Structured Programming Language

Lecture 7

Outline

- Constants
- Inputs and Outputs
- Formatted IO

Constants

- Constants refer to fixed values that the program may not alter during its execution.
- These fixed values are also called **literals**.
- Constants can be of any of the basic data types like
  - *an integer constant,*
  - *a floating constant,*
  - *a character constant,*
  - *or a string literal*.
- There are enumeration constants as well.

Constants Cont..

Here are some examples of integer literals –

```
212 /* Legal */
 215u /* Legal */
0xFeeL /* Legal */
078 /* Illegal: 8 is not an octal digit */
032UU /* Illegal: cannot repeat a suffix */
```

Following are other examples of various types of integer literals –

```
85 /* decimal */
0213 /* octal */
0x4b /* hexadecimal */
30 /* int */
30u /* unsigned int */
30l /* long */
30ul /* unsigned long */
```

Constants Cont…

- A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part.
- You can represent floating point literals either in decimal form or exponential form.
- Here are some examples of floating-point literals −

```
3.14159 /* Legal */
 314159E-5L /* Legal */
 510E /* Illegal: incomplete exponent */
210f /* Illegal: no decimal or exponent */
.e55 /* Illegal: missing integer or fraction */
```

Constants Cont…

- Character literals are enclosed in single quotes(' '), e.g., 'x' can be stored in a simple variable of **char** type.

Escape sequence	Meaning	Escape sequence	Meaning	Escape sequence	Meaning
\	\ character	\a	Alert or bell	\v	Vertical tab
\'	' character	\b	Backspace	\ooo	Octal number of one to three digits
\"	" character	\f	Form feed	\xhh . . .	Hexadecimal number of one or more digits
\?	? character	\n	Newline	\r	Carriage return
\t	Horizontal tab				

Constants Cont…

- String literals or constants are enclosed in double quotes ("") .
- A string contains characters that are similar to character literals:
  - plain characters,
  - escape sequences,
  - and universal characters.
- Examples:

```
"hello, dear"
"hello, \ dear"
 "hello, "
 "d"
"ear"
```

Defining Constants

- There are two simple ways in C to define constants −
  - Using **#define** preprocessor.
  - Using **const** keyword.
- Given below is the form to use #define preprocessor to define a constant −
  - #define identifier value

Defining Constants : Using #define

- Symbolic names have the same form as variable names
- No blank spaces between the # sign and the word define
- # must be the first character in the line
- Blank space required between # and identifier
- #define must not end with a semicolon
- Symbolic name should not be assigned with assignment operator
- Examples…

Program: Constant using #define

```c
#include <stdio.h>

#define LENGTH 10
#define WIDTH 5
#define NEWLINE '\n'

int main()
{
    int area;

    area = LENGTH * WIDTH;
    printf("value of area : %d", area);
    printf("%c", NEWLINE);

    return 0;
}
```

Defining Constants

- We can use const prefix to declare constants with a specific type as follows −
- const type variable = value;

```
#include <stdio.h>

int main()
{
        const int LENGTH = 10;
        const int WIDTH = 5;
        const char NEWLINE = '\n';
        int area;

        area = LENGTH * WIDTH;
        printf("value of area : %d", area);
        printf("%c", NEWLINE);

        return 0;
}
```

Managing input
and
Output operation

- Reading, processing and writing of data are three essential functions of a computer program.

- Most programs take some data as input and display the processed data or result as output.

- So, we need some methods that can read input and write output on a suitable input/output medium.

- Because unlike other high-level languages, C does not have any built-in input/output statements as part of its syntax.

Input/output functions

- For outputting the results we have used the function printf which sends the results out to a terminal.

- And for inputting, so far we have seen one methods for providing data to the program variables.

  1. That is to assign values to variables through the assignment statements such as x = 5; and so on.

The second method is

2. Is to use input function scanf which can read data from a terminal.

Reading a character

- There are also some functions other than
  - printf
  - and scanf

which can be used as input/output functions.

- The simplest of all input/output operations is
  - reading a character from the standard input
  - and writing it to the standard output unit.

The getchar() function

- ❑ Reading a single character can also be done by using the function *getchar()*.

