Programming In C

Control Statements

Objectives of these slides:

- **to introduce the main kinds of C control flow**

Control Structures

- There may be situations where the programmer requires to alter normal flow of execution of program or to perform the same operation a no. of times.
- Various control statements supported by c are-
  - Decision control statements
  - Loop control statements

Decision Control Statements

- Decision control statements alter the normal sequential execution of the statements of the program depending upon the test condition to be carried out at a particular point in program.
- Decision control statements supported by c are:-
  - if statement
  - if-else statement
  - Else if Ladder
  - Nested If
  - switch statement

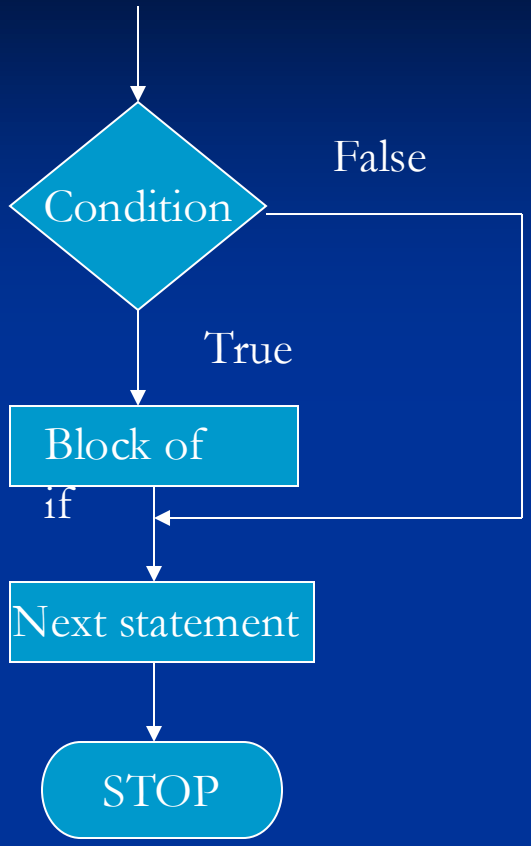
Decision Control Statements

- if statement
  - Most simple and powerful decision control statement.
  - It executes a statement or block of statements only if the condition is true.
  - Syntax: if (condition)
    {
    statement (s);
    }
    Next statement;

    OR

    if (condition)
    {
    statement 1;
    statement 2;
    }
    statement 3;
  - In above syntax : if condition is true only then the statements within the block are executed otherwise next statement in sequence is executed.

Flowchart

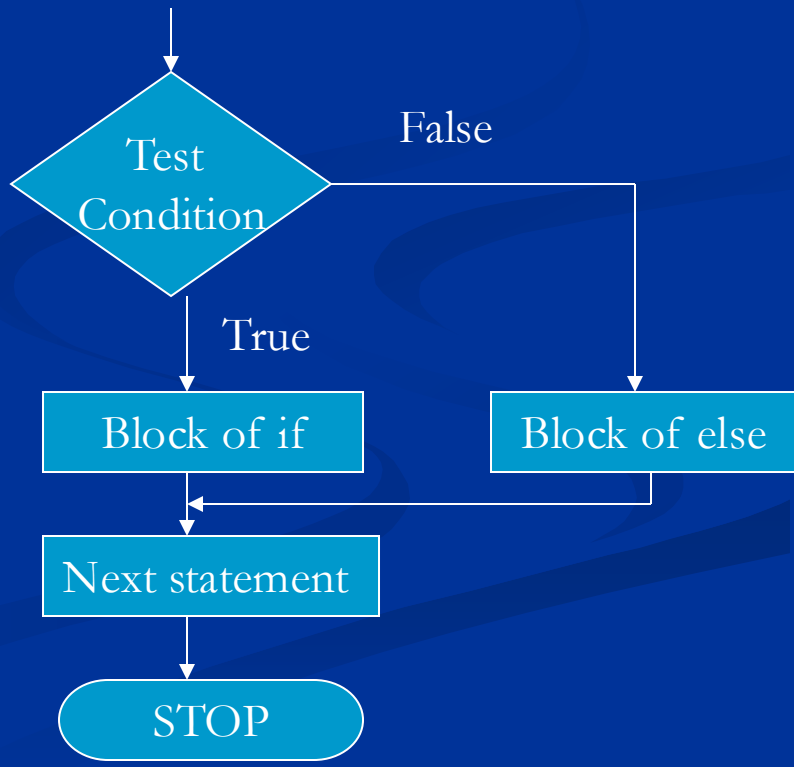



```
/* Program to check whether a no. is even */
# include<stdio.h>
# include<conio.h>
 void main()
{
int num;
clrscr();
printf("enter the number");
scanf("%d",&num)
if(num%2==0)
{
printf("\n Number is even");
}
printf(" End of program");
getch();
}
```

if – else statement

- In case of if statement, the block of statements is executed only when the condition is true otherwise the control is transferred to the next statement following if block.
- But if specific statements are to be executed in both cases (either condition is true or false) then if – else statement is used.
- In if – else statement a block of statements are executed if the condition is true but a different block of statements is executed when the condition is false.
- Syntax: if (condition)

```
        {
                statement 1;
                statement 2;
        }
      else
        {
                statement 3;
        }
```



Exercise: WAP to check whether a given no. is even or odd?

