**CHAPTER – 3**

**Selection**

**Operator:**

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate the data and variables. C language is rich in built-in operators and classified into a number of categories. They include:

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Assignment Operators
* Increment and decrement operators
* Conditional operators
* Bitwise Operators
* Special Operators

An expression is a sequence of operands and operators that reduces to a single value.

Ex: 10+15

is an expression whose value is 25. The value can be any type other than void.

**Arithmetic Operators:**

|  |  |
| --- | --- |
| Operator | Meaning |
| + | Addition or unary plus |
| - | Subtraction or unary minus |
| \* | Multiplication |
| / | Division |
| % | Modulo division |

C provides all the basic arithmetic operators. They are

The operators +,-,\* and / all work the same way as they do in other languages. These can operate on any built-in data type allowed in C. The unary minus operator, in effect, multiplies its single operand by -1. Therefore, a number preceded by a minus sign changes its sign.

Integer Arithmetic:

When both the operands in a single arithmetic expression such as a+b are integers, the expression is called an integer expression, and the operation is called integer arithmetic. This mode of expression always yields an integer value. The largest integer value depends on the machine, as pointed out earlier

Example:

If a and b are integers then for a=14 and b=4

We have the following results: a – b=10

a + b = 18

a \* b = 56

a / b = 3

a% b = 2

During integer division, if both the operands are of the same sign, the result is truncated towards zero. If one of them is negative, the direction of truncation is implementation dependent. That is,

6/7=0 and -6/-7=0

but -6/7 may be zero or -1 (Machine dependent)

Similarly, during modulo division, the sign of the result is always the sign of the first operand(the dividend)

That is -14 % 3 =-2

-14 % -3= -2

14 % -3=2

Real Arithmetic: An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation. Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result.

If x, y and z are floats, then we will have: x = 6.0 / 7.0 = 0.857143

y = 1.0 / 3.0 = 0.333333

z = - 2.0 / 3.0 = - 0.666667

The operator % cannot be used with real operands.

Mixed-mode Arithmetic: when one of the operands is real and the other is integer, the expression is cal. led a mixed-mode arithmetic expression. If either operand is of the real type, then only the real operation is performed and the result is always a real number. Thus 15/10.0 = 1.5 whereas 15/10=1

**Relational Operators:**

If we want to compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help relational operators.An expression such as

a<b or 1<20

containing a relational operator is termed as relational expression. The value of relational expression is termed as either one or zero. It is one if the specified relation is true and zero if the condition is false.

Ex: 10<20 is true

but 20<10 is false

C supports 6 relational operators in all. They are:

Operator Meaning

< is less than

> is greater than

<= is less than or equal to

>= is greater than or equal to

= = is equal to

!= is not equal to

A simple relational expression contains only one relational operator and has the following form:

ae-1 relational operator ae-2

ae-1 and ae-2 are arithmetic expressions, which may be simple constants, variables or combination of them. Given below are some examples of simple relational expressions and their values:

4.5 <= 10 TRUE

4.5 < 10 FALSE

-35 >= 0 FALSE

10 < 7+5 TRUE

a+b == c+d TRUE only if the sum of values of a and b is equal to the sum c & d.

When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared. That is, arithmetic operators have a higher priority over relational operators. Relational expressions are used in decision statements such as, if and while to decide the course of action of a running program.

**Logical Operator:**

In addition to the relational operators . C has the following three logical operators.

&& logical AND

|| logical OR

! logical NOT

The logical operators && and || are used when we want to test more than one condition and make decisions.

Ex: a>b && x == 10

Theabove expression is termed as a logical expression or a compound relational expression which combines two or more relational expressions. Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table shown below. The logical expression given above is true only if a>b is true and x==10 is true. If either (or both) of them are false, the expression is false.

**Truth Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Op-1 | op-2 | Value of the expression | |
| Op-1 && op-2 | op-1 || op-2 |
| Non-zero | Non-zero | 1 | 1 |
| Non-zero | 0 | 0 | 1 |
| 0 | Non-zero | 0 | 1 |
| 0 | 0 | 0 | 0 |

Ex: 1. If(age>55 && salary <1000)

2. If (number<0 || number>100)

**Precedence of relational operators and logical operators:**

Ex: (a>b \*5) &&(x<y+6)

In the above example, the expressions within the parentheses are evaluated first. The arithmetic operations are carried out before the relational operations. Thus b\*5 is calculated and after that a is compared with it. Similarly y+6 is evaluated first and then x is compared with it.

**Assignment Operator:**

Assignment operators are used to assign the result of an expression to a variable. We have seen the usual assignment operator, ‘=’. In addition, C has a set of ‘shorthand’ assignment operators of the form.

v op = exp;

Where v is a variable, exp is an expression and op is a C binary arithmetic operator. The operator op= is known as the shorthand assignment operator.

