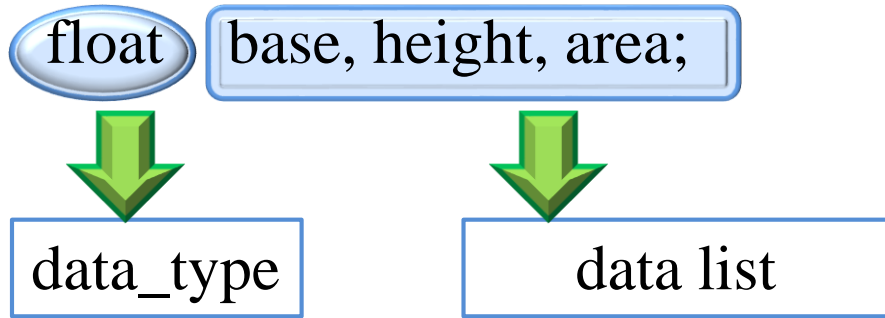
data_type



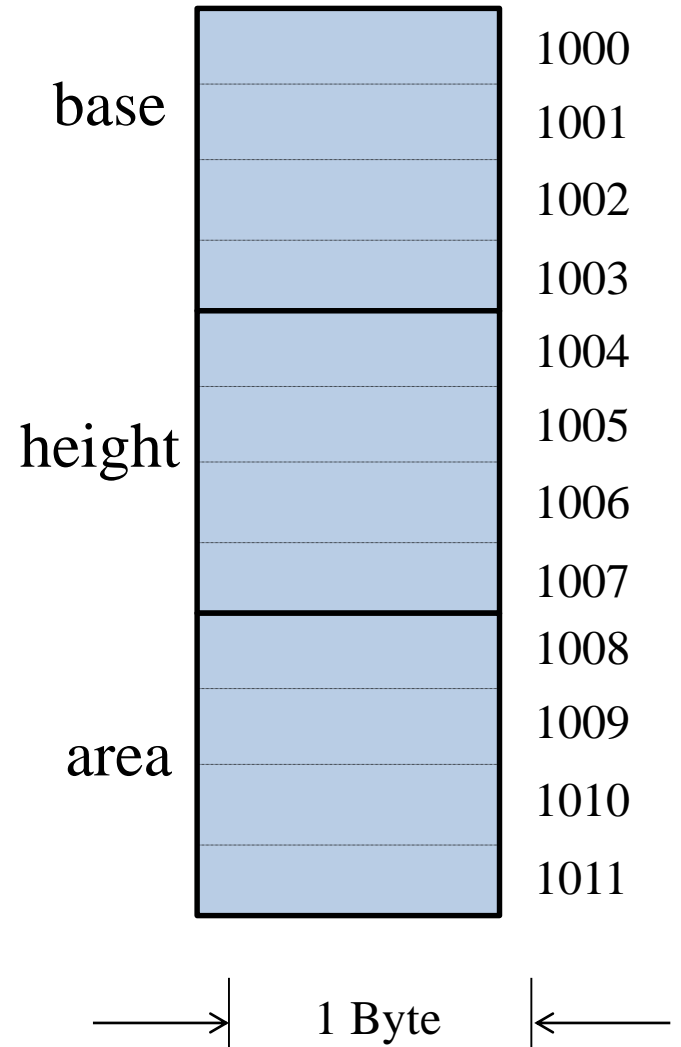
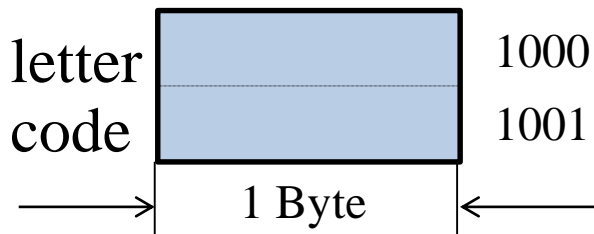
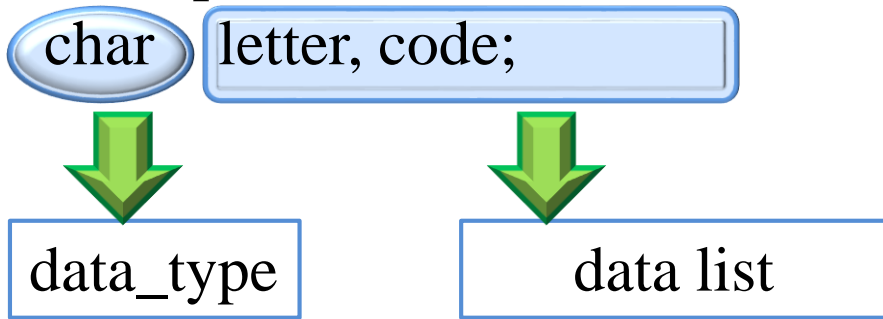
data list