The **getchar()** function is used to read a single character from the standard input unit.

It takes the following form:

```
variable_name = getchar();
```

- ❑ variable_name is a valid C name that has been declared as char type.

- ❑ When this statement is encountered, the computer waits until a key is pressed and then assigns this character as a value to the *getchar()* function.

Example

Since *getchar()* is used on the right-hand side of an assignment statement, the character value of *getchar()* is assigned to the variable name on the left side.

For example:

```
char alphabet;

 alphabet = getchar();
```

Will assign the character ‘A’ to the variable **alphabet** when we press the key A on the keyboard.

- The **getchar()** function may be called successively to read the characters contained in a line of text.

- The **getchar()** function returns any character keyed in. this include RETURN and TAB also. This means that it will not return, until u press ENTER.

Writing a character

- Like **getchar**, there is an analogous function **putchar** for writing characters one at a time to the terminal.

- It takes the following form:

putchar(variable_name**)**;

- where *variable_name* is a type char variable containing a character.
- This statement displays the character contained in the *variable_name* at the terminal.

Example:

```
char answer = ‘Y’;
putchar(answer);
```

- ❑ It will display the character Y on the screen.
- ❑ The statement **putchar(‘\n’)**; would cause the cursor on the screen to move to the beginning of the next line.

Formatted Input

❑ Formatted input refers to an input data that has been arranged in a particular format.

❑ For inputting the data, we use the scanf function. The general form of this **scanf** is

scanf("controlString",arg1,arg2,...argn);

Control string specifies the field format in which the data is to be entered.

❑ Arguments arg1, arg2,…..specify the address of locations where the data will be stored.

❑ The field (or format) specifications, consisting of the

- conversion character %,
- and an optional number specifying the field width.
- a data type character(or type specifier)

Inputting Integer Numbers

- The field specification for reading an integer number is:

  %wd

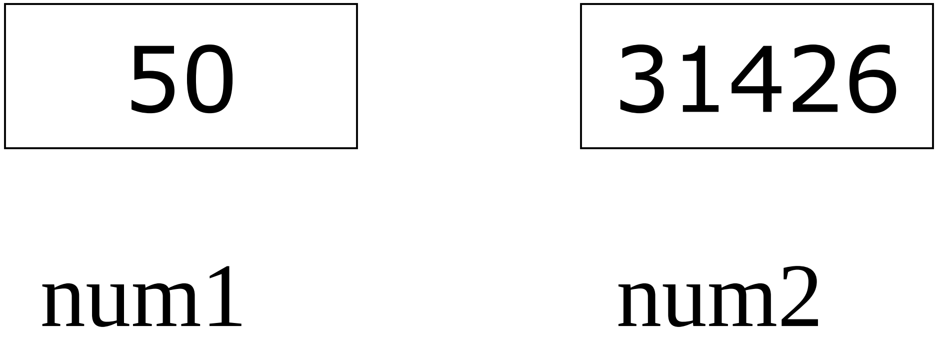
- **w** is an integer number that specifies the field width of the number to be read and **d**, known as data type character(here int).

Example:-

```
scanf(“%2d%5d”,&num1,&num2);
```

Input: 50 31426

- ❑ Then,



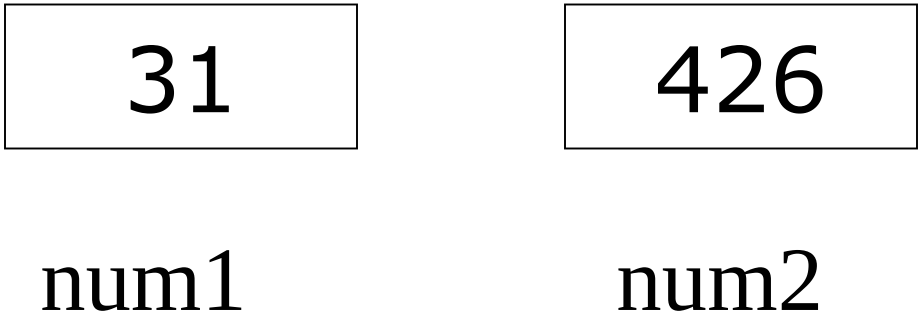
50

31426

num1

num2

But if input : 31426 50



31

426

num1

num2

And the value 50 that is unread will be assigned to the first variable in the next scanf call.

