Year : I

Semester : II

Subject Title : C PROGRAMMING

Sub Code : 18BCS23C

UNIT II

CHAPTER IV

4.1 READING A CHARACTER

- The simplest of all input/output operations is reading a character from the '***standard input***'(keyboard) unit and writing it to the '***standard output***' (screen) unit.
- Reading a single character-***getchar()*** and ***scanf().***

Syntax

variable-name=getchar();

- Variable-name is a valid c name that has declared as char type.

Example

```
char name;
name=getchar();
```

- The getchar() function may be called successively to read the characters contained in a line of text.
- The following example program segment reads a charcters from the keyboard one after the another until the 'Return' key is pressed.

```
------
------
char c;
c = '';
while(c!='\n')
{
    c=getchar();
}
------
------
```

4.2 WRITING CHARACTER

- Writing a single character-***putchar()*** and ***printf().***

Syntax

putchar(varaible-name);

- Where variable-name is a char type variable.
- putchar() is used to display the character.

Example

1. ans='y';
   putchar(ans);

Display the character y.

2. putchar('\n');

The cursor move to the beginning of the next line.

Character Test Functions

Function	Test
isalnum(c)	Is c an alphanumeric char?
isalpha(c)	Is c an alphabetic char?
isdigit(c)	Is c a digit?
islower(c)	Is c a lower case letter?
isprint(c)	Is c a printable char?
ispunct(c)	Is c a punctuation mark?
isspace(c)	Is c a white space char?
isupper(c)	Is c an upper case letter?

4.3 FORMATTED INPUT

scanf() funtion

Syntax

scanf("control string", arg1, arg2, … , argn);

- The control string specifies the field format in which the data is to be entered.
- arg1,arg2,…argn specify the address of the locations where the data is stored.
- Control string (format string) contains field specifications, which direct the interpretation of input data.
- Control string may includes

1. Field specification-%, a data type character and an optional field width(number).
2. Blanks, tabs, or newlines (ignored).

Integer data type

%wd

- The % symbol indicates that a conversion specification follows, w is an integer number that specifies the field width of the number to be read and d, known as data type character, indicates that the number to be read is in integer mode.

Example

scanf("%2d,%5d",&num1,&num2);

Data line : 15 12345

scanf("%d,%d",&num1,&num2);

Data line: 12345 154

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

Example

scanf(%d %*d %d",&a,&b);

- The data type character d may be preceded by 'l' to read long integers and 'h' to read short integers.
- It is legal to use a non-whitespace character between field specifications.
- The scanf() expects a matching character in the given location.

Example

scanf("%d %d", &a, &b);

Accepts input like 125 456 to assign 125 to a and 456 to b.

Inputting real numbers

The field width of real numbers is not to be specified.

scanf() reads real numbers using %f for both notations(decimal point notation and exponential notation).

Example

Scanf("%f%f%f",&x,&y,&z);

- If the number to be read double type then the specification is %lf.
- A number may be skipped using %*f specification.

Inputing Character Strings

- A single character may be read using the getchar() function.
- Scanf()function can input strings containing more than one characrter.
- The specification is %ws or %wc.
- The curresponding argument should be a pointer to a character array.
- %c may be used to read a single character when the argument is a pointer to a char variable.
- %s terminates reading at the encounter of a blank space.
- Scanf() funtion support %[characters] and %[^caharacter] specification for getting only the specified input string not except the specified input string respectively.

- Reading blank space
  - The specification %[ ] is for reading blank spaces.
  - The blank spaces may be included within the brackets, thus enabling the scanf() function to read strings with spaces.

Reading mixed mode data types

scanf(%d%c%f%s",&a,&b,&c,&d);

- will read input data **15 x 3.14 hello**
- Commonly used scanf() format codes

code	Meaning
%c	Read a single character
%d	Read a decimal no
%e	Read a floating point value
%f	Read a floating point value
%g	Read a floating point value
%h	Read a short integer
%i	Read a decimal, hexadecimal or octal value
%o	Read an octal
%s	Rea a string
%u	Read an unsigned decimal integer
%x	Read a hexadecimal integer
%[..]	Read a string or word(s)

Points to remember

- All functions arguments except the control string, must be pointers to variable.
- Format specification contained in the control string should match the arguments in order.
- Input data items must be separated by spaces and must match the variables receiving the input.
- The reading will be terminated, when scanf() function encounters a 'mismatch' of data or a character that is not valid for the value being read.
- When searching for a value, scanf() ignores line boundaries and simply looks for the next appropriate character.
- Any unread data items in a line will be considered as part of the data input line to the next scanf() function call.
- When the field width specifier w is used , it should be large enough to contain the input data size.

Rules for scanf() function

1. Each variable to be read must have a filed specification.
2. For each field specification, there must be a variable address of proper type.
3. Any non-whitespace character used in the format string must have a matching character in the user input.
4. Never end the format string white space. It is a fatal error.
5. the scanf()function reads until:
   - a white space character is found in a numeric specification, or
   - The maximum no. of characters have been read or
   - An error is detected, or
   - The end of file is reached.

4.4 FORMATTED OUTPUT

Printf() function

Syntax

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

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

- The control string indicates how may arguments follow and what their types are.
- The arguments arg1,arg2, …,argn are the variables whose values are formatted and printed according to the specification of the control string.
- The arguments should match in number, order and type with format specifications.
- A simple format specification form
- %w.p type specifier.
- Where w and p is an integer number that specifies the total number of columns for the output value and the number of digits or the number of characters to be printed respectively.

Output

1. ***Integer numbers - %wd*** – w specifies the maximum width for the output.
2. ***Real numbers***

   -%w.pf- w specifies the maximum number of positions that are to be displayed and the integer p indicates the number of digits to be displayed after the decimal point.