The assignment statement v op = exp;

Is equivalent to v = v op (exp); with v evaluated only once.

Ex: x += y+1;

This is same as x = x + (y+1);

The shorthand operator += means ‘add y+1 to x’ or ‘increment x by y+1’.

For y = 2, the above statement becomes x += 3;

and when this statement is executed, 3 is added to x. If the old value of x is, say 5, then the new value of x is 8. Some of shorthand assignment operators are:

Statement with simple Statement with Assignment operator shorthand operator

a = a + 1 a + = 1

a = a - 1 a - = 1

a = a \*(n +1) a \* = (n+1)

a = a / (n+ 1) a / = (n + 1)

a = a %b a % = b

The use of shorthand assignment operators has three advantages:

1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
2. The statement is more concise and easier to read.
3. The statement is more efficient.

#include<stdio.h>

#define A 2

int main()

{

int a;

a=A;

printf("%d \n",a);

a\*=a;

printf ("%d \n",a);

return0;

}

Output: 2

4

**Increment and Decrement operators:-**

The increment operator ++ and decrement operator -- are unary operators with the same precedence as the unary -, and they all associate from right to left. Both ++ and -- can be applied to variables, but no to constants or expressions. They can occur in either prefix or postfix position, with possibly different effects occurring. These are usually used with integer data type.

syn: ++variable or --variable or variable++ or variable—

The operator ++ adds 1 to the operand, while – subtracts 1.

Ex: ++m; or m++;

--m; or m--;

++m; is equivalent to m = m+1; (or m+=1;)

--m; is equivalent to m = m-1; (or m-=1;)

We use the increment and decrement statements in for and while extensively.

Ex: m=5;

y=++m;

In this case, the value of y and m would be 6. Suppose, if we rewrite the above statements as m=5;

y=m++;

then the value of y would be 5 and m would 6. A prefix operator first adds to 1 to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

Similar is the case, when we use ++(or--) in the subscripted variables. That is, the statement a[i++]=10;

is equivalent to a[i]=10;

i=i+1;

The increment and decrement operators can be used in complex statements.

Ex: m=n++ -j+10;

Old value of n is used in evaluating the expression. n is incremented after the evaluation.

**Conditional operator:**

An operator called ternary operator pair “?:” is available in C to construct conditional expressions of the form.

exp1? exp2: exp3;

where exp1,exp2, and exp3 are expressions.

The operator ?; works as follows: exp1 is evaluated first. If it is nonzero(true), then the expression exp2 is evaluated and becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expressions(either exp2 or exp3) is evaluated.

Ex: x=3; y=15;

z=(x>y)?x:y;

In this example, z will be assigned the value of y. This can be achieved using the if..else statements as follows:

If (x>y)

z=x;

else

z=b;

Ex: main ()

{

int a,b;

clrscr ();

printf (“Enter the values of a and b”);

scanf (“%d%d”,&a,&b);

(a>b)?printf(“%d is bigger”,a):printf (“%d is bigger”,b);

Getch ();

}

Ex: main ()

{

int a,b,c,d;

clrscr ();

a=15;b=10;

c=++a – b;

printf (“The values of a, b, c are %d , %d , %d”,a,b,c);

d=b++ +a;

printf (“The values of a, b, d are %d , %d , %d”,a,b,d);

printf (“a / b = %d”,a/b);

printf (“a %% b = %d”,a%b);

printf (“a \* = b = %d”,a\*=b);

printf (“ %d”,(c>d)?1:0);

printf (“ %d”,(c<d)?1:0);

}

Output: The values of a, b, c are 16, 10, 6

The values of a, b, d are 16, 11, 26

a/b = 1 a%b = 5 a \*= b = 176 0 1

**Bitwise operators**:

All data items are stored in the computer’s memory as sequence of bits (i.e. 0s and 1s) and some applications need the manipulation of bits. C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double.

**Operator Meaning**

& bitwise AND

| bitwise OR

^ bitwise exclusive OR

<< shift left

>> shift right

~ One’s Complement

The result of bitwise AND operator is 1 when both the bits are 1, otherwise it is 0. Similarly the result of bitwise OR is one if any one of the bit is 1, otherwise it is 0. The result of bitwise XOR is 1 when the bits are different. The bitwise complement operator reverses the state of each bit. i.e. if the bit is zero, complement of it will 1 and vice versa.