- ❑ On the other way if we have written :
  scanf(“%d%d”,&num1,&num2);
  Will read the data 31426 50
  Correctly in **num1**, **num2**.
- ❑ Input data items must be separated by *spaces*, *tabs* or *new lines*.
- ❑ When the **scanf** reads a particular value, reading of the value will terminate as soon as the number of characters specified by the field width is reached(if specified) or until a character that is not valid for the value being read is encountered.

- An input field may be skipped by specifying ***** in the place of field width.

For example,

scanf (%d%*d%d”,&a,&b);

Input: 123 456 789

123	456	789
a	skipped	b

The data character d may be preceded by ‘ll’ to read long long integers. (%lld)

Inputting Real Numbers

- The field width of real numbers is not to be specified and therefore **scanf** uses simple specification **%f** for both the notations, decimal point and exponential notation.
- For double type , **%lf** is used instead of **%f**.
- A using **%*f** specification. number may be skipped

Inputting Character Strings

❑ Single character can be read from the terminal using **getchar** function. The same can be achieved using the **scanf** function also.

❑**In addition**, a **scanf** can input strings containing more than one character.

- The specifications for reading character strings are:

  %s or **%c**

- **%c** used to read a single character while **%s** terminates the reading at the encounter of blank space.

Points to remember for scanf

1. All function arguments must be a pointers to variables.
2. Format specifications contained in the control string should match the argument order.
3. Input data items must be separated by spaces and must match the variables receiving the input in the same order.

4. The reading will be terminated, when the scan encounters an ‘invalid mismatch’ of data or a character that is not valid for the value being read.

5. When searching for a value, **scanf** ignores line boundaries and simply looks for the next appropriate character.

6. Any unread data items in a line will be considered as a part of the data input line to the next **scanf** call.

7. When the field width specifier **w** is used, it should be large enough to contain input data size.

Formatted Output

- ❑ It is highly desirable that the outputs are produced in such a way that they are understandable and are in an easy-to-use form.

- ❑ It is therefore necessary for the programmer to give careful consideration to the appearance and clarity of the output produced by program.

- **printf** statement provides certain features that can be effectively control the alignment and spacing of print-outs on the terminals.

printf format

printf("control string",arg1,arg2...argn);

Control string consists of three types of items:

1. Characters that will be printed on the screen as they appear.
2. Format specifications that will define the output format for display of each item.
3. Escape sequence characters such as \n, \t, and \b.

Output of Integer number

- The format specification for printing an integer is %wd. Where w specifies the minimum field width for the output.

Format Output

- printf(“%d”,9876)

9	8	7	6

- printf(“%6d”,9876)

		9	8	7	6

- printf(“%2d”,9876)

9	8	7	6

- printf(“%-6d”,9876)

9	8	7	6		

- printf(“%06d”,9876)

0	0	9	8	7	6

Output of Real number

- ❑ The output of real number may be displayed in decimal notation using the format %w.pf

If y = 98.7654

Format Output

printf(“%7.4f”,y)

9	8	.	7	6	5	4

printf(“%7.2f”,y)

		9	8	.	7	7

printf(“%-7.2f”,y)

9	8	.	7	7		

printf(“%f”,y)

9	8	.	7	6	5	4

Printing of a single character

- A single character can be displayed in a desired position using the format **%wc**.
- The character will be displayed right-justified in the field of **w** columns.
- We can make the display left-justified by placing a minus sign before w.
- The default value for w is 1.

Printing of strings

- The format specification for outputting string is of the form **%w.ps**.
- To print string “HELLO DHAKA 110001”, containing 18 characters.