   %w.p e – display a real number in exponential notation

   [-]m.nnnne[±]xx – where the length of the string of n's is specified by the precision is 6. the default precision is 6. the field width w should satisfy the condition. Ie W≥p+7. the value will be rounded off and printed right justified in the field of w columns.

   padding the leading blanks with zeros and printing with left-justification.
3. ***Single character - %wc-*** the character will be displayed right-justified in the field of w columns.
4. ***String- %w.ps*** –where w specifies the field width for display and p instructs that only the first p characters of the string are to be displayed. The display is right justified.

EXAMPLES

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													
printf("%10.2e",y);			9	.	8	8	e	+	0	1										
printf("%11.4e",f)		9	.	8	7	6	5	e	+	0	1									
printf("%-10.2e",y)	9	.	8	8	e	+	0	1												
printf("%e",y);	9	.	8	7	6	5	4	0	e	+	0	1								
printf(%s",a);	N	E	W		D	E	L	H	I		1	1	0	0	0	1				
printf("%20s",a)					N	E	W		D	E	L	H	I		1	1	0	0	0	1
printf("%.5s",a)	N	E	W		D															
printf("%-20.10s",a)	N	E	W		D	E	L	H	I											
printf("%5s",a)	N	E	W		D	E	L	H	I		1	1	0	0	0	1				

CHAPTER V

DECISION MAKING AND BRANCHING

5.1 INTRODUCTION

Decision making statements (control statements)in c are

1. if statement
2. switch statement
3. conditional operator statement
4. goto statement

if Statement

- It is basically, a two way branching statement and is used in conjunction with an expression.

If statement types

1. simple if statement
2. if—else statement
3. nested if statement
4. else if ladder.

5.1.1. Simple if statement

Syntax

if (test expression)

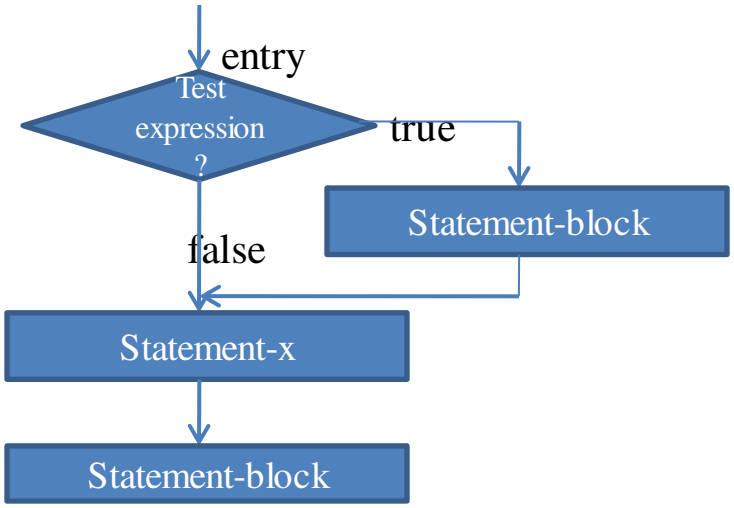
{

Statement-block;

}

Statement –x;

- Statement-block may be a single statement or a group of statements.
- If the test expression is true, the statement-block will be executed. Otherwise the statement-block will be skipped and the execution will jump to the statement-x.
- If the condition is true, both statement block and statement-x are executed in sequence.



Example

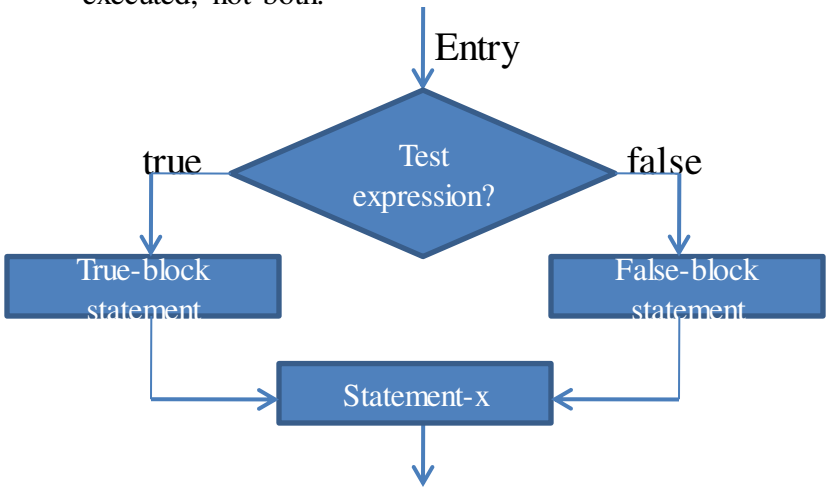
```
main()
{
int a,b,c;
printf("enter the value of a and b\n");
scanf("%d%d",&a,&b);
if (a-b!=0)
{
c=a-b;
printf("the difference is%d",c);
}
}
```

if..else Statement

Syntax

```
if (test expression)
{
    True-block statement(s)
}
else
{
    False-block statement(s)
}
Statement-x;
```

- If the test expression is true then the true-block statement (s), immediately following if statement are executed.otherwise, false-block statement(s) are executed.
- In either case ,either true-block or false-block will be executed, not both.