Example 2:



Example 3:



EXAMPLE

short int i,j,k;

int m,n;

long int p,q;

unsigned short int u1,u2;

unsigned int m1,m2;

unsigned long int v1,v2,v3;

char letter1, letter2;

float sum,cost,rate;

double phi,charge;

long double ratio,mass,speed;

Structure of a C Program

Pre Compiler
directives



Preprocessor Directives

Visible
Globally



Global Declarations

Function Beginning



void main(void)
{

Local Declarations
Statements

Function end



}

A SIMPLE PROGRAM

```
#include<stdio.h>
    /*preprocessor*/
    /* main function declaration */
void main (void)
    {
        /* beginning of main */
        printf(“Welcome to UniMAP\n”);
    }
    /* end of main */
```

main function

Every C program has a primary function that must be assigned the name 'main'.

The name 'main' can not be altered by you.

The C compiler needs to know where execution is to begin. 'main' is the first function to be executed.

Comment statement

Comments are notes describing what a particular portion of your program does and how it does it does.

Comments make the program more readable and it is an important part of documentation.

Text that are written between the starting symbol pair `/*` and the ending symbol pair `*/` forms a comment and it will be ignored by the compiler.

There should not be any blank space between `/` and `*`. The `/*` and `*/` form a couple, but they need not be on the same line.

A comment statement can start at any column and may continue to another line also.

A comment line can be written in the very first line of a program or on the very last line of a program.

Normally, most of the C compilers treat comment like a single white space. Therefore, a comment is placed wherever a white space is allowed to be placed.

Comment Statement Examples:

```
/* This is a comment */
```

```
/ *This is not a comment /
```

```
void /*comment*/ main(void)
```

```
printf /* comment */ (“UMS”);
```

```
printf(/*can not place comment like this */);
```

```
#include</*not a valid comment*/stdlib.h>
```

FORMATED INPUT/OUTPUT FUNCTIONS

A C program accepts data or output the result only through a file.

The keyboard is considered as the standard input file. The monitor is considered as the standard output file.

When we enter a data on the keyboard, the C program receives them.

Everything entered into the C program through the keyboard must be in the form of a sequence of characters.