Ex: int a =5, b=3;

Then equivalent binary representation of a = 5 = 0000 0101 and b=3= 0000 0011

Therefore

a = 0000 0101

b = 0000 0011

a& b = 0000 0001 = 1

a | b = 0000 0111 = 7

a ^ b = 0000 0110 = 6

~a = 1111 1010

a << 1 = 0000 1010 = 10 equivalent to multiplying a by 2

a >> 1 = 0000 0010 = 2 equivalent to dividing a by 2

**Special Operators:**

**1) Comma Operator**

C has some special operators. The comma operator is one among them. This operator can be used to link the related expressions together. A comma-linked list of expressions is evaluated left to right and the value of right-most expression is the value of the combined expression.

For example, the statement

Value=(a=2, b=6 ,a+b);

First assigns the value 2 to a, then assigns 6 to b, and finally assigns 8(i.e. 2+6) to value. The comma operator has the lowest precedence of all operators, hence the parentheses are necessary.

Ex: In for loops:

for(a=1, b=10;a<=b; a++, b++)

In while loops:

while(c=getchar(), C!=’10’)

Exchanging values:

t=x, x=y, y=t;

**2) Sizeof operator**

The sizeof operator returns the memory size (in bytes) of the operand. The operand may be a constant, variable or a data type. It is normally used to determine the size of arrays and structures.

syn: sizeof(operand);

Ex: x = sizeof(int); will return 2

y = sizeof(10.2) will return 4 since a float point constant requires 4 bytes of memory

z = sizeof(x) will return 2

Ex: main()

{

int x;

double y;

char ch=’y’;

clrscr();

printf(“Size of x = %d\n”, sizeof(x));

printf(“Size of y = %d\n”, sizeof(double));

printf(“Size of char = %d\n”, sizeof(ch));

getch();

}

Output: size of x = 2

size of y = 8

size of char = 1

**Selection:**

C language possesses different decision-making capabilities by supporting the following statements:

1. If statement

2. Switch statement

3. Conditional operator statement

These statements are popularly known as decision-making statements. Since these statements ‘control’ the flow of execution, they are also known as control statements.

**Decision making with if statement:**

The if statement is a powerful decision-making statement and is used to control the flow of execution of statements. It is basically a two-way decision statement and is used in conjunction with an expression. It takes the following form: Entry

test exp

?

**If (test expression)**

It allows the computer to evaluate the expression first

and then, depending on whether the value of the expression False (relation or condition) is ‘true’ (or non-zero) or ‘false’ (zero), it transfers the control to a particular statement. True

The if statement may be implemented in different forms depending on the complexity of additions to be tested. The different forms are:

1. Simple if statement

2. if.....else statement

3. Nested if...else statement

4. else if ladder

**Simple if statement:**

The general form of simple if statement is Entry

test exp

?

Statement-block

Statement-x

Next Statement

if (test expression)

{ true

Statement-block;

}

Statement-x; false

The ‘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

execution will jump to the statement-x. When the condition is true both the statement-block and the statement-x are executed in sequence.

Ex: # include <stdio.h>

# include <conio.h>

void main ()

{

int bonus,cy,yoj,service;

printf ("Enter current year and year of joining");

scanf ("%d%d",&cy,&yoj);

service=cy-yoj;

if (service>3)

{

bonus=2500;

printf ("Bonus = Rs.%d",bonus);

}

getch();

}

Output: Enter current year and year of joining 2014 2008

Bonus = Rs.2500

Output: Enter current year and year of joining 2014 2012

**The if....else statement:**

The if..else statement is an extension of the simple if statement. The general form is

Syn: if (test expreesion) Entry

{

test exp

?

True-block statement(s) True False

True- block stat

Statement-x SssdfsdfsfsfSStatememStatement-x

}

else

False- block stat

{

False-block statement(s)

}

Next statment

Statement-x

If the test expreesion is true, then the true-block statement(s), immediately following the if statements are executed; Otherwise, the false-block statement(s) are executed. In either case, either true-block or false-block will be executed, not both. In the above diagram, the control is transferred subsequently to the statement-x.

Ex: # include <stdio.h>

# include <conio.h>

Void main ()

{

int a;

printf (“Enter any number”);

scanf (“%d”,&a);

if (a>0)

printf (“ %d is positive number”);

else

printf (%d is negative number”);

}

Output: Enter any number -1

-1 is negative number

Enter any number 4

4 is positive number

**Nesting of if....else statements:**

When a series of decisions are involved, we may have to use more than one if.....else statement in nested form as shown below:

entry

Statement-x

Next Statement

Statement-1

Statement-2

Statement-3

Test cond 1 ?

Test cond 2 ?

false true

false true

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

If (test condition-1)

{

If (test condition-2)

{

Statement-1;

}

Else

{

Statement-2;

}

}

Else

{

Statement-3;

}

Statement-x;

Ex: # include <stