CHAPTER 9

FLOW OF CONTROL

FLOW CONTROL

- In a program statement may be executed sequentially, selectively or iteratively.

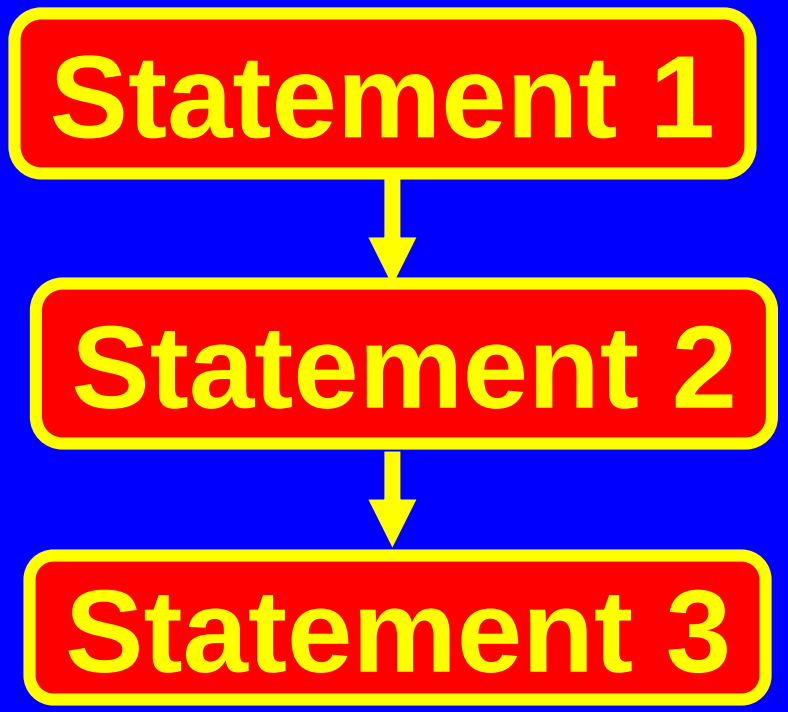
- Every program language provides constructs to support sequence, selection or iteration.

SEQUENCE

- Sequence construct mean statement are executed sequentially.

- Every program begins with the first statement of main(). Each statement in turn executed sequentially when the final statement of main() is executed the program is done.