Example

```
main()
{
int a;
printf("Enter the value of a \n");
scanf("%d%d",&a,&b);
if (a>=0)
{
    printf("a is positive");
}
else
{
    printf("a is negative");
}
}
```

Output

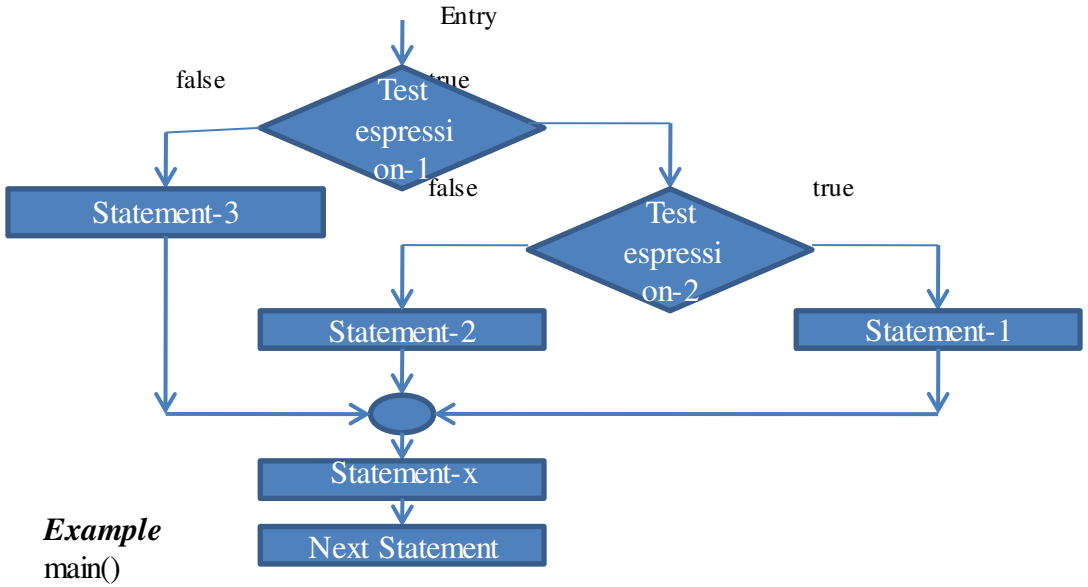
```
Enter the value of a 5
a is positive
Enter the value of a -12
a is negative
```

Nested if..else statement

Syntax

```
if (test expression-1)
{
    if (test expression-2)
    {
        statement-1;
    }
    else
    {
        statement-2;
    }
}
else
{
    statement-3;
}
statement-x;
```

- If test expression-1 is false then the statement-3 will be executed; otherwise, it continues to perform the second test. If the test expression-2 is true then the statement-1 will be executed. otherwise, statement-2 will be executed and then the control is transferred to the statement-x..



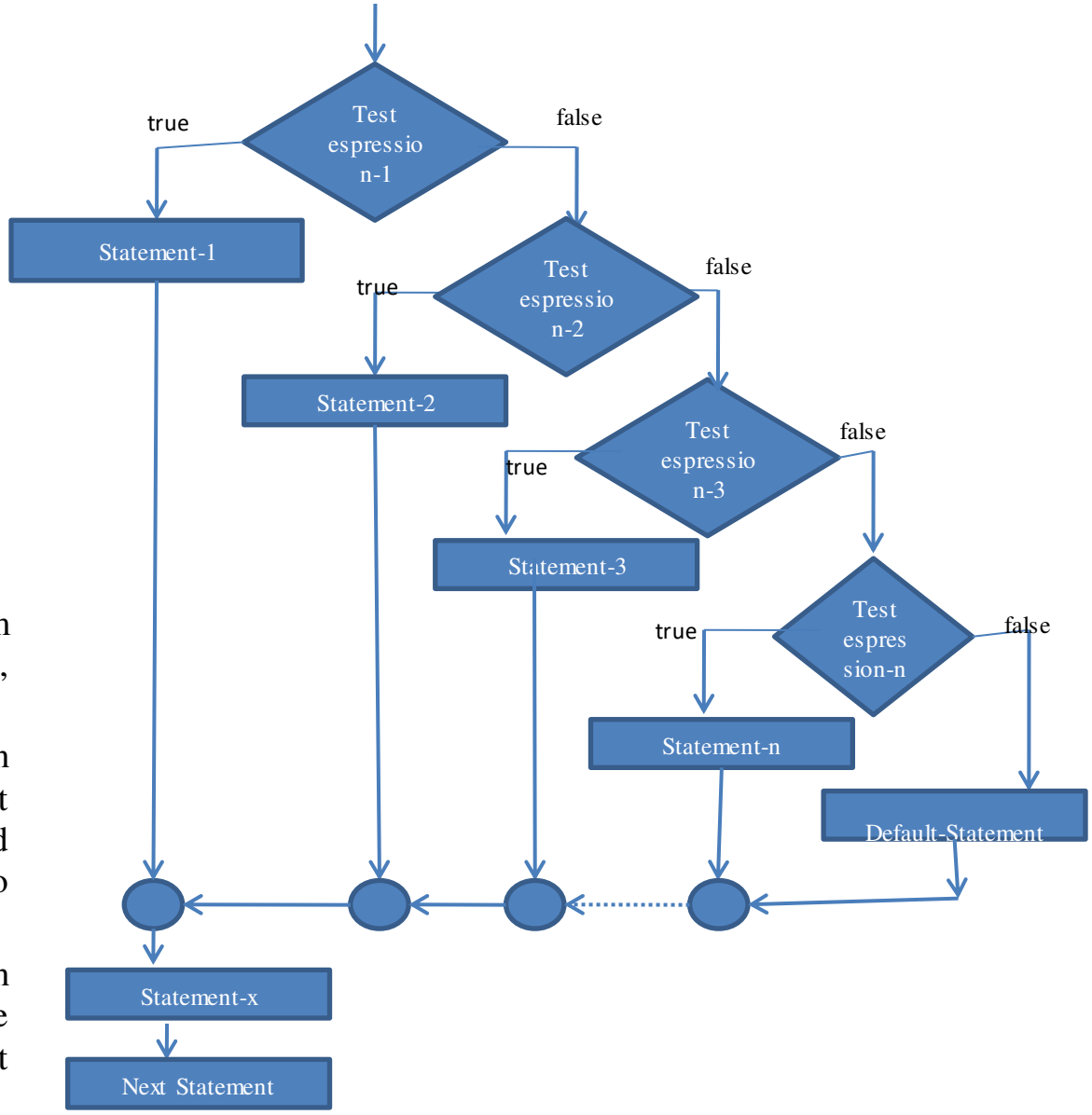
Example

```
main()
{
int a,b,c;
printf("enter the value of a,b and c\n");
scanf("%d%d%d",&a,&b,&c);
if (a>b)
{
    if (a>c)
        printf("largest value is %d",a);
    else
        prinf("largest value is %d",c)
}
else
{
    if (b>c)
        printf("largest value is%d".b);
    else
        printf("largest value is %d",c);
}
}
```

