**CONTROL STRUCTURES:**

They are three control structures in C

1. Sequence structure
2. Selection structure
3. Repetition structure
4. Unconditional structure

**SEQUENCE STRUCTURE**: This is default structure. Unless directed otherwise (means selection and repetition), the computer automatically executes C statements one after the other in the order in which they are written.

**SELECTION STRUCTURE**: C provides three types of selection structures.

1. if selection structure
2. if/else selection structure
   1. Conditional operator (ternary operator)
3. switch selection structure

**if selection structure:**

**if** selection structure either performs (selects) an action if a condition is true or skips the action if the condition is false. It is also called single-selection structure because it selects or ignores a single action.

**if(condition)**

**{**

**statement block;**

**}**

**statement-x;**

The statement block may be a single statement or a group of statements. If the condition is true, the statement block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x. Remember, when the condition is true both the statement-block and the statement-x executes in sequence.

**Note that in c a non-zero value is considered to true and, whereas zero is considered to be false**. We check the condition using relational and equivalent operators. The relations expressions(**x>y,x<=y,x==y**) always returns 1 if the condition is true and returns 0 if condition is false.

## Example:

main()

{ int x=5,y=10,z;

z=x<y; /\* Expression is true it returns 1 to z \*/ printf(“%d”, z);

z=x>y; /\*Expression is false it returns 0 to z \*/ printf(“%d”,z);

}

### output : 1 and 0

|  |  |  |
| --- | --- | --- |
| **Example:**  int x=5, y=10; if(x<y) |  | if(1) /\*non zero means true\*/ |
| {  executes these stmts because cond is true  } |  | {  } |

int x=5, y=10;

if(x>y) if(0)/\* 0 means false\*/

{

cannot executes these stmts 

because condition is false.

}

We can use arithmetic expressions in the if statement. For example all the following if statements are valid.

1. if(3+2)

printf(“This works”);

1. if(a=10)

printf(“This also works”);

1. if(-5)

printf(“ even this works”);

In the first if, the expression **3+2** is evaluate to 5 since 5 is a non-zero it is considered to be true.

Hence the printf executed.

In the second if, 10 gets assigned to a so the if is now reduced to if is now reduced to if(10). Since 10 is non-zero, it is true hence again printf() gets executed.

In the third if, -5 is a non-zero number, hence true. So again printf() goes to work. In place of -5 even if a float like 3.14 were used it is considered to be true. So the issue is not whether a number is integer or float , or whether it is positive or negative. Issue is whether it is **zero or non-zero**.

**Example:** #include<stdio.h>

void main()

{

int num1,num2;

printf(“Enter the numbers\n”); scanf(“%d %d”,&num1,&num2); if(num1==num2)

printf(“%d is equal to %d”,num1,num2); if(num1>num2)

printf(“%d is greater than %d”,num1,num2); if(num1<num2)

printf(“%d is less than %d”,num1,num2);

}

output : Enter the numbers: 3 4 3 is less than 4

## The if/else selection structure:

The if/else structure performs an action if the condition is true and performs a different action if condition is false. The if/else structure is called **double-selection/Bi-directional structure** because it selects between two different actions.

**if(condition)**

**{**

**True-block statement(s)**

**}**

**else**

**{**

**False-block statement(s)**

**}**

**statement-x;**

If the condition is true, then the true-block statements, immediately followed the if statement are executed; other wise; the false block statements are executed.

**% is an operator in which gives reminder where as / operator gives quotient**.

### e.g a=15 b=2 a%b=1 and a/b=7

**Example:**

/\* program to find whether a number is even or odd \*/ #include<stdio.h>

void main()

{

int a;

printf(“Enter the no\n”); scanf(“%d”,n); if(a%2==0)

printf(“The no %d is even”); else

printf(“The no %d is odd”);

}

OUTPUT :Enter the no 21 The no 21 is odd

**Nested if/else statement:** When a series of decisions involved, we may have to use more than one **if…else** statement. If we write an entire if-else construct within either the body of the **if** statement or the body of an else statement. This is called **nesting of if’**

**Department of Computer Science and Engineering**

**If(condition 1)**

**{**

**if(condition 2)**

**{**

**statement 1;**

**}**

**else**

**{**

**statement 2;**

**}**

**}**

**else**

**{**

**statement 3;**

**}**

If the condition1 is false, the statement 3 will be executed; otherwise it continues to perform the second test. If the condition 2 is true, the statement 1 is evaluated; otherwise the statement 2 well be evaluated and then the control is transferred outside.