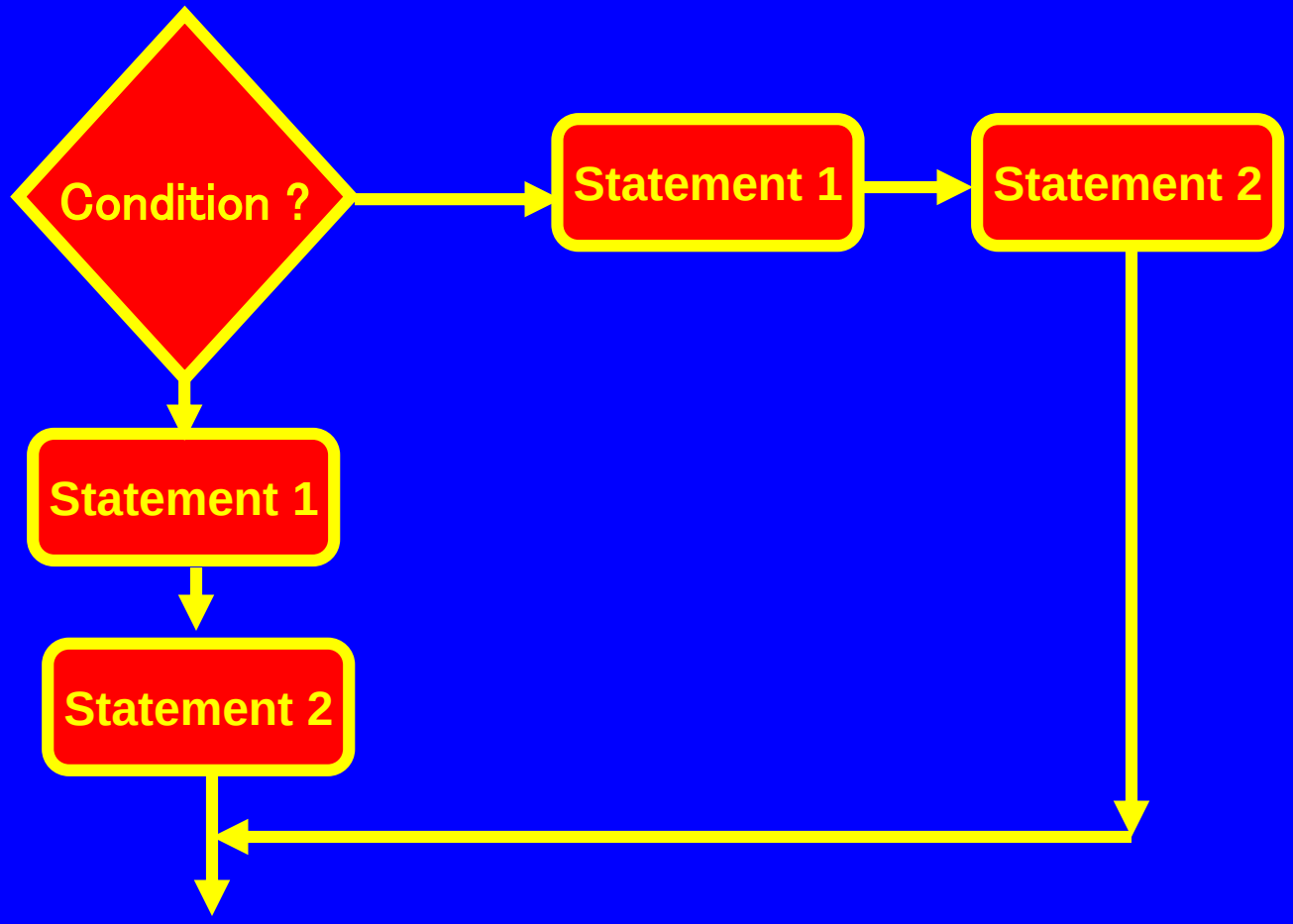
THE SEQUENCE CONSTRUCT



SELECTION

- The Selection construct means the execution of statement(s) depending upon a condition-test. If a condition evaluates to true, a course-of-action (a set of statements) is followed otherwise another course-of-action (a different set of statements).
- This construct(selection construct) is also called decision construct because it helps in making decision about which set-of-statements is to be executed.

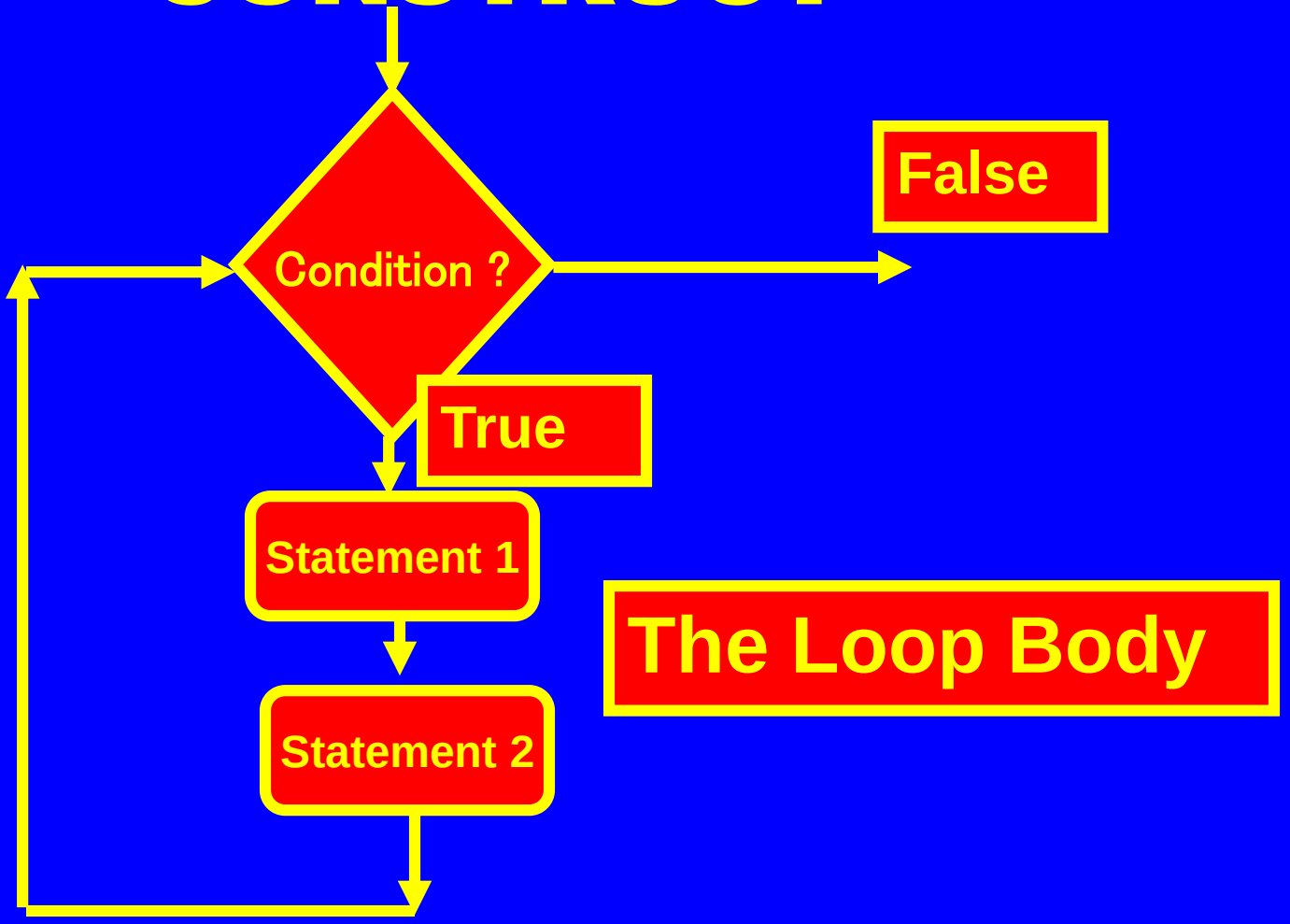
THE SELECTION CONSTRUCT.



ITERATION

- Iteration construct means repetition of set of statements depending upon a condition test. Till the time of condition is true. (or false depending upon the loop). A set of statements are repeated again and again. As soon as the condition become false (or true), the repetition stops. The iteration condition is also called **"Looping Construct"**.

THE ITERATION CONSTRUCT



THE SELECTION STATEMENT – if Statement

- An if statement test a particular condition, if the condition evaluated to true, a course of action is followed, i.e., a statement or a set of statement is executed. Otherwise if the condition evaluated to false then the course of action is ignored.

SYNTAX OF IF STATEMENT

- if (condition)

  statement 1;

The statement may consist of single or compound. If the condition evaluates non zero value that is true then the statement 1 is executed otherwise if the condition evaluates zero i.e., false then the statement 1 is ignored.

Example of if statement

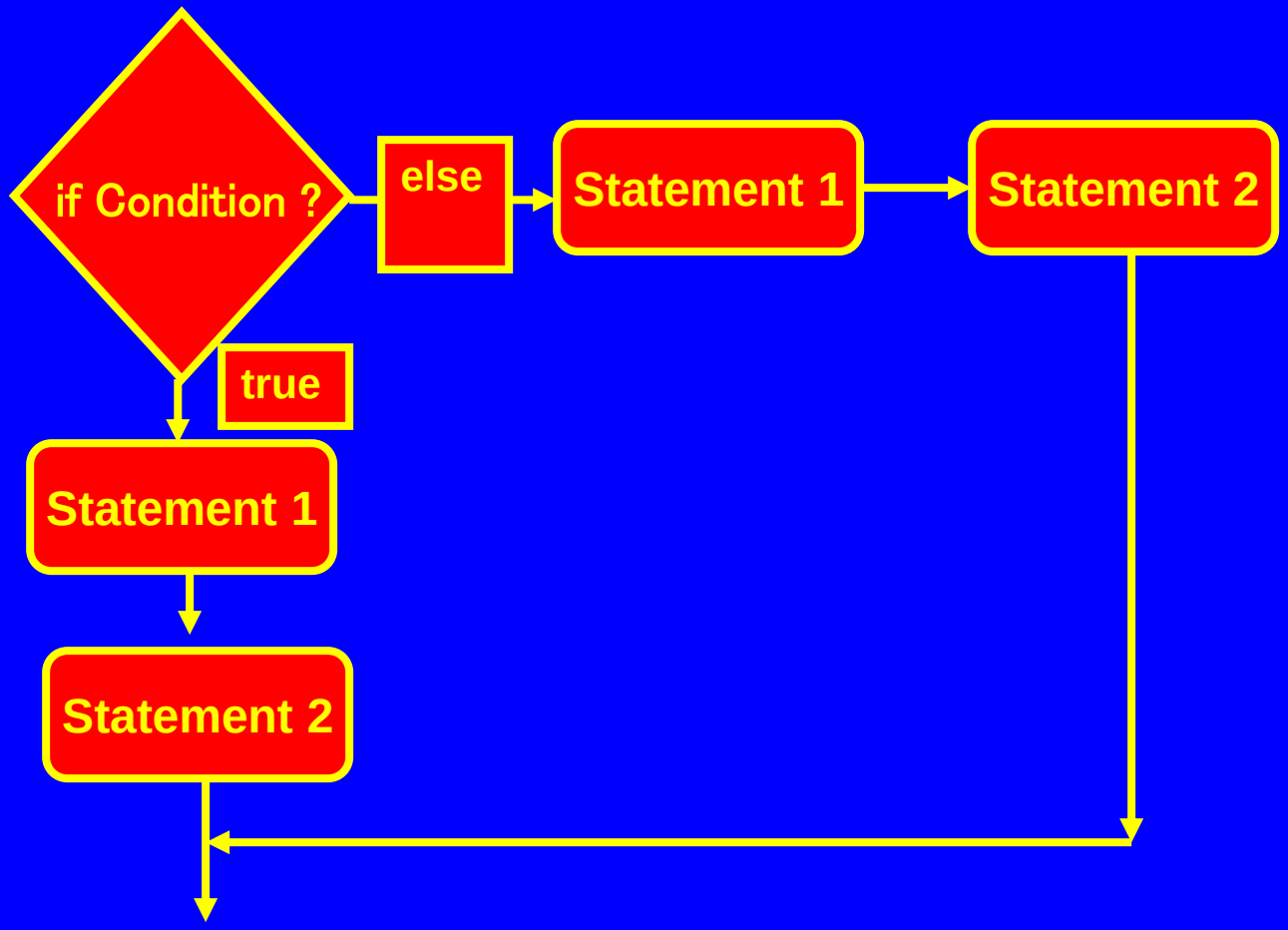
Example 1:

```
if (age>18)
cout<<“The person is eligible for vote”
```

Example 2:

```
if(ch==‘ ‘)
spaces++;
```

Flow chart of if statement



IF - ELSE FORMAT

```
if (condition)
{
        Statement 1
        Statement 2
}
else
{
    Statement 1
    Statement 2
}
```

Example of if-else

```
If (basic>8000)
{
        total_da=(40*basic)/100
        gross=total_da + basic
}
else
{
        total_da=(40*basic)/100
        gross=total_da + basic
}
```

NESTED IFs

- A nested if is an if that has another if in its body or in its else body. The nested if can have one of the following three forms

Form 1 :

```
if  (expression 1)
{
        if  (expression 2)
                statement 1
        else
                statement 2
}
else
   body of else
```

NESTED IF contd..

- Form 2:

```
if  (expression 1)
        {
                if  (expression 2)
                        statement 1
                else
                        statement 2
        ……..
        }
        else
        {
                if  (expression 2)
                        statement 1
                else
                        statement 2
                ……..
        }
```

NESTED IF contd..

- Form 3:

```
if  (expression 1)
        {
                body of if
        }
        else
        {
                if  (expression 2)
                        statement 1
                else
                        statement 2
                ..........
        }
```

Program to create the equivalent of a four function calculator

```
#include<iostream.h>
#include<conio.h>
int main()
{
clrscr();
char ch;
float a,b, result;
cout<<"Enter the two values" ;
cin>>a>>b;
cout<<"Enter the Operator [ + - * / ] : ";
```

Program Contd..

```
cin>>ch;
if(ch=='+')
result=a+b;
else
if(ch=='-')
result=a-b;
else
if(ch=='*')
result=a*b;
else
if(ch=='/')
```

Program Contd..

```
result=a/b;
else
cout<<"Unknown Operation ";
cout<<"\nThe Resultis : "<<result;
getch();
return 0;
}
```

THE if-else-if LADDER

- A common programming construct in C++ is the if-else-if ladder, which is often also called as the if-else-if ladder because of its appearance. It takes the following general form.

```
if  (expression 1)          statement 1;
                  else
                     if  (expression 2)  statement 2
                  else
                     if (expression 3)  statement 3
                              …….
                  else
                     Statement 4;
```

THE ? : ALTERNATIVE TO if

- C++ has an operator that can be alternative to if statement. The **conditional operator ? :**
- This operator can be used to replace the if statement of C++.

CONDITIONAL OPERATOR ? :

```
if  (expression 2)
                    statement 1
          else
                    statement 2
```

- The above form of if else statement can be replaced as,

  expression1?expression2:expression3;

CONDITIONAL OPERATOR ? :

- For example

```
int c;
if (a>b)
        c=a;
else
        c=b;
```

This can be alternatively written as,

```
int c;
c=a>b?a : b;
```

COMPARISON OF if AND ? :

1. compared to if –else sequence, ?: offers more concise, clean and compact code, but it is less obvious as compared to if.
2. Another difference is that the conditional operator ?: produces an expression, and hence a single value can be assigned or incorporated into a larger expression, where as if is more flexible. if can have multiple statements multiple assignments and expressions (in the form of compound statement) in its body.
3. When ?: operator is used in its nested form it becomes complex and difficult to understand. Generally ?: is used to conceal (hide) the purpose of the code.

THE switch STATEMENT

- C++ provides **multiple- branch selection statement** known as **switch**

  This selection statement successively tests the value of an expression against the list of integer or character constants. When a match is found, the statements associated with that construct are executed.

THE switch STATEMENT

- The syntax is,

```
switch(expression)
{
case constant 1 :statement sequence 1;
                        break;
case constant  2 : statement sequence 2;
                          break;
case constant n-1 :statement sequence n-1;
                            break;
default:      statement sequence n;
                      break;
}
```

fall through

- The expression is evaluated and its values are matched against the values of the constants specified in the case statements. When the match is found, the statement sequence associated with that case is executed until the break statement or the end of switch statement is reached. If a case statement does not include break statement then the control continues right on the next case statement(s) until either a break is encountered or end of switch is reached this situation(missing break in case statement) is known as **“fall through”.**

default STATEMENT

- The default statement gets executed when there is no match found. The default is optional, and if it is missing then no action takes place if all matches fail.

Example of switch

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int dow;
cout<<“Enter the number of week’s day”;
cin>>dow;
switch(dow)
{
case 1 : cout<<“\n Sunday”;
break;
```

Example of switch

```
case 2 : cout<<“\n Monday”;
break;
case 3 : cout<<“\n Tuesday”;
break;
case 4 : cout<<“\n Wednesday”;
break;
case 5 : cout<<“\n Thursday”;
break;
```

Example of switch

```
case 6 : cout<<“\n Friday”;
break;
case 7 : cout<<“\n Saturday”;
break;
default :cout<<“Wrong number of day”
break;
}
getch();
}
```

OUT PUT

Enter the number of week's day 5

Thursday

THE switch Vs. if-else

- The switch and if-else are selection statements and they both let you select an alternative out of many alternatives by testing an expression. However there are some differences in their operation and they are,

1. The switch statement differs from the if statement in that switch can only test for equality where as if can evaluate a relational or logical expressions i.e multiple conditions.

THE switch Vs. if-else

2. The switch statement selects its branches by testing the value of same variable (against the set of constants) where as the if else construction lets you to use a series of expressions that may involve unrelated variables and complex expressions.

THE switch Vs. if-else

3. The if-else is more versatile of two statements where as switch cannot. Each switch case label must be a single value.

4. The if-else statement can handle floating point tests also apart from integer and character tests where as switch cannot handle floating point tests. The case labels of switch must be an integer or character.

The Nested Switch

- Like if statement, switch can also be nested. For example following code fragment is perfectly all right in C++.

The Nested Switch

```
switch (a)
{
   case 1: switch(b)
                {
                        case 0 : cout<<“Divide by zero error”;
                            break;
```

The Nested Switch

```
                case 1 : res=a/b;
                         break;
                      } // inner  switch end
              break;        // outer switch case 1's break
        case 2 :            //outer switch case 2
        ……
        ……
}                           // outer switch end.
```

More about Switch

1. A switch statement can only work for equality comparisons.
2. Now two case labels in the same switch can have the identical values but in case of nested witch the case constants of inner and outer switch can contain common values.

More about Switch

3. If a characters constants are used in switch statements, they are automatically converted into integers (equivalent ASCII codes).
4. The switch statement is more efficient than if in a situation that supports the nature of switch operation.

More about Switch

- For example a statement that tests values against a set of constants like this,

```
if (wish==‘a’)
{ …..
    …..
}
else if (wish ==‘b’)
{ …..
   …..
}
```

More about Switch

```
else if (wish ==‘c’)
 { …..
   …..
}
else
{ …..
   …..
}
```

More about Switch

is better written as a switch statement as,

```
switch(wish)
{       case ‘a’:     ……
                      ……
                      break;
        case ‘b’ :     ……
                       ……
                      break;
```

More about Switch