Nested if – else statement

- When an entire if-else is enclosed within the body of if statement or/and in the body of else statement, it is known as nested if-else statement.
- The ways of representing nested if –else are-

```
if (condition1)
{
if (condition2)
    statement 1;
else
    statement 2;
}
else
 statement 3;
```

```
if (condition1)
{
if (condition2)
   statement 1;
else
   statement 2;
}
```

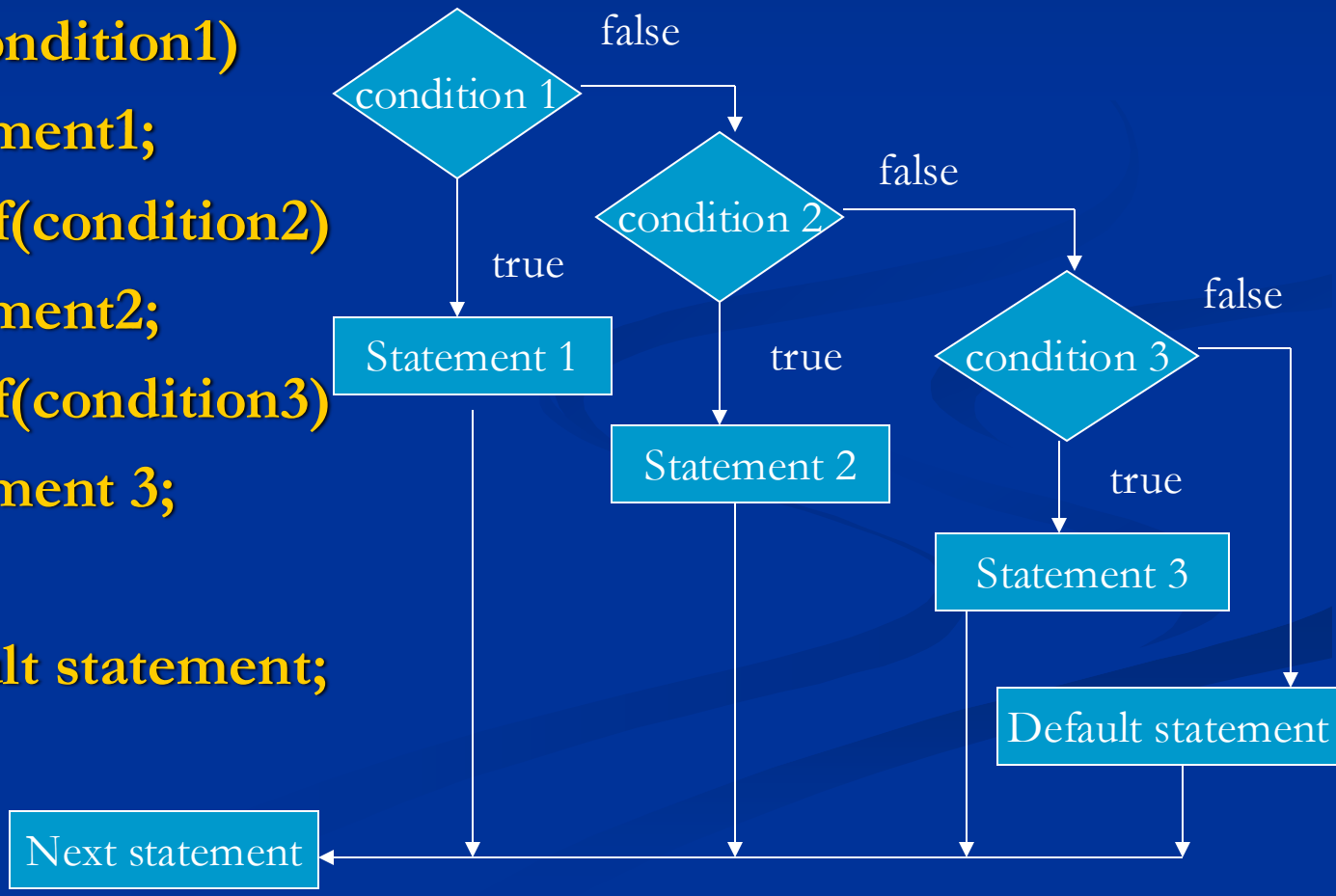
```
else
{
if (condition 3)
   statement 3;
else
   statement 4;
}
```

```
if (condition1)
   statement 1;
else
{
if (condition2)
   statement 2;
else
   statement 3;
}
```