else if ladder

```
if (test expression-1)
  statement-1;
else if (test expression-2)
  statement-1;
else if (test expression-3)
  statement-1;
else if (test expression-n)
  statement-1;
else
  default-statement;
statement-x;
```

- The condition is evaluated from the top(of the ladder), downwards.
- As soon as a true test expreesion is found, the statement associated with it is executed and the control is transferred to the statement-x..
- When all the n test expreesion become false then the final else containing the default–statement will be executed.



Example

- Program to read the customer number and power consumed and prints the amount o be paid by the customer.

Consumption units	rate of charges
0-200	Rs. 0.50 per unit
201-400	Rs. 100+0.65 per unit excess of 200
401-600	Rs. 230+0.80 per unit excess of 400
601 and above	Rs 390+Rs.1 per unit excess of 600

Program

```
main()
{
int units,num;
float charge;
printf("enter the customer no and units consumed\n");
scanf("%d%d",&units,&num);
if (units<=200)
    charge=units*0.50;
else if (units<=400)
    charge=100.00+0.65*(units-200);
else if 9units<=600)
   charge=230+0.80*(units-400);
else
     charge=390+(units-600);
printf("\n Customer No:%d",num);
printf("\nUnits Consumed :%d",units);
printf("\n Charge : Rs. %.2f",charge);
}
```

Output

```
Enter customer no  units consumed
1
500
Customer No:1
Units Consumed :500
Charge: Rs. 310.00
```

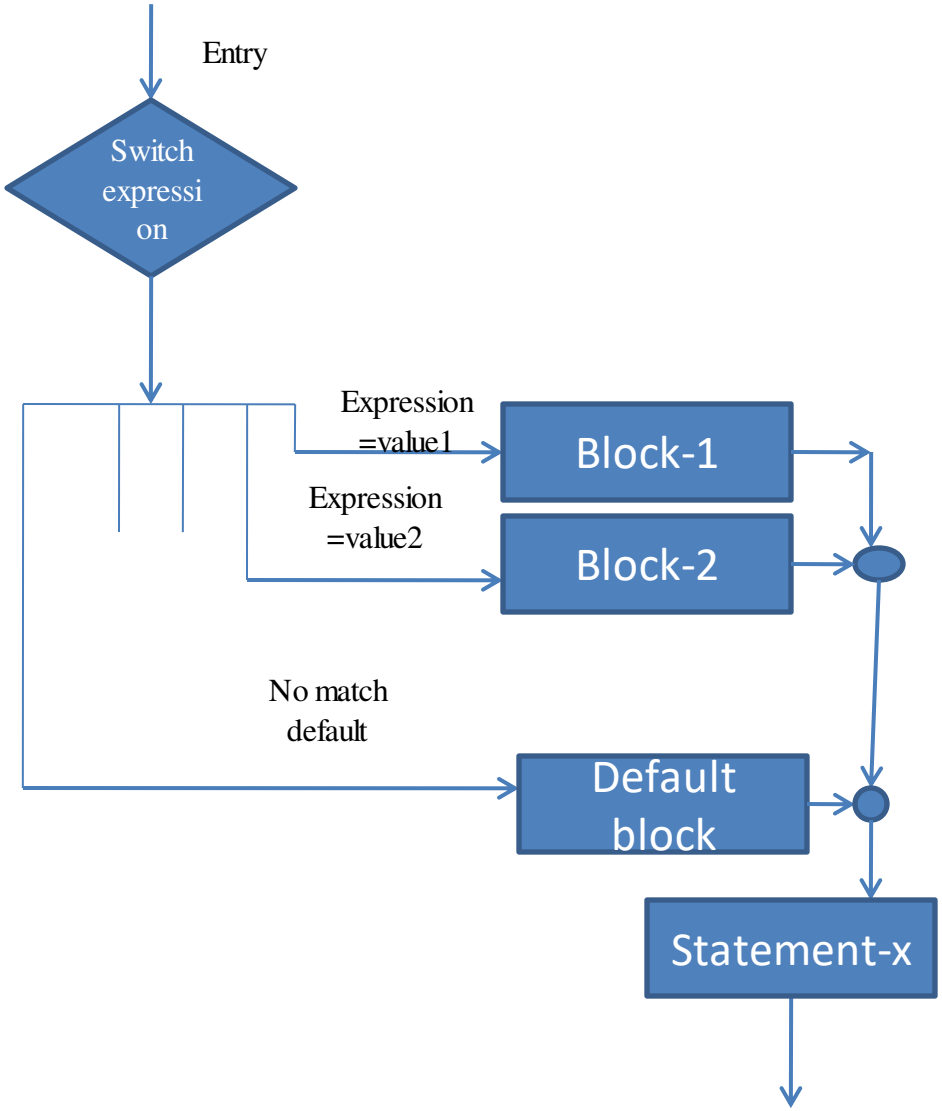
5.1.2 SWITCH STATEMENT

- The ***switch*** statement tests the value of a given variable against a list of case values and when a match is found, a block of statements associated with that case is executed.