```
        case ‘c’:    ……
                     ……
                   break;
        default :    ……
                     ……
                   break;
}              //end of switch body
```

NOTE

Always put break statement after the last case statement in switch.

ITERATION STATEMENT

- The iteration statement allows instructions to be executed until a certain condition is to be fulfilled.
- The iteration statements are also called as **loops** or **Looping statements.**
- C++ provides three kinds of loops
- for
- while
- do-while

Elements that control a Loop

- Every loop has its elements that

  control and govern its execution.

  Generally a loop has four elements that

  have different purposes, they are,

1. INITILIZATION EXPRESSIONS

Before entering in a loop, its control variable must be initialized. The initialization expression executed at only once.

2. TEST EXPRESSION

- The test expression is an expression whose truth values decides weather the loop- body will be executed or not. If the test expression evaluates to true I.e., the loop gets executed, otherwise the loop terminated.

3. UPDATED EXPRESSION

The update expression change the value(s) of loop variable(s). The update expression(s) is executed; at the end of the loop after the loop-body is executed.

4. THE BODY OF THE LOOP

The statements that are executed repeatedly as long as the value of expression is non zero. If it evaluates to zero then the loop is terminated.

THE for LOOP

The for loop is the easiest to understand of the C++ loops. The general form of for loop is,

for(initialization expression(s); test expression;update expression)
body of for loop

Example :for LOOP

For example:

```
#include<iostream.h>
int main()
{
int i;
for  (i=1;          i<=10;   ++i)   // do not give semicolon here.
cout<<“\n”<<i;
return 0;
}
```

NOTE: NO SEMICOLON IN FOR STATEMENT

Example :for LOOP

Initilization exp	Test Exp	Update Exp
for (i=1;	i<=10;	++i)
cout<<“\n”<<i;	Body of the loop	

THE for LOOP VARIATIONS

- C++ offers several variations that increase the flexibility and applicability of for loop

1.MULTIPLE INITIALIZATION & UPDATE EXPRESSIONS.

- A for loop may contain multiple initialization and multiple update statements.

For example:

```
for(i=1,sum=0;I<=n; sum+=i,++i)
        cout<<"\n"<<i;
```

2. PREFER PREFIX INCREMENT / DECREMENT OPERATOR OVER POSTFIX WHEN TO BE USED ALONE.

- When you have to simply increment or decrement value of variable by one, then prefer prefix over postfix that is for ++i or –i. The reason being that when used alone, prefix faster executed than postfix. i.e.,
- for(i=1;i<n;++i) // prefer this

Over this, for(i=1;i<n;i++)

3. OPTIONAL EXPRESSIONS

- In a for loop initialization expression, test expression and update expression are optional. i.e., you can skip any or all of these expressions.
- for example you have initialize the variables you want to scrap off the initialization expression then you can write as,
- for(; test expression;update expression)

3. OPTIONAL EXPRESSIONS

For example,

```
int i=0,sum=0;

for(;i<=n; sum+=i,++i)

        cout<<“\n”<<i;
```

4. INFINITE LOOP

- An infinite for loop can be created by omitting the test expressions.
- For example,

```
for(j=25;    ;    --i)
cout<<“An infinite Loop”;
```

Similarly the following for loop also infinite loop

```
for( ; ;)
cout<<“Endless for loop;
```

NOTE: PRESS CTRL + BREAK TO TERMINATE THE PROGRAM EXECUTION

5. EMPTY FOR LOOP

- If a loop does not contain any statement in its loop-body, it is said to be an empty for loop.
- For example,
- for(j = 20; (j) ; --j); // j tests for non
  //zero value of j.
- See here the loop body contains null statement. An empty for loop has an application in pointer manipulation where you need to increment or decrement the pointer position without doing anything else.

TIME DELAY LOOPS

- Time delay loops are often used in the programs. It is created by using for loop
- for example,
- For(t=0;t<300;++t);

  That means if you put a semicolon after for's parenthesis it repeats only for counting the control variable and if you put a block of statements after such a loop then it is not a part of for loop.

TIME DELAY LOOPS

For example,

```
for(i=0;i<10;++i) ;   ← this semicolon ends
                        the loop here only.
{
cout”l=“<<i;<<endl;   this is not a body of
}                       for loop.
```

6. DECLARATION OF VARIABLES IN THE LOOP

- C++ allows to declare variables anywhere in a program. So they are generally declared immediately before there first reference.
- For example

```
for(int i=0;i<10;++i)
```

NOTE : Variables can be accessed only in the block where it has been declared.

VARIABLE'S SCOPE

- The program area inside which a variable can be accessed, is called **variable's scope**.

THE SCOPE OF LOCAL LOOP VARIABLE

- Up till now, a variable declared in the for or while loop could be accessed after the statement because the variable declaration had not taken place within the braces of the loop block, the item would still be in scope when the loop terminates. That means the same variable could not be declared in another loop in the same scope.

THE SCOPE OF LOCAL LOOP VARIABLE

- For example,

```
for(char ch='a'; ch<='z'; ++ch)
{
    ……
    ……
}
cout<<ch;  // ch was still valid. It was still in the
                                    //scope
for(char ch='a'; ch<='z'; ++ch)                    // Wrong!!!
{
    ……
    ……
}
```

THE SCOPE OF LOCAL LOOP VARIABLE

- As per the latest ANSI/ISO specifications, the variables declared inside the parenthesis of for and while loops are not accessible after the loop is over.but this would be implemented in newer versions of the compilers.

THE while LOOP – AN ENTRY CONTROLLED LOOP

- The second loop available in C++ is the while loop. The while loop is an entry controlled loop the syntax of while is,

  while(expression)

  Loop body

  Where, loop body contain the single statement or set of statements (compound statement) or an empty statement. The loop iterates while the expression evaluates to **true,** when expression becomes **false** the loop terminates.

VARIATIONS IN while LOOP

- A while loop can also have variations.it can be

1. Empty loop : it does not contain any statement in its body.
2. An infinite loop : while can be infinite if you forget to update expression inside the body of the loop.

EXAMPLE : EMPTY LOOP