**Example:**

/\*Program to find greatest of three no’s \*/

#include<stdio.h> void main()

{ int a,b,c;

printf(“Enter the values\n”); scanf(“%d %d %d”,&a,&b,&c);

if(a>b)

{

if(a>c)

printf(“%d is greater”,a); else

printf(“%d is greater”,c);

}

else

{

if(b>c)

printf(“ %d is greater”, b); else

printf(“ % d is greater”, c);

}

}

**Else-if Ladder:**

There is another way of putting **if**s together when multipath decisions are involved. A multipath decision is a chain of **if**s in which the statement associated with **else** is an **if**



**If(condition 1)**

**{**

**Statement 1;**

**}**

**Else if(condition 2)**

**{**

**Statement 2;**

**}**

**Else if(condition 3)**

**{**

**Statement 3;**

**}**

**else**

**default statement; statement-x;**

## Example:

/\*program to calculate roots of quadratic equations\*/ #include<stdio.h>

#include<math.h> main()

{ int a,b,c; float r1,r2,d;

printf("Enter the coefficients of equation\n"); scanf("%d %d %d",&a,&b,&c);

d=(b\*b)-(4\*a\*c); if(d<0)

{

printf("The roots are imaginary");

}

else if(d==0)

{

printf("Roots are equal\n"); r1=-b/(2\*a);

r2=r1;

printf("r1=%f r2=%f",r1,r2);

}

else if(d>0)

{

printf("Roots are real");

r1=(-b+sqrt(d))/(2\*a);

r2=(-b-sqrt(d))/(2\*a); printf("r1=%f r2=%f",r1,r2);

}

}

**Logical Operators:**

C provides logical operators that may be used to form more complex condition by combining simple conditions. These are used to combine multiple condition checking.

The logical operators are

**&& logical AND**

**|| logical OR**

**! logical NOT**

**Logical AND:**

Like the simple relational expressions(x>y), a logical expression also yields a value of zero or one.

**AND operator:**

**(cond1&&cond2)**

If both conditions are true the output of the logical AND expression is true(i.e it returns 1) and if any of the condition is false the output is false(i.e. it returns 0)

**Example:**

main()

{ char ch;

printf(“Enter any character\n); scanf(“%c”,&ch);

if(ch>=65 && ch<=90)

printf(“U entered capital letter\n”); else if(ch>=97 && ch<=122) printf(“U entered small alphabet\n”); else if(ch>=48 && ch<=57) printf(“U entered number\n”);

else

printf(“U entered special character\n”);

}

## OR Operator:

**(condt1 || condt2)**

If any of the condition is true the output of the expression is true and the output will be false when both the conditions are false

**Example:** #include<stdio.h> main()

{

char ch;

printf("Enter alphabet:\n"); scanf("%c",&ch);

if(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u') printf("You entered vowel");

else

printf("you entered consonant");

}

OUTPUT : Enter alphabet: a

you entered vowel.

**Not Operator:**

## (!conditon)

If the condition is true the output of the expression will be false and if the condition is false the output of the expression is true.

**Example:**

#include<stdio.h> main()

{ int a,b;

printf("enter two no"); scanf("%d %d",&a,&b);

if(!(a>b)) /\*Here if the condtion a>b is true the total !(a>b) becomes false\*/ printf("%d is greater",b);

else

printf("%d is greater",a);

}

**Conditional Operator or Ternary Operator(? :) :-**

The C language has an unusual operator, useful for making two-way decisions. This operator is a combination of **?** and **:** and takes three operands(**therefore called as ternary operator**). This operator is popularly known as the conditional operator. The general form of use of the conditional operator is as follows:

**conditional expression ? expression1:expression2**

The conditional expression is evaluated first. If the result is nonzero (true) **expression1** is evaluated and returned, otherwise **expression 2** is evaluated and its value is returned.