Syntax

```
switch (expression)
{
case value1:
        block1;
        break;
case value2:
        block2;
        break;
case value3:
        block3;
        break;
---------
---------
default :
        default block;
        break;
}
Statement-x;
```

- The expression is an integer expression or characters.
- value1,value2,……are constants or constant expressions and are known as case labels.
- case labels end with colon(:).
- Each of these values should be unique within a switch statement
- block1,block2…. are statement lists and may contain zero or more statements.
- There is no need to put braces around these blocks.
- When the switch is executed, the value of the expression is successfully compared with the values value1,value2….
- If a case is found whose value matches with the value of the expression then the block of statements that follows the case are executed.
- The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring the control to the statement-x following the switch.
- The default is an optional case.
- When present, it will be executed if the value of the expression does not match with any of these case values.


Entry
Switch expressi on
Expression =value1
Block-1
Expression =value2
Block-2
No match default
Default block
Statement-x

Example

```
main()
{
int rno,mark1,mark2,mark3,mark4,mark5,total,index;
float ave;
printf("enter rno and fivel subjects marks");
scanf("%d%d%d%d%d%d",&rno,&mark1,&mark2,&mark3,&mark4,&mark5);
total = mark1+mark2+mark3+mark4+mark5;
ave= total/5.0;
index=(int)ave/10;
printf("rno:%d",rno);
printf("subject marks : %d %d %d %d
%d",mark1,mark2,mark3,mark4,mark5);
printf("total: %d",total);
printf("percentage:%.2f",ave);
switch (index)
{
case 10:
case  9:
case 8 :
        printf("garde: distinction");
        break;
case7:
case 6:
        printf("first  class);
        break;
case 5:
        printf("second   class);
        break;
case 4:
        printf("third   class);
        break;
default :
        printf("fail");
        break;
}
}
```

RULES FOR SWITCH STATEMENT

1. The switch expression must be an integral type.
2. Case labels must be constants or constants expressions.
3. Case labels must be unique. no two labels can have same value.
4. Case labels must end width colon.
5. The break statement transfers the control out of the switch.
6. The break statement is optional. Ie, two or more case labels may belong to the same statements.
7. The default label is optional. If present then it will executed when the expression does not find the matching case label.
8. There can be at most one default label.
9. The default may be placed anywhere but usually placed at the end.
10. It is permitted to nest switch statement.

5.2 CONDITIONAL OPERATOR

Conditional operator(?:)

-Two way decision

Syntax

conditional
expression?expression1:expression2;

- The conditional expression is evaluated first.
- If the result is non-zero then expression1 is evaluated and is returned as the value of the conditional expression.
- Otherwise, expression2 is evaluated and its value is returned.

Example 1:

```
if (x<0)
    flag =0;
else
    flag =1;
```

Can be written as

flag=(x<0)?0:1;

Example 2:

```
if (x<=40)
    if (x<40)
            salary = 4 * x+100;
     else
            salary= 300;
else
     salary=4.5*5+150;
```

Can be written as

Salary=(x!=40)?((x<40)?(4*x+100):(4.5*x+150)):300;

5.3 GOTO STATEMENT

- The goto statement is used to unconditionally transfer the control from one point to another in the program.
- The goto requires the label in order to identify the place where the branch is to be made.
- A label is any valid variable name, and must be followed by a colon.
- The label is placed immediately before the statement where the control is to be transferred.

Syntax

goto label;

----

label:

statement;

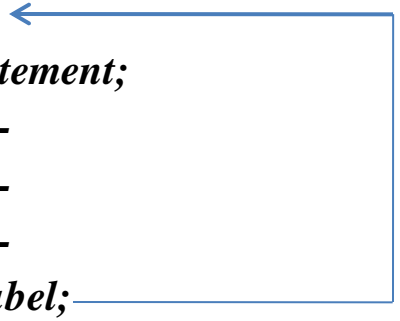
Forward Jump

label:

statement;

----

goto label;



Backward Jump

- The label can be anywhere in the program either before or after the goto statement.

Example

goto begin;

- During the execution of the program, when goto begin encountered then the control will immediately transferred to the specified label.
- A goto statement breaks the normal sequential execution of the program.

- If the label ; is before the goto statement then a loop will be formed and some statements will be executed repeatedly. Such jump is known as backward jump.
- On the other hand, if the label; is placed after the goto statement then some statement will be skipped and the jump is known as forward jump.
- A goto is often used at the end of a program to direct the control to go to the input statement, to read further data.