```
....
....
long wait=0;
while (++wait<10000)
```

The above given is the **TIME DELAY LOOP.** It is useful for pausing the program for some time.

EXAMPLE : INFINITE LOOP

```
j=0
while(j<=n)
cout<<“\n”<<  j *  j ;
j++;
….
….
```

The above loop is an infinite loop as a only one statement taken into a loop’s body

EXAMPLE : FINITE LOOP

```
j=0
while(j<=n)
{
cout<<“\n”<<  j *  j ;
j++;
}
….
….
```

The above loop is an finite loop. It will terminate as soon as the value of j exceeds the n.

THE do-while LOOP – AN EXIT CONTROLLED LOOP

- Unlike the for and while the do-while loop is an exit controlled loop. i.e., it evaluates its test expression at the bottom of the loop after executing its loop –body statements.
- The syntax is,

```
        do
{
Statements
}while(test expression);   // here semicolon must
```

THE do-while LOOP – AN EXIT CONTROLLED LOOP

```
char ch='A';
do  {
        cout<<"\n"<<ch;
        ch++;
   } while (ch<='Z');
```

The above code prints the character from 'A' onwards util the condition ch<='Z' becomes false.

The most common use of the do-while loop is in menu selection routine, where the menu is flashed at once and depending upon the users response either it is repeated or terminated.

NESTED LOOPS

- A loop can contain another loop in its body. This form of a loop is called **nested loop.** In nested loop the inner loop must terminate before the outer loop.

```
for(i=1;i<=5;i++)
        {           cout<<"\n";
                    for(j=1;j<=i;j++)
                                cout<<"* ";
        }
```

The above prints following out put

```
*
* *
* * *
* * * *
* * * * *
```

COMPARISON OF LOOPS

- The for loop is appropriate when you know in advance how many times the loop will be executed.
- The other two loops while and do-while are more suitable in the situations where it is known before –hand when the loop will terminate. The while should be preferred when you may not want to execute the loop body even once (in case test condition is false), and the do-while loop should be preferred when you are sure you want to execute the loop body at least once.

JUMP STATEMENT - goto

- The goto statement is rarely used in the programming.
- A goto statement can transfer the program control anywhere in the program. The target destination of a goto statement is marked by the label. The syntax is,

goto label; //here you put **semicolon**

…….

…….

…….

label **:** **//**here you put **colon**

JUMP STATEMENT

- C++ has the four statements that perform an unconditional branch. They are,

    1. return
    2. goto
    3. break
    4. continue

  In addition to four statements C++ library function provides exit() that helps you break out of the program.

JUMP STATEMENT – goto Example

```
A=0;
start :
cout<<“\n”<<++a;
if(a<50) goto start;
```

NOTE:

Label may not immediately precede the closing right brace. If so then a null statement may be used.

For example

```
.......
{ goto last;
.....
.....
last:   // wrong!
}
```

NOTE:

For example

```
.......
{ goto last;
.....
.....
last:    ;  // null statement right!
}
```

NOTE:

- goto statement may not jump forward over the variable definition.

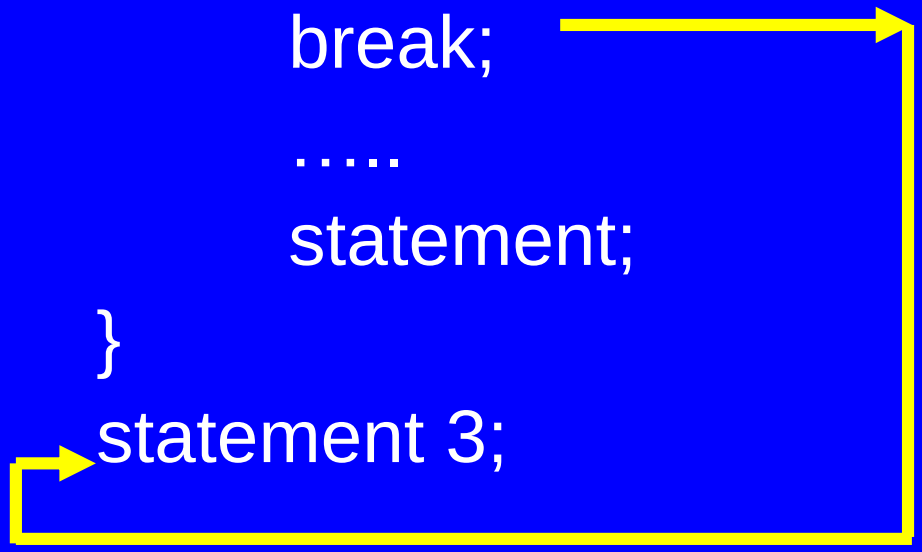
```
main()
{
goto last;  // Wrong! Jumping over the variable definition
char ch='a';
……
last:
}
```

break STATEMENT

The break statement enables the program to skip over the part of the code. A break statement terminates the smallest enclosing while,do-while for or switch statement,

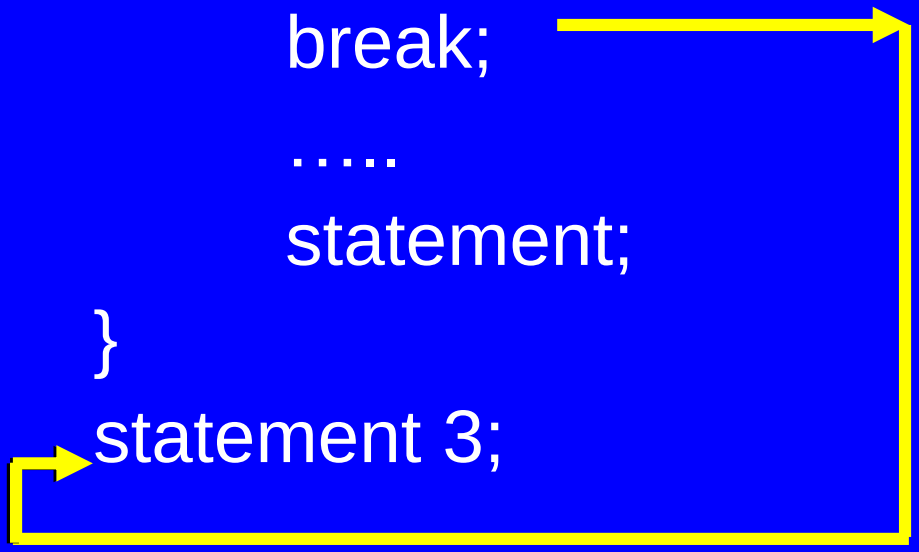
break STATEMENT - EXAMPLE

```
while (test expression)
{
        statement
        if(val>2000)
        break;
        …..
        statement;
}
statement 3;
```



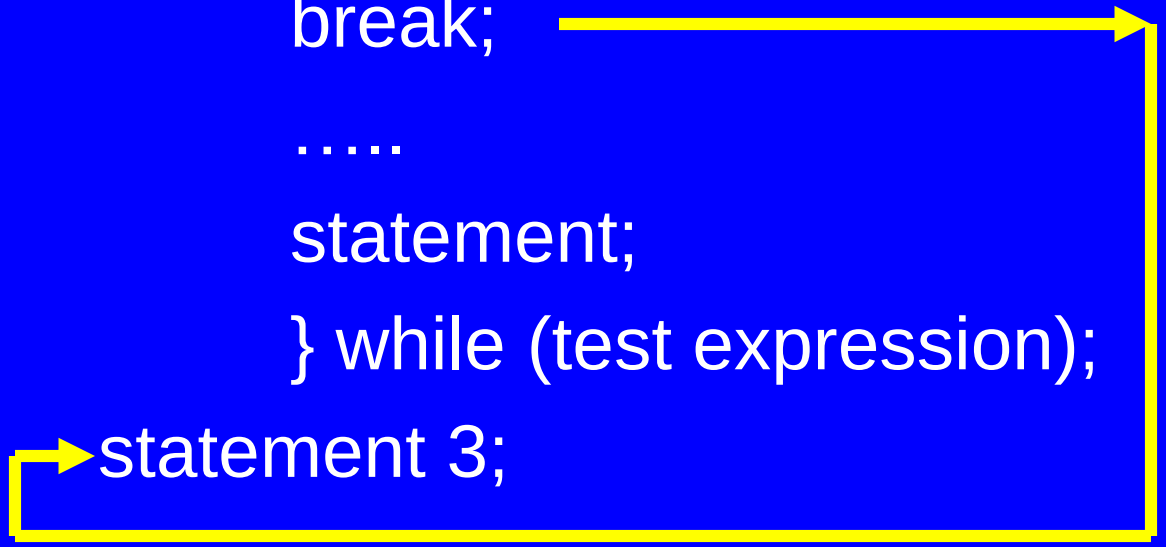
break STATEMENT - EXAMPLE

```
for( int;test expression;update expression)
   {
        statement
        if(val>2000)
        break;
        …..
        statement;
}
statement 3;
```



break STATEMENT - EXAMPLE