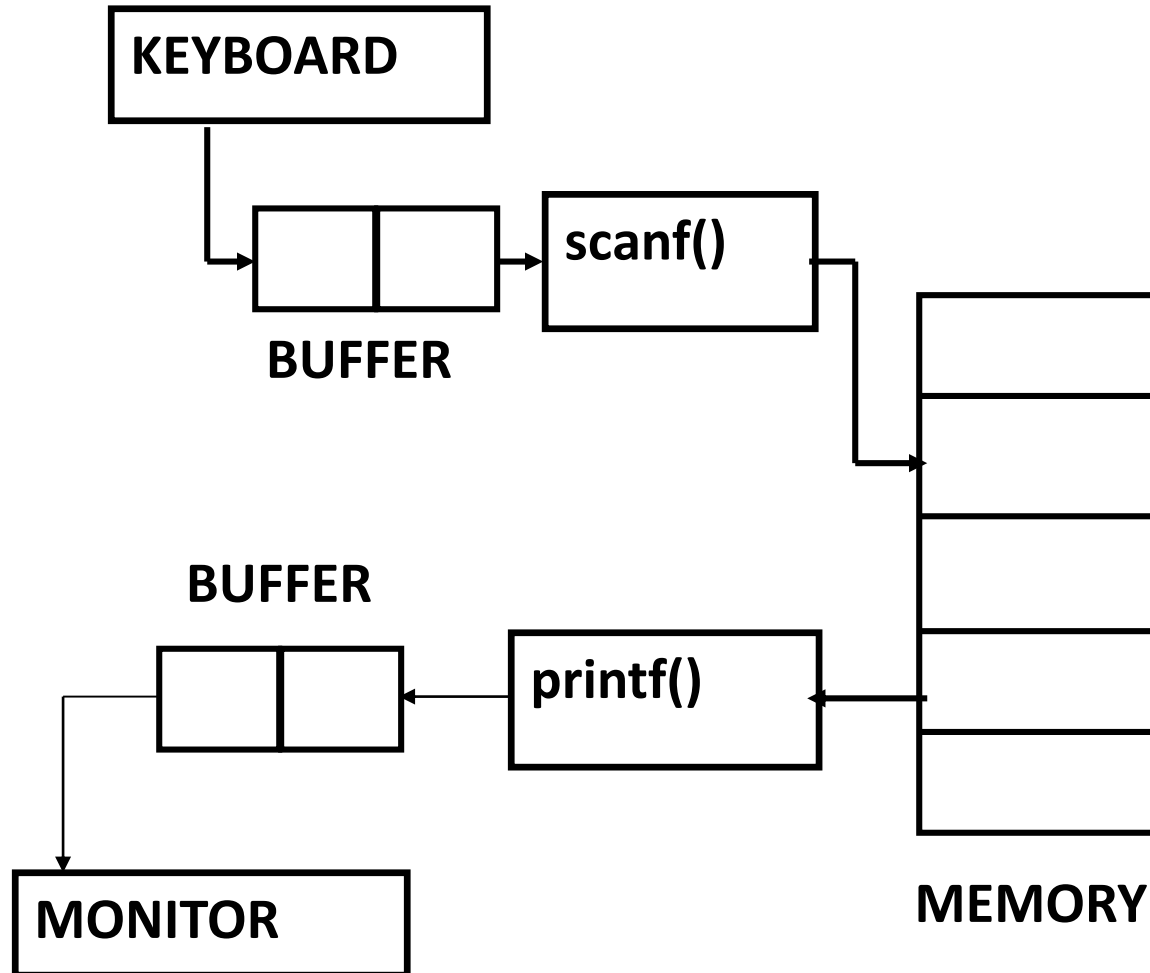
The standard input file is buffered.

ie. The data are stored into a temporary memory location till the return key is pressed.

The C program formatting instructions then interpret the sequence of characters into appropriate form.

scanf()

scanf() function is used to format the input sequence of characters.

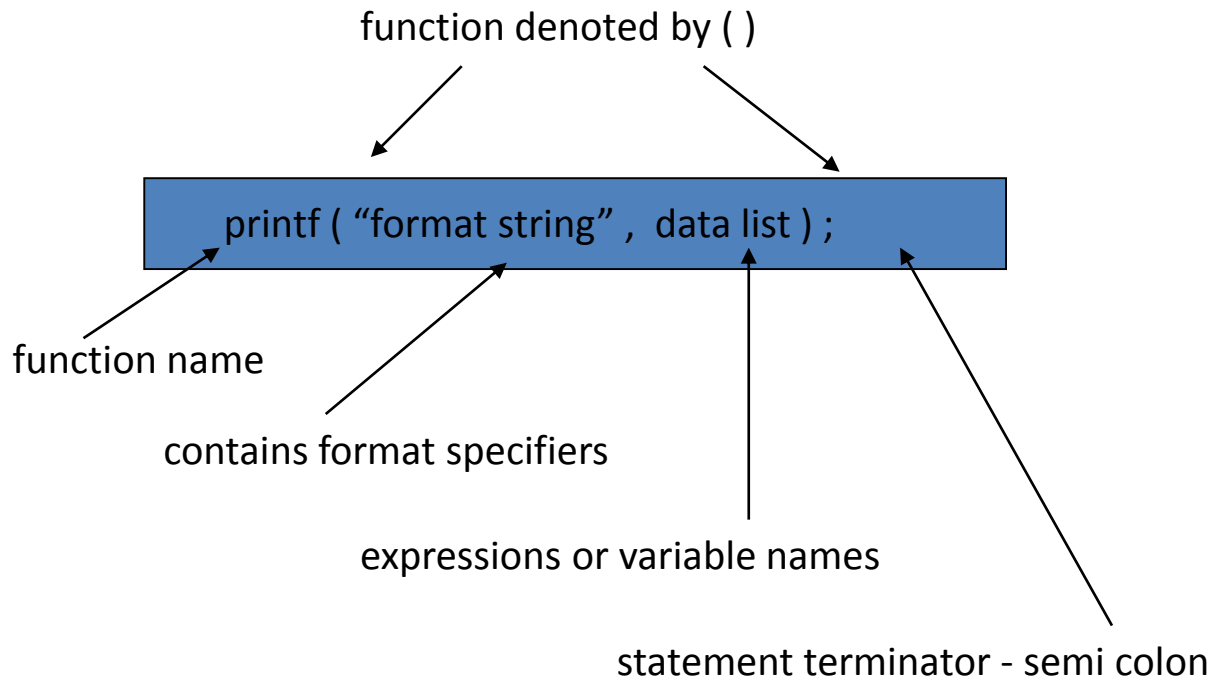


printf()