Example

```
#include <math.h>
main()
{
int n,count;
double x,y;
count=1;
printf("enter the value of n\n");
scanf("%d",&n);
printf("enter the %d no of values \n",n);
read:
    scanf("%lf",&x);
    printf("\n");
    if (x<0)
        printf("value =%lf is negative\m",x);
    else
    {
        y=sqrt(x);
        printf("%lf\t %lf\n",x,y);
    }
    count=count+1;
    if (count<=n)
        goto read;
    printf("\n end of the program");
}
```

CHAPTER VI

DECISION MAKING AND LOOPING

6.1 INTRODUCTION

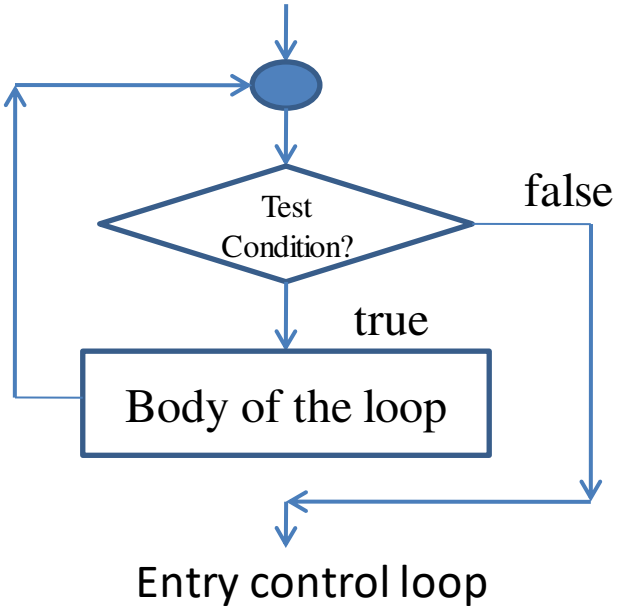
- A sequence of statement executed until some conditions for termination of the loop are satisfied.
- A program loop consists of two segments , body of the loop and control statement.
- The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.
- Depending on the position of the control statement in the loop, control structures may classified as the entry-control loop and exit-control loop.

Entry-control loop

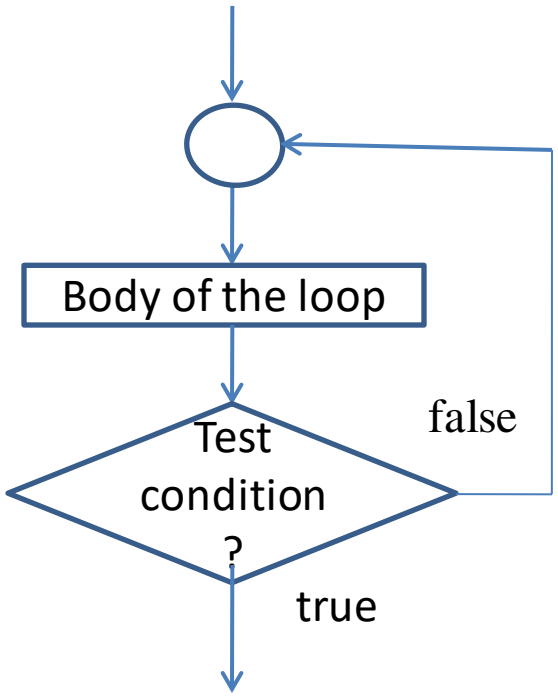
- In the entry-control loop, the control condition is tested before the start of the loop execution. If the condition is not satisfied then the body of the loop will not be executed.
- It is knon as pre-test loops

Exit-control loop

- The control condition is tested at the end of the body of the loop and therefore the body is executed unconditionally for the first time.
- It is known a s post-test loops.



Exit-control loop



- A looping process would include the following tour steps.

1. Setting and initialization of a condition variable.
2. Execution of the statements in the loop
3. Test for a specified value of the condition variable for execution of the loop.
4. Incrementing or updating the condition variable.

Looping statements

1. The while statement
2. The do statement
3. The for statement.

- Based on the nature of control variable and the kind of value assigned to it for testing the control expression, the loops may be two types.

I. Counter-controlled loops

II. Sentinel-controlled loops

i. Counter-controlled loop (repetition loop)

- No. of times the loop will be executed known in advance then we may use counter-control loop.
- A control variable as counter
- The counter must be initialized, tested and updated.
- No. of times the loop will be executed may be a constant or a variable that is assigned a value.

.

ii. Sentinel-controlled loop(indefinite loop)

- A special value called sentinel value is used to change the loop control expression from true or false.
- The control variable is called sentinel value.
- The no. of repetitions is not known before the loop begins executing

6.2 WHILE STATEMENT

while is entry-control loop.

Syntax

while (test condition)

{

Body of the loop

}

- The test condition is evaluated and if the condition is true then the body of the loop is executed.
- After execution of the body, the test-condition is once again evaluated and if it is true then the body of the loop is executed once again.
- The process of repeated execution of the body continues until the test-condition becomes false and the control is transferred out of the loop.
- On exit, the program continues with the statement immediately after the body of loop.
- The body of the loop may have one or more statements.
- The body of the loop may not be executed at all if the condition is not satisfied at very first attempt.

Example 1

```
main()
{
int i, n, sum;

printf("Enter the value of n\n");
scanf("%d", &n);
i=1;
sum=0;
while(i<=n)
{
        sum = sum + i;
        i = i+1;
}
printf("sum of natural numbers up to %d = %d", n, sum);
}
```

Example2

Program to evaluate x^n the equation

$$y = x^n$$

Program

```
main()
{
int count,n;
float x,y;
printf("enter the value of x and n" );
scanf("%f %d", &x ,&n);
y=1.0;
count=1;
while(count<=n)
{
    y=y*x;
    count++;
}
printf("\nx= %f; = %d; x to power n = f\n",x,n,y);
}
```

6.3 DO STATEMENT

Do statement is exit-controlled loop.

Syntax

```
do
{
    body of the loop
} while (test condition);
```

- The program proceeds to evaluate the body of the loop first.
- At the end of the loop, test condition in the while statement is evaluated.
- If the condition is true then the program continues to evaluate the body of the loop once again.
- This process continues as long as the condition is true.
- When the condition becomes false then the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

The body of the loop is always executed at least once.