```
do {
    statement
    if(val>2000)
    break;
    …..
    statement;
    } while (test expression);
statement 3;
```

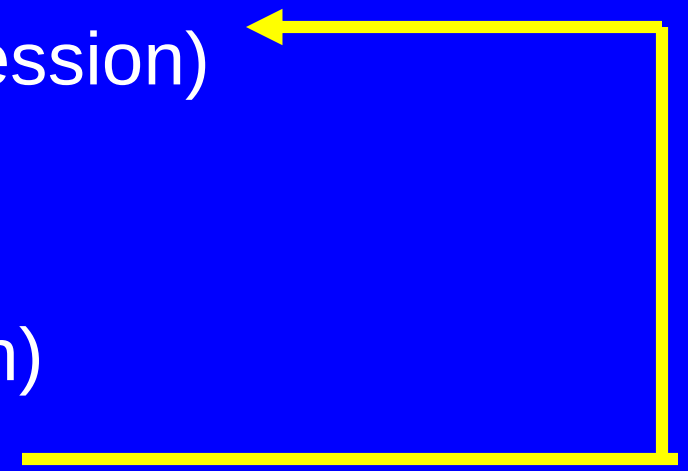


THE continue STATEMENT

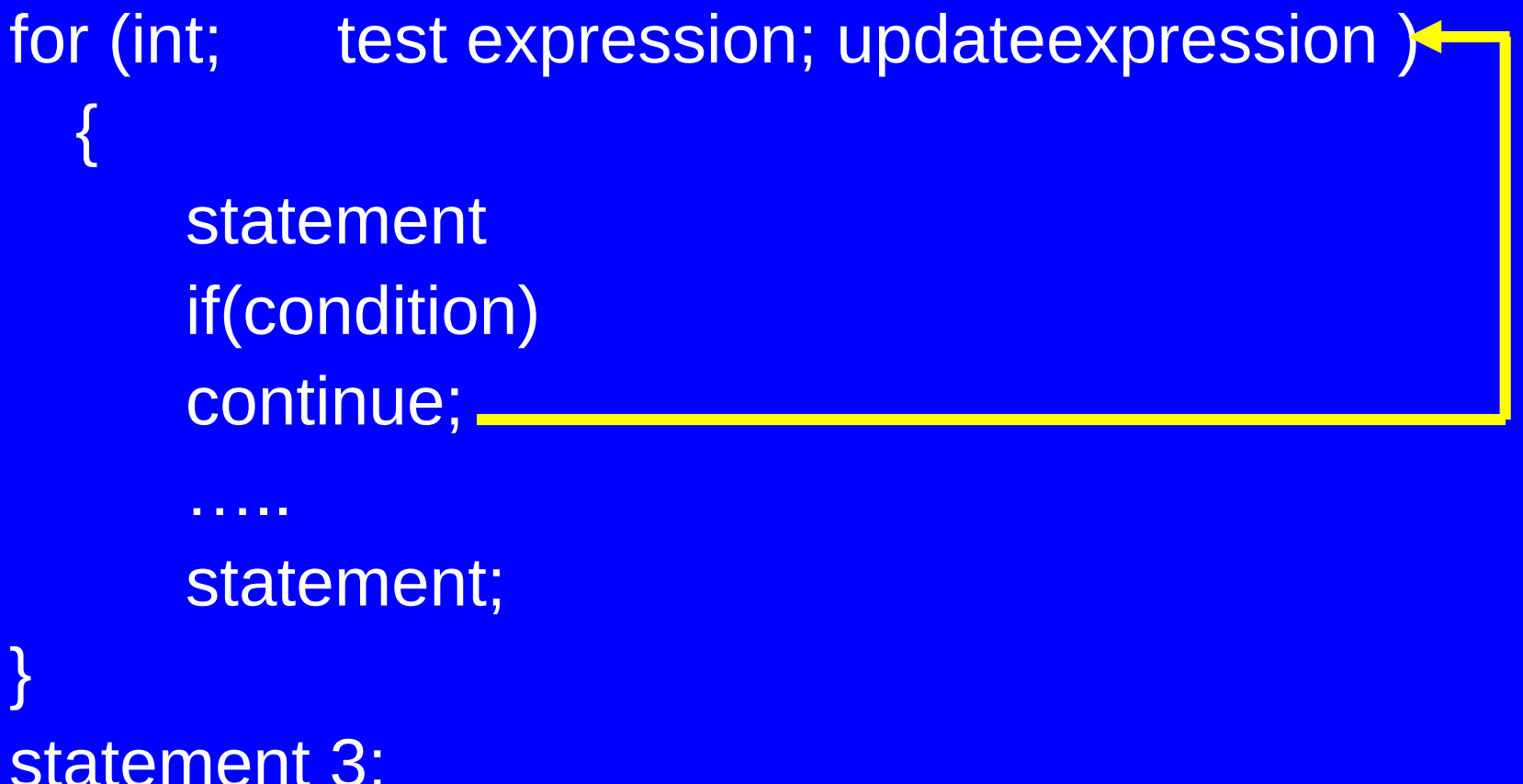
- The continue statement is the another jump statement like the break as the both the statements skips over the part of code but the continue statement is some what different than the break. Instead of forcing for termination it forces for the next iteration of the loop to take place.

THE continue STATEMENT EXAMPLE

```
while (test expression)
   {
        statement
        if(condition)
        continue;
        …..
        statement;
}
statement 3;
```

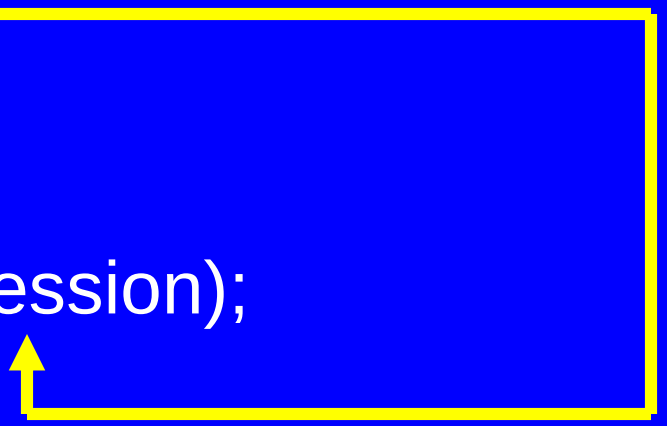


THE continue STATEMENT EXAMPLE



THE continue STATEMENT EXAMPLE

```
do
   {
        statement
        if(condition)
        continue;
        …..
        statement;
} while (test expression);
statement 3;
```



THE exit() FUNTION

- The exit() function causes the program to terminate as soon as it is encountered.
- The exit() function is a C++ standard library function defined in process.h file.

  which must be included in the program that uses exit() function

EXAMPLE-exit() FUNCTION

```
// Program to check entered number is prime number or not
#include<iostream.h>
#include<conio.h>
#include<process.h>
void main()
{
int num,i;
clrscr();
cout<<"Enter the Number: ";
cin>>num;
for(i=2; i<=num/2;++i)
{
cout<<"\n Not a Prime Number";
exit(0);
}
cout<<"\n It is Prime Number";
getch();
}
```

Enter the Number: 4

Not a Prime Number