**Example:**

1) #include< stdio.h> 2)#include<stdio.h> main() main()

{ {

int a,b,c; int a,b,c,max;

printf(“Enter two no’s\n”); printf(“Enter three no’s\n”);

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

c = (a>b?a:b); max=(a>b?(a>c?a:c):(b>c?b:c));

printf(“%d is greater”,c); printf(“%d is greater”,max);

} }

OUTPUT: Enter two no’s OUTPUT: Enter three no’s 56 78 34 55 23

78 is greater 55 is greater.

**Switch Statement:**

The switch selection structure performs one of many different actions depending on the value of an expression. The switch structure is called **multiple-selection structure** because it selects among many different actions.

Switch (expression)

{

case const1:

block-1; break;

case const2:

block-2; break;

……….

………. default :

}

default block; break;

The expression is an integer expression or characters. const1,const2…. are constant expressions(evaluable to an integral constant) and are known as case labels. Each of these values should be unique within a switch statement. **block-1**, **block-2...** are statements lists and may contain zero or more statements. There is no need to put braces around these blocks. Note that case labels end with a **colon(:)**

When the switch is executed, the values of the expression is successively compared against the values const1, const2.... 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 the case values. If not present, no action take place if all matches fail and the control goes to the statement-x.

**Example:**

/\* Calculator program using switch case \*/ #include<stdio.h>

main()

{ int a,b,c,n;

printf(“Enter two no’s\n”); scanf(“%d %d”, &a,&b);

printf(“1.Addition\n2.Substration\n3.Multiplication\n4.Division scanf(“%d”,n)

switch(n)

{

case 1 :

c = a + b; break; case 2 :

c = a – b; break; case 3 :

c = a \* b; break; case 4 :

c = a / b; break; default :

printf(“Invalid Number\n”);

}

printf(“%d”,c);

}

# Increment and Decrement Operators:

C has two very useful operators called increment and decrement operators.

The operator ++ add 1 and operator -- subtracts 1. Both are unary operators (because we are operating on one operand). They can be used in two ways 1) **Prefix** 2**) Postfix**

Pre increment - ++m m will have increment value by 1.

Post increment - m++ m will have m value and then increments m by 1; pre decrement and post decrement also same as above.

we use increment and decrement statements in loops structures extensively.

When m++ and ++m mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement. Consider the following programs:

|  |  |  |
| --- | --- | --- |
| 1) #include<stdio.h>  main() | 2) | #include<stdio.h>  main() |
| { |  | { |
| int i=5; |  | int i=5,j; |
| ++i; |  | j=++i; |
| printf(“%d\t”,i); |  | printf(“%d %d \n”,i,j); |
| i=5; |  | i=5; |
| i++; |  | j=i++; |
| printf(“%d”,i); |  | printf(“%d %d”,i,j); |
| } |  | } |

Output: 6 6 Output: 6 6

6 5

If we observe the second program for preincrement of **i** first **i** is incrementing and afterwards it is assigning to j and in the post increment case first the value of **i** is assigned to **j**(i.e 5) and afterwards the value in the i is incrementing. Therefore we get the output 6 6 and 6 5.

**Repetition Structure:**

C provides three repetition structures

**1)** while loop 2) do-while loop 3) for loop

A loop is a group of instructions the computer executes repeatedly as long as loop-continuation condition remains true**.**

**WHILE LOOP:**

The basic format of while loop is

**While(condition)**

**{**

**body of loop**

**}**

The while is entry-controlled loop statement. The test-condition is evaluated first 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, the body is executed once again. This process of repeated execution of the body continues until the test-condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statements. The braces needed only if the body contains one or more statements.