If- else- if ladder

- In a program involving multiple conditions, the nested if else statements makes the program very difficult to write and understand if nested more deeply.
- For this ,we use if-else-if ladder.
- Syntax: if (condition1)

```
        statement1;
        else if(condition2)
        statement2;
        else if(condition3)
        statement 3;
        else
        default statement;
```



Switch statement

- Switch is a multi-way decision making statement which selects one of the several alternatives based on the value of single variable or expression.
- It is mainly used to replace multiple if-else-if statement.
- The if-else-if statement causes performance degradation as several conditions need to be evaluated before a particular condition is satisfied.
- Syntax:

```
switch (expression)
{
case constant1 : statement (s); [break;]
case constant2 : statement (s); [break;]
……..……..……..……..……..……..
default: statement (s)
}
```

Break statement

- Break statement terminates the execution of the loop in which it is defined.
- The control is transferred immediately to the next executable statement after the loop.
- It is mostly used to exit early from the loop by skipping the remaining statements of loop or switch control structures.
- Syntax: break;

Looping Structures

- When we want to repeat a group of statements a no. of times, loops are used.
- These loops are executed until the condition is true.
- When condition becomes false, control terminates the loop and moves on to next instruction immediately after the loop.
- Various looping structures are-
  - while
  - do – while
  - for

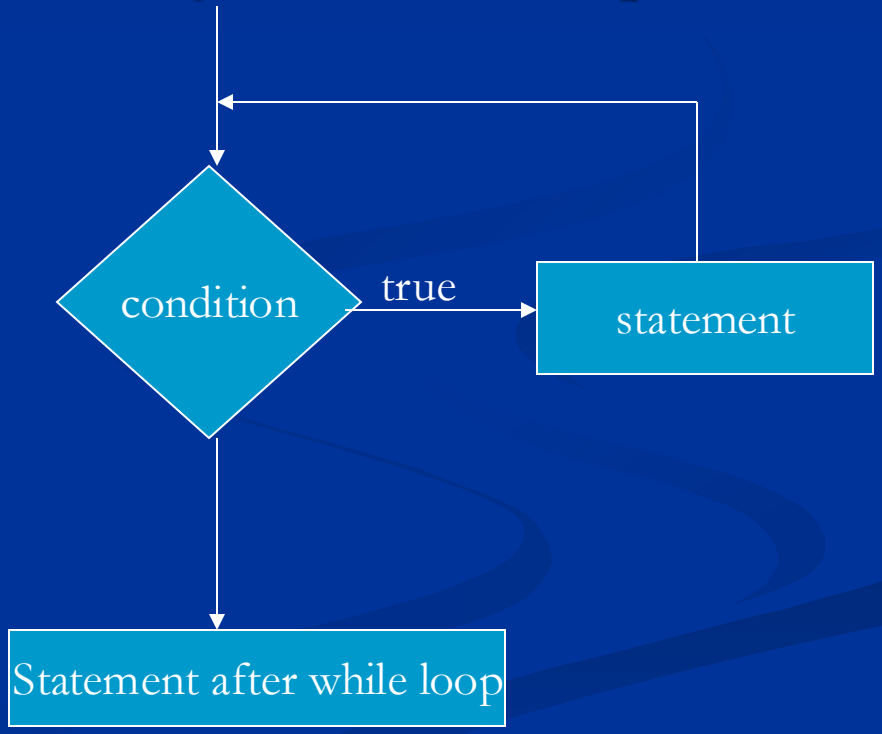
LOOPING STATEMENTS

- Loop is divided into two parts:
  - Body of the loop
  - Control of loop
- Mainly control of loop is divided into two parts:
  - Entry Control loop (while, for)
  - Exit Control loop (do-while)

while statement

- While loop is used to execute set of statements as long as condition evaluates to true.
- It is mostly used in those cases where the programmer doesn't know in advance how many times the loop will be executed.
- Syntax: while (condition)