The printf() function is used to convert the binary and text data stored in the memory into user readable formatted data.

printf() function requires the following parameters:

Instructions for formatting the data (also known as format string)
The actual data to be printed (also known as data list).



General syntax of printf ()

```
printf(format string, data list);
```

The format string is enclosed in a set of double quotation marks.

The format string contains the text data to be printed and instructions for formatting the data.

The instructions for formatting the data are specified by field specifiers.

Each field specification begins with a percent sign (%).

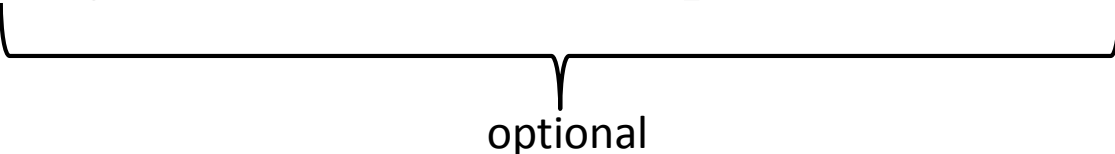
FIELD SPECIFIER

1. a percent sign token ,
2. a conversion code ,
3. optional modifier namely
 - (i) flag
 - (ii) minimum width
 - (iii) precision
 - (iv) size.

The conversion code specifies the data type. There are 30 different conversion codes.

The general form of a field specifier is

%<flag><minimum width><precision><size>conversion_code



optional

CONVERSION CODE

Char – c

float – f

float scientific – e

double - lf

integer – d

long integer-ld

unsigned – u

long unsigned- lu

octal – o

hexadecimal – x or X

string – s

The size is used to modify the type specified by the conversion code.

SIZE REPRESENTATION

h – used with integer to represent short integer.

l – used with integer to represent long integer.

L – used with float to represent long double

The width modifier is used to specify the minimum number of positions in the output.

If the data require more spaces the printf() will override the width modifier.

The width modifier is used to align the output in columns.

If we don't use the width modifier, each output will take just enough room for the data.

Size	Code	Type	Example
None	c	Char	%c
h	d	short int	%hd
None	d	int	%d
l	d	long int	%ld
None	f	float	%f
None	f	double	%f
L	f	Long double	%Lf

Value	%d	%4d
12	12	BB12
123	123	B123
1234	1234	1234
12345	12345	12345

Precision modifier

- 1.The precision modifier is used to specify the number of decimal places.
- 2.The precision modifier has the following format:
 .m
where m represents the number of decimal digits.
- 3.If the precision modifier is not specified
 then printf() prints six decimal positions.

When both width and precision are specified, the width must be large enough to contain the integral value of the number, the decimal point and the number of digits in the decimal position.

EXAMPLE

Field Specifier	Meaning
%2hd	Short integer minimum 2 print positions
%4d	Integer 4 print positions
%8ld	Long Integer 8 positions
%7.2f	Float 7 print positions, nnnn.nn
%10.3L	Long double 10 positions, nnnnnn.nnn

Flag

Flag is used for two print modifications.

If the flag is a minus sign, then the data are formatted as left justified.

If the flag is zero and there is width specification, the number will be printed with leading zeros.

Field specifier	Meaning
% -8d	Integer 8 print positions, left justify
%08d	Integer 8 print positions with leading zeros

Example 1:

```
printf(“%d%f%c”,123,4.56,’a’);  
1234.560000a
```

Example 2:

```
printf(“%d %f %c”,123,4.56,’a’);  
123 4.560000 a
```

Example 3:

```
printf(“%5d%7.2f”,123,4.56);  
bb123bbb4.56
```


Example 4:

```
printf("Transformers");  
Transformers
```

Example 5:

```
printf("My age is %d",21);  
My age is 21
```

Example 6:

```
printf("The cost of pen is %4.2f",3.20);  
The cost of pen is 3.20
```

Example 7:

```
printf("Good Morning Class");  
Good Morning Class
```

Example 8:

```
printf(“Good\nMorning\nClass”);
```

Good

Morning

Class

Example 9:

```
Printf(“Good\tMorning\nClass”);
```

Good Morning

Class

Example 10:

```
Printf(“%06d\t%2hd”,22,22);
```

000022 22

scanf() function

scanf() is used to read the data supplied by the user and to transfer them to the variable names.