Example

```
main()
{
int i, n, sum;

printf("Enter the value of n\n");
scanf("%d", &n);
i=1;
sum=0;
do
{
    sum = sum + i;
    i = i+1;
} while(i<=n);

printf("sum of natural numbers up to %d = %d", n,
    sum);
}
```

6.4 FOR STATEMENT

For statement is entry-controlled loop.

Syntax

for(initialization;test-condition,increment)

{

body of the loop;

}

The execution of the for statement is as follows

1. Initialization of the control variable is done first, using assignment statement.
2. The value of the control variable is tested using the test-condition. The test-condition is a relational expression. If the condition is true then body of the loop is executed. Otherwise, the lopp is terminated and the execution continues with the statement that immediately follows the loop.
3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. The control variable is incremented using the assignment statement and the new value of the control variable is again tested to see whether it satisfies the loop condition. If the condition is true then the body of the is executed. This process is continues till the value of the control variable fails to satisfies the test-condition.

Example

```
main()
{
int i, n, sum;

printf("Enter the value of n\n");
scanf("%d", &n);
sum=0;
for (i=1;i<=n;i++)
{
    sum = sum + i;
}
printf("sum of natural numbers up to %d = %d", n, sum);
}
```

- The for statement allows the negative increments.

Example

```
for(x=9;x>=0;x--)
    printf("%d",x);
printf("\n");
```

- The above loop is executed 10 times, but the output would be from 9 to 0 instead of 0 to 9.
- Since the conditional test is always performed at the beginning of the loop, the body of the loop may not be executed at all, if the condition fails at the start.

Example

```
for(x=9;x<9;x--)
    printf("%d",x);
```

- The above loop never executed because the test condition fails at the beginning itself.

COMPARISON OF THE THREE LOOPS

for	while	do

for(n=1;n<=10;n++)
{
 	
---------- 	
---------- 	
---------- 	
 	
}
	n=1;
	while(n<=10)
	{
	---------- 	
	---------- 	
	---------- 	
	n=n+1;
	}
		n=1;
		do
		{
		---------- 	
		---------- 	
		---------- 	
		n=n+1;
		}while(n<=10);

Example1:Program to generate prime numbers between 1 and n.

```
#include<stdio.h>
#include <conio.h>
void main()
{
int i,j,n;
lbl: printf("enter the value of n\n");
 scanf("%d",&n);
if (n <1)
{
      printf("invalid no. try again");
      goto lbl;
}
printf("prime numbers are \n");
 if (n>0 && n <=2)
{
      printf("the prime no is 2");
      goto end;
}
for (i=3;i<=n;i++)
{
  for (j=2;j<i;j++)
    {
         if (i %j ==0)
               goto m;
     }
     printf("%d",i);
m:
 }
end: getch();
}
```

Example2:

Program To Print N Fibonacci Number

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n1=0,n2=1,n,i,fib;
clrscr();
printf ("enter the value of n\n");
printf("%d",&n)
printf("%d\t%d",n1,n2);
for(i=1;i<=n-2;i++)
{
fib=n1=+n2;
printf("\t%d",fib);
n1=n2;
n2=fib;
}
getch();
}
```

Additional features of for loop

1. **More than one variable can be initialized at a time in the for statement.**

Example

```
p=1;
for(n=0;n<10;n++)
```

Can be written as

```
for(p=1,n=0;n<10;n++)
```

2. **The increment section may also have more than one part.**

Example

```
for (n=0;n<10;n++,m--)
```

3. **Test condition may have any compound relation and the testing need not be limited only to the loop control variable.**

Example

```
sum=0
for (i=0;i<10&& sum <100;i++)
{
sum=sum+i;
}
```

4. **Expression can be used in the assignment statements of initialization and increment section.**

Example

```
for(x=(m+n)/2;x>0;x=x/2)
```

5. **One or more section can be omitted in for loop.**

```
----
m=5;
for(;m<10;)
{
        printf("%d\n",m);
        m=m+2;
}
```

6. **We can set up the time delay loops using for statement.**

Example

```
for(j=1;j<1000;j++);
```

Nesting of for loops

- One for statement within another for statement is called nesting of for.

Example

```
----------
----------
for (i=0;i<n;i++)
{
    for(j=1;j<m;j++)
    {
            ----------
            ----------
    }
    ----------
    ----------
}
----------
----------
```

Example

Program to read the marks by n students in various subject and print the total.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n, m, i, j, rno, marks, total;
printf("Enter the no. of students & subjects\n");
scanf("%d%d",&n,&m);
for(i=1;i<=n;i++)
{
    printf("Enter the roll no. :");
    scanf("%d",&rno);
    total=0;
    printf("Enter marks of %d subjects for roll
    no %d\n", m, rno);
    for(j=1;j<=m;j++)
    {
    scanf("%d",&marks);
    total= total+marks;
    }
    printf("total marks =%d" ,total);
}
getch();
}
```

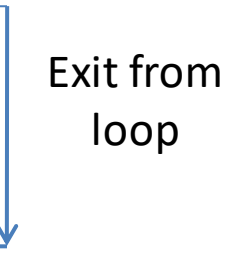
6.5 JUMPS IN LOOP

- Loops perform a set of operations repeatedly untill the control variable fails to satisfy the test-condition.
- The no. of times the loop is repeated is decided in advance and the test condition is written to achieve this.