%20s

1									10										20

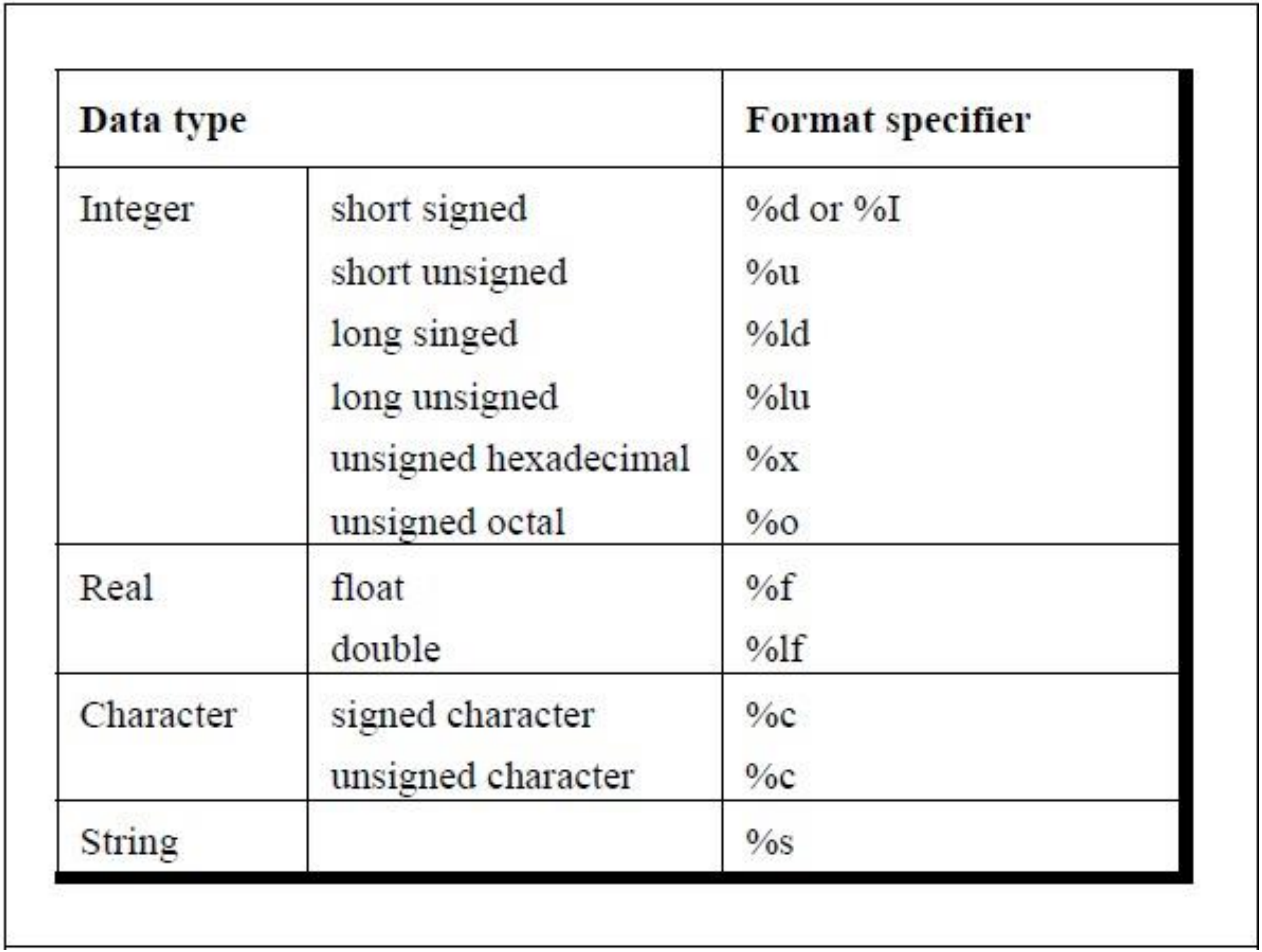
To print string “HELLO DHAKA 110001”, containing 18 characters.

%20s

		H	E	L	L	O		D	H	A	K	A		1	1	0	0	0	1

%20.10s

										H	E	L	L	O		D	H	A	K



Data type		Format specifier
Integer	short signed	%d or %I
	short unsigned	%u
	long singed	%ld
	long unsigned	%lu
	unsigned hexadecimal	%x
	unsigned octal	%o
Real	float	%f
	double	%lf
Character	signed character	%c
	unsigned character	%c
String		%s