The general syntax of scanf function is:

```
scanf(format string, address list);
```

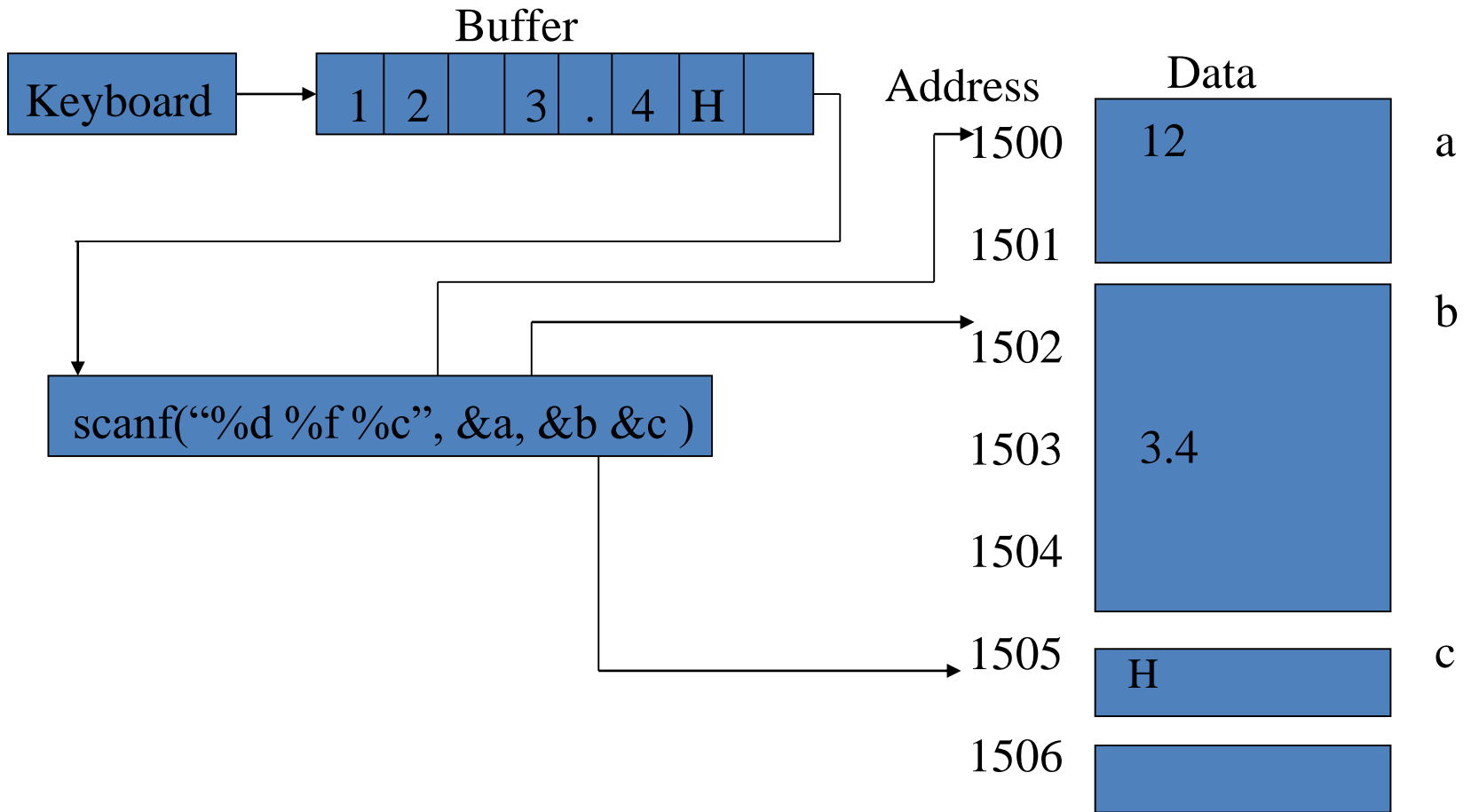
scanf() requires that the variables in the address list be represented by their addresses. To specify an address, you prefix the variable name with the address operator, the ampersand (&).