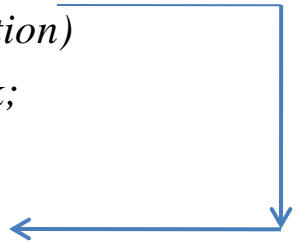
Jumps out of loop

An early exit from a loop can be accomplished by using the break statement or the goto statement.

```
while(....)
{
--------
--------
 if (condition)
      break;
--------
--------
}
--------
```




```
for(....)
{
--------
--------
 if (condition)
      break;
--------
--------
}
--------
```



```
do
{
-
-------
-------
if (condition)
      break;
-------
-------

}while(-----);
-------
```

Exit from loop

```
for(------)
{
-------
      for(------)
      {
                -------
                -------
                if (condition)
                          break;
                -------
                -------
      }
-------
-------
}
```

Exit from inner loop

- goto statement can transfer the control to any place in a program.

```
while(----)
{
    if (error)
        goto stop;
    --------
    if (condition)
        goto lbl;
    --------
    --------
    lbl: --------
    --------
}
stop:
    --------
    --------
```

Jump within loop

Exit from loop

```
for(----)
{
    for(----)
    {
        --------
        --------
        if (error)
            goto stop;
        --------
        --------
    }
--------
--------
}
stop:
--------
--------
```

Exit from two loops

6.6 STRUCTURED PROGRAMMING

- Basic three control structures

1. Sequence structure
2. Selection(branching) structure
3. Repetition(looping) structure

Advantage

Ensure well-designed programs that are easier to write, read ,debug and maintain compare to unstructured.

Skipping a part of loop

continue

- Break statement terminate the loop.
- Continue statement causes the loop to be continued with the next iteration after skipping any statements in between.
- The continue statement tells the compiler "skip the following statements and continue with the next iteration".

syntax

```
    continue;
while(test-condition)
{
    --------
    --------
    if(------)
        continue;
    --------
    --------
}
do
{
--------
if (-----)
    continue;
--------
}while(test-condition);
```

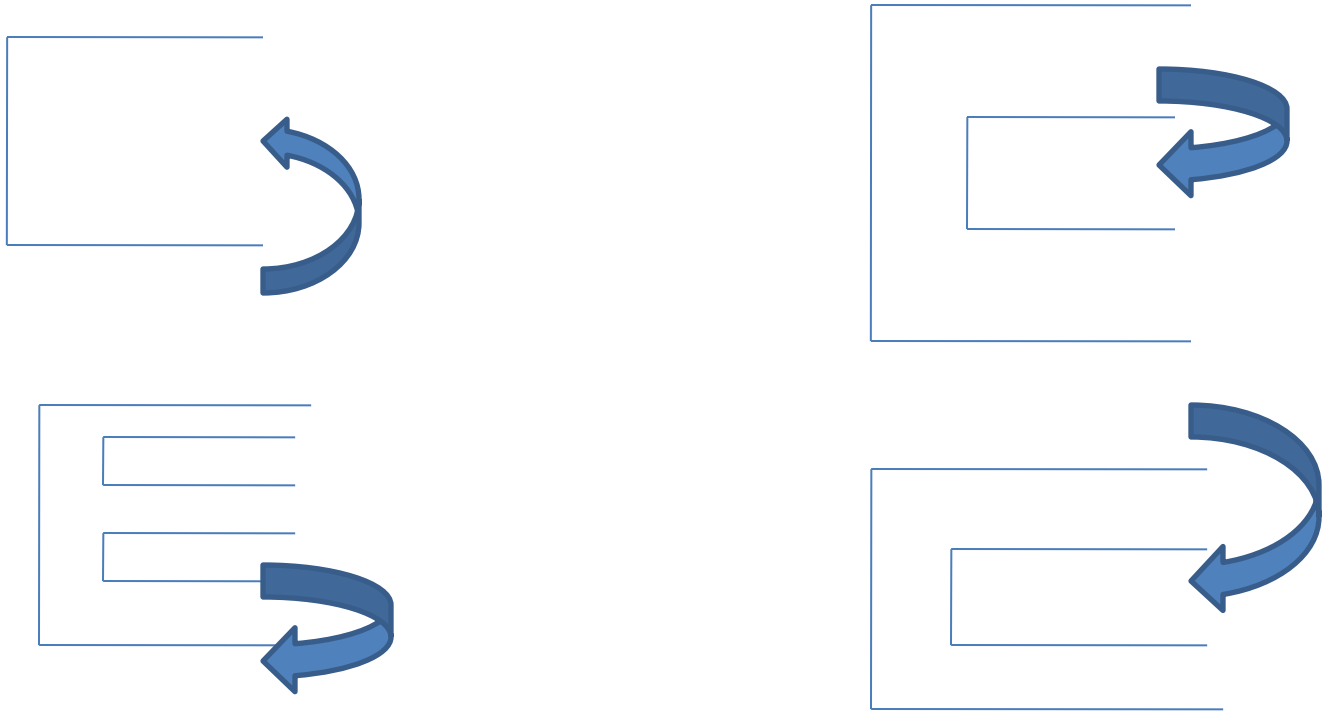
```
for(initialization;test-condition;increment)
{
      -------
      -------
      if(-----)
            continue;
      -------
      -------
}
```

Example

Program to illustrate the use of continue statement

```
#inckude<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int count,neg;
double no,sqroot;
printf("enter 9999 to stop\n");
count=0;
neg=0;
while(count<=100)
{
printf("Enter a number:");
scanf("%d",&no);
if (no==9999)
      break;
if (no<0)
{
printf("No. is negative\n");
neg++;
continue;
}
sqroot=sqrt(no);
printf("No=%lf   Square
      root=%lf",no,sqroot);
count++;
}
printf("No. of items done : %d",count);
printf("\nNo. of negative nos :%d",neg);
}
```

Avoiding goto



REFERENCE

- E. Balagurusamy, ”Programming in ANSI C”, Seventh Edition, McGraw Hill Education India Private Ltd, 2017.