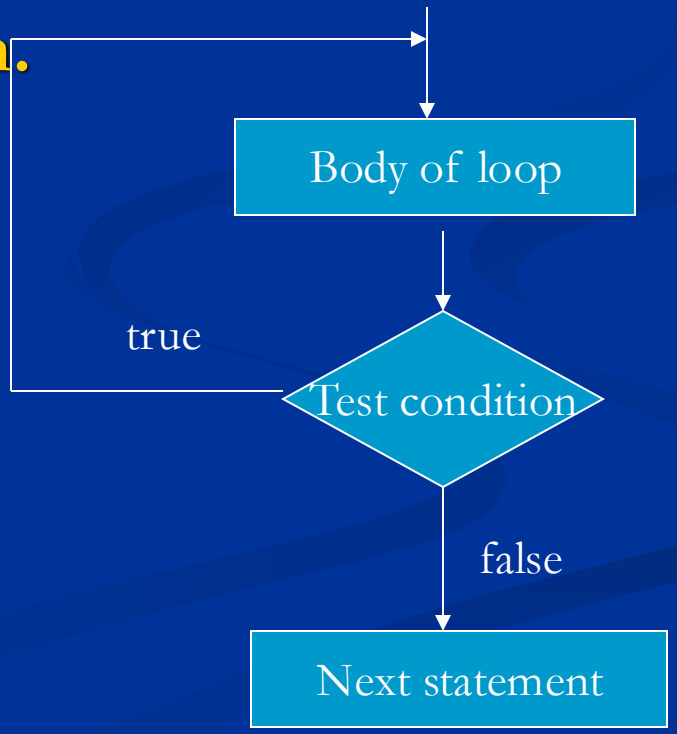
```
{
    Statement 1 ;
    Statement 2 ;
}
```



do- while

- do-while is similar to while except that its test condition is evaluated at the end of the loop instead at the beginning as in case of while loop.
- So, in do-while the body of the loop always executes at least once even if the test condition evaluates to false during the first iteration.
- Syntax: do

```
{
    statement 1;
    statement 2;
}while (condition);
statement;
```



for loop

- Most versatile and popular of three loop structures.
- Is used in those situations when a programmer knows in advance the number of times a statement or block will be executed.
- It contains loop control elements all at one place while in other loops they are scattered over the program and are difficult to understand.
- Syntax:-

```
for (initialization; condition; increment/decrement)
{
Statement( s);
}
```

The `for` is a sort of `while`

```
for (expr1; expr2; expr3)
  statement;
```

is equivalent to:

```
expr1;
while (expr2) {
  statement;
  expr3;
}
```

Various other ways of writing same for loops

```
i = 1
 for (; i<=15;i ++)
{
……..
}
```

```
for (i=1; ;i++)
{
……..
if (i>15)
break;
……
}
```

```
for (i=1;i<=15;)
{
…….
i++;
}
```

Some Examples

```
for(i = 7; i <=77; i += 7)
    statement;
for(i = 20; i >= 2; i -= 2)
                statement;

for(j = 10; j > 20; j++)
    statement;
for(j = 10; j > 0; j--)
    statement;
```

Incrementing and Decrementing

Incrementing

- Add 1 to c by writing:

```
c = c + 1;
```

Also: c += 1;

Also: c++;

Also: ++c;

```
/* Preincrementing and postincrementing */

#include <stdio.h>

int main()
{
   int c;

   c = 5;
   printf("%d\n", c);
   printf("%d\n",c++); /*post increment*/
   printf("%d\n\n", c);
         :
```

continued

```
 c = 5;
  printf("%d\n", c);
  printf("%d\n",++c);  /*pre-
increment*/
  printf("%d\n", c);
  return 0;
}
```

Output:

```
5
5
6
5
6
6
```

Decrementing

- Take 1 from c by writing:

```
c = c - 1;
```

Also:
```
c -= 1;
```

Also:
```
c--;
```

Also:
```
--c;
```

Continue statement

- Like break ,continue statement also skips the remaining statements of the body of the loop where it is defined but instead of terminating the loop, the control is transferred to the beginning of the loop for next iteration.
- The loop continues until the test condition of the loop become false.
- Syntax: continue;
- E.g.

```
for (m=1;m<=3;m++)
        {
            for (n=1;n<=2;n++)
            {
              if (m==n)
              continue;
              printf(" m=%d  n=%d");
            }
        }
```

Output:

1 2

2 1

3 1

3 2

goto Statement

- An unconditional control statement that causes the control to jump to a different location in the program without checking any condition.
- It is normally used to alter the normal sequence of program execution by transferring control to some other part of the program.
- So it is also called jump statement.
- Syntax: goto label;
- Label represents an identifier which is used to label the destination statement to which the control should be transferred.

  label : statement;
- The goto statement causes the control to be shifted either in forward direction or in a backward direction .

exit() function

- C provides a run time library function exit() which when encountered in a program causes the program to terminating without executing any statement following it.
- Syntax: exit(status);

  Status is an integer variable or constant.
- If the status is 0,then program normally terminates without any errors.
- A non-zero status indicates abnormal termination of the program.
- The exit() function is defined in the process.h header file.

Difference b/w exit() & break

- Exit() is used to transfer the control completely out of the program whereas break is used to transfer the control out of the loop or switch statement.