Example:

To read the following data: 214 156 14z

```
scanf(“%d%d%d%c”,&a,&b,&c,&d);
```

The ‘scanf’ function allows the user to enter the data through the standard input (keyboard), formats the data entered and assign them to variables.



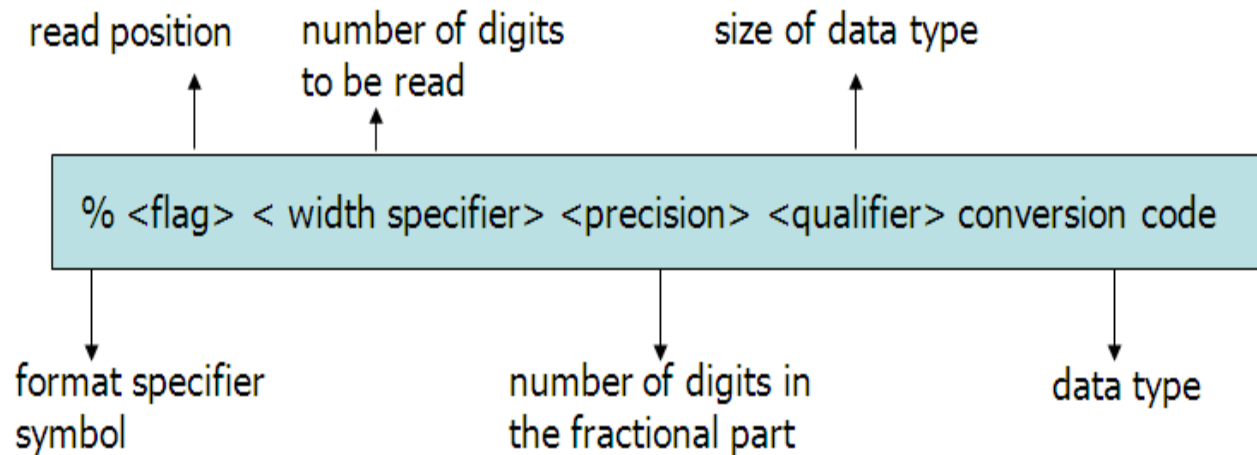
The general form a ‘scanf’ function is

```
scanf( “format string” , argument address list);
```

The format string determines how the ‘scanf’ function reads information into the variables pointed by the argument address list.

The format string contains format specifiers that must match in order with the arguments type. The number of format specifiers must be equal to the number of addresses in the address list. The ‘scanf’ functions returns the number of fields assigned values. If an error occurs before any assignments are made, EOF (end of file) is returned.

A field specifier must begin with a percentage sign. A field specifier must have a conversion code. With a few exceptions, the conversion code used in a 'printf' function is used in the 'scanf' function also. The general form of a format specifier is shown below



The conversion codes are used to interpret the data value keyed in by the users and store them in the variable names. For example, a '%c' field specifier specifies that the input data has to be interpreted as a character.

Data type	Conversion code
Char	c
short int int long int	d
unsigned int unsigned short int unsigned long int	u
short-octal int-octal long-octal	o
short-hexadecimal int-hexadecimal long-hexadecimal	x, X
float double long double	f
float-scientific double-scientific long double-scientific	e, E, g, G
String	s
Pointer	p

When a user enters data, the data values are buffered in the input stream.

The 'scanf' function reads the sequence of characters stored in the buffer and interprets them using the format specifier.

Except for a character field specifier '%c', the scanf function skips all the leading white spaces, tabs or new-lines from the input stream.

If the conversion code is of character type, then the scanf function reads one character. This may be any character including white space. While reading a character data and to skip the leading white space a blank space is always included before the character field specification (example “ %c”).

If the conversion code is of type numerical (‘%d’ , ‘%f’ , ‘%e’), and the format specifier has **no width specifier**, the ‘scanf’ function reads the numeric data until it finds a trailing white space.

If the conversion code is of type numerical (‘%d’ , ‘%f’ , ‘%e’), and the format specifier has a **width specifier**, the ‘scanf’ function reads the numeric data until the maximum number of numerical digits as specified in the maximum width –specifier has been reached or if it finds a white space character.