|  |  |
| --- | --- |
| #include <stdio.h>  int main()  {  int n, n1, rem, num=0;  printf("Enter a positive integer: ");  scanf("%d", &n);  n1=n;  while(n1!=0)  {  rem=n1%10;  num+=rem\*rem\*rem;  n1/=10;  }  if(num==n)  printf("%d is an amstrong number.",n);  else  printf("%d is not an Amstrong number.",n);  }  Input: Enter a positive integer: 371  Output: 371 is an Armstrong number. | #include<stdio.h>  main()  {  int n,flag=0,i=2;  printf("Enter the number\n"); scanf("%d",&n);  while(i<n)  {  if(n%i==0)  flag=1;  i++;  }  if(flag==1)  printf("The no is not a prime\n"); else  printf("The no is prime");  }  OUTPUT: 7 The no is prime. |

**DO-WHILE:**

The general format of do while is

**do**

**{**

**body of loop**

**}**

**while(condition);**

The while loop constructs that we have discussed in the previous section makes a test of condition before the loop is executed. Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. On some occasions it might be necessary to execute the body of the loop before the test is performed. Such situations can be handled with the help of the **do** statement.

On reaching the do statement, the program proceeds to evaluates the body of the loop first. At the end of the loop, the test-condition in the while is evaluated. If the condition is true, 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 become false, the loop will be terminated and the control goes to the statement that appears immediately after the while condition.

**Example:**

#include<stdio.h>

main()

{

int n,num,r,ans=0;

printf("Enter a positive integer: ");

scanf("%d", &num);

n=num;

do

{

r=n%10;

ans=ans+r\*r\*r;

n=n/10;

}while(n>0);

if(ans==num)

{

printf("%d is an Armstrong number.",num);

}

else

{

printf("%d is not an Armstrong number.",num);

}

}

**FOR Statement:**

The for loop is another entry-controlled loop that provides a more concise loop control structure.

**for(initialization; test-condition; increment/decrement)**

The execution of the for statement is as follows:

1. Initialization of the control variables is done first, using assignment statements such as i=1 and count=0. The variable i and count are known as loop-control variables.
2. The value of the control variable is tested using the test-condition. The test-condition is a relational expression, such as i<10 that determines when the loop will exit. If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately followed 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. Now, the control variable is incremented/decrement using an assignment statement such as i=i+1/i=i-1 and the new value of the control variable is again tested to see whether it satisfied the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues still the value of the control variable fails to satisfy the test-condition.

## Example:

### /\* program to generate the table of a given no \*/

#include<stdio.h>

main()

{

int n,i,a;

printf("enter the no\n");

scanf("%d",&n);

for(i=1;i<=10;i++)

{

a=n\*i;

printf("\n %d \* %d = %d",n,i,a);

}

}

**BREAK :**

When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the **break** would only exit from the loop containing it. That is, break will exit only a single loop.

**While( --- )**

**{**

**-------------**

**-------------**

**if(condition) break;**

**------------**

**------------**

**}**

**For( )**

**{**

**-------------**

**for( )**

**{**

**if(condition) break;**

**}**

**------------**

**------------**

**}**

**Example:**

#include<stdio.h>

main()

{

int i;

for(i=1;i<=10;i++)

{

if(i==5)

break;

printf("%d\t",i);

}

printf("\n Breaks the iterations and comes out of the loop at x == %d\n",i);

}

**OUTPUT:** 1 2 3 4

Breaks the iterations and comes out of the loop at x == 5

**CONTINUE:**

When the continue statement is encountered in the loop, as the name implies, it 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 THE NEXT**

**ITERATION**”. The format of the continue statement is

# continue;

In **while** and **do** loops, continue causes the control to go directly to the test-condition and then to continue the iteration process. In the case of **for** loop, the increment section of the loop is executed before the test-condition is evaluated.



**While(condition)**

**{**

**---**

**if(condition) continue~~;~~**

**-----**

**}**

**For(init;test;incre)**

**{**

**-----**

**if(condition) continue;**

**-----**

**}**

**Example:** #include<stdio.h>

main()

{

int i;

for(i=1;i<=10;i++)

{

if(i==5)

continue;

printf("%d\t",i);

}

printf("\n continue skip the iteration I == 5 and continue the loop up to 10\n");

}

OUTPUT: 1 2 3 4 6 7 8 9 10

Continue skip the iteration I == 5 and continue the loop up to 10