If the ‘scanf’ finds a white space before the maximum number of numeric digits are processed, it stops.

The user can stop a ‘scanf’ function by signaling that there is no more input to process by keying end of file (EOF). The user can also signal the EOF by using ‘ctrl + z’ in microcomputers or using ‘ctrl + d’ in Unix and Apple machines.

If a 'scanf' encounters any invalid character, when it is trying to convert the input to the stored data type, it stops. For example, if the scanf encounters an alphabetical character when it is trying to read a numerical value, it stops.

When reading a numerical data, the 'scanf' can encounter a plus sign or a minus sign, numerical digits and one decimal point. If the 'scanf' function encounters any other character, it will cause an error.

The second argument in the ‘scanf’ function is an address list.

We have already seen that each and every memory location has an address and using a type declaration statement we have also attached a name to it. The name is referred as a variable name. Address operator ‘&’ is used to get the address associated to a variable name. For example if ‘count’ is a variable name and prefixing a ‘&’ symbol to the variable name, &count is the address.

The following section contains several examples on 'scanf' functions.

1. To read integer values

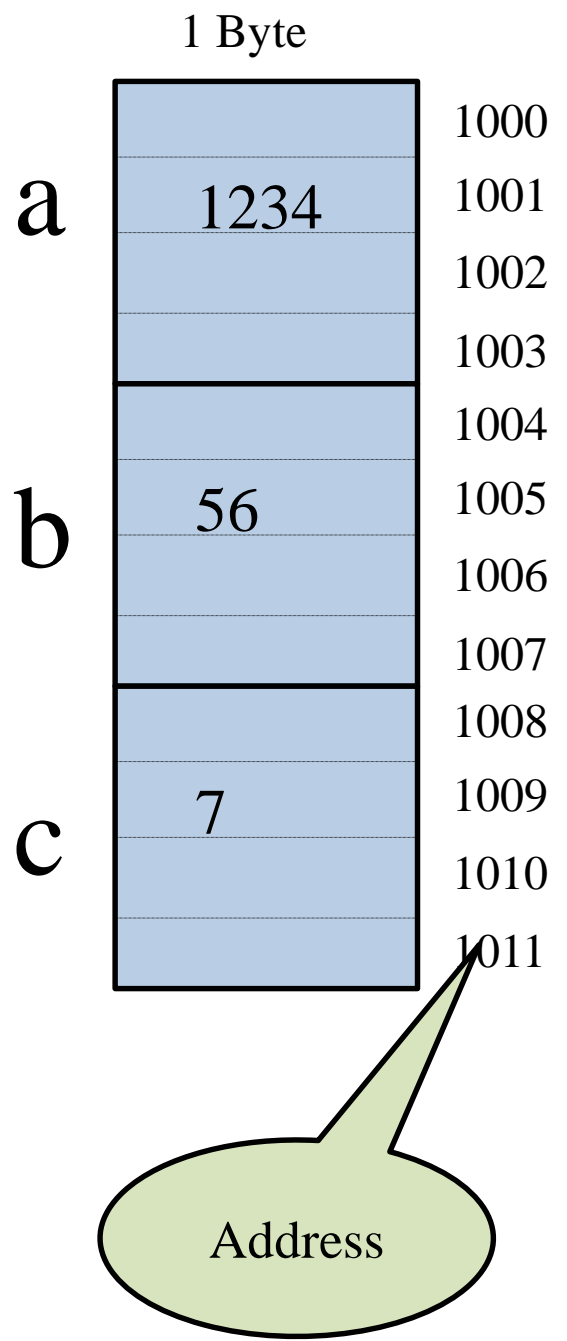
```
int a,b,c;  
scanf(“%d%d%d”,&a,&b&c);
```

When the user enters the following input on a single line
1234**b**56**b**7

then 'a' will be assigned with the value 1234, the variable '**b**' will be assigned with 56 and c will be assigned with the value 7.

The same result will occur if the user enters the three data on three different lines as

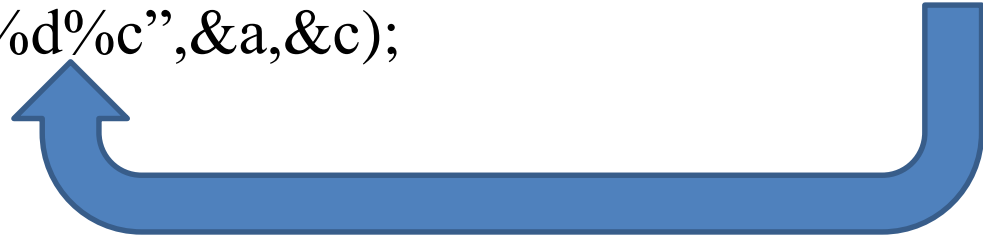
```
1234  
56  
7
```



2. A leading white space is required to read a character. Consider the following example.

```
int a;  
char c;  
scanf("%d%c",&a,&c);
```

NOTE:
NO BLANK SPACE



- a. If the user enters the following data as: 1234bx
Then the variable 'a' will be assigned with 1234 and a white space character is assigned to c.
- b. If the user enters as: 1234x
The variable 'a' is assigned with 1234 and the variable c will be assigned with 'x'.

The 'scanf' function shown above is modified as

```
scanf("%d %c",&a,&c);
```



Note the blank space

c. If the user enters the data as

1234bx

The variable 'a' is assigned with 1234 and the variable c will be assigned with 'x'.

3. To read an integer, float and a character type data

```
int a;  
float f;  
char c;  
scanf(“%d %f %c”,&a,&f,&c);  
printf(“%d %f %c:”,a,f,c);
```

If the user enters the following data

12 34.435 A

The output of the program fragment is

12b34.435000bA

4. To read data using a format specifier that has a maximum width specifier

```
int a;  
float f1,f2;  
scanf(“%03d%05f%05f”,&a,&f1,&f2);  
printf(“%0d b%f b%f\n”,a,f1,f2);
```

If the user enters the following data

12312.34**12.34**

the ‘printf’ functions prints the output as

123**b**12.340000**b**12.345000

5. If the number of field specifiers is more than the number of addresses in the address list, the ‘scanf’ forces the user to input as many data values as the number of field specifiers. After reading the data, the system will indicate the null pointer assignment.

```
int a,b;  
scanf(“%d%d%d”,&a,&b);
```

The ‘scanf’ will force the user to input three values and as it is not possible to assign all the three values, it will give an error message as a “Null pointer assignment”.

Write a program that prompts the user to enter four integer numbers and print them horizontally and vertically.

```
#include<stdio.h>

int main(void)
{
int a, b, c, d;
printf(“Please enter four numbers “);
scanf(“%d %d %d %d”,&a,&b,&c,&d);
printf(“\nYou have entered \n”);
printf(“%d %d %d %d\n”,a,b,c,d);
printf(“\nThe numbers are printed vertically\n”);
printf(“%d\n%d\n%d\n%d\n”,a,b,c,d);
return 0;
}
```

A sample run of the above program
is shown below.

Please enter four numbers 12 345
6789 123

You have entered
12b345b6789b123

The numbers are printed vertically
12
345
6789
123

Write a program in C that prompts the user to enter the length, breadth and height of a room. The program should accept the data keyed in by the user and displays them with appropriate messages.

```
/* Program to Read and Display room dimensions */
```

```
int main(void)
{
float length, breadth, height;
printf("Enter the length of the room ");
scanf("%f", &length);
printf("Enter the breadth of the room ");
scanf("%f",& breadth);
printf("Enter the height of the room ");
scanf("%f", &height);
printf("\nRoom Dimensions \n");
printf("Length = %.2f\n", length);
printf("Breadth = %.2f\n", breadth);
printf("Height = %.2f\n", height);
return 0;
}
```

A sample run of the above program is shown below.

Enter the length of the room 10.23

Enter the breadth of the room 6.0

Enter the height of the room 5.12

Room Dimensions

Length = 10.23

Breadth = 6.00

Height = 5.12

Write a program in C that prompts the user to enter the three sides of a triangle. The program should accept the data keyed in by the user and displays them with appropriate messages. Further, the program must compute and display the area of the triangle.

```
/* Program to Compute Area of a triangle*/
int main(void)
{
float side1, side2, side 3, s, area;
printf("Enter the three sides ");
scanf("%f%f%f", &side1, &side2, &side3);
s = (side1+side2+side3)/2.0;
area = sqrt(s((s-side1)*(s-side2)*(s-side3)));
printf("The three sides are %f\t%f\t%f\n", side1, side2, side3);
printf("The area is %f\n", area);
return 0;
}
```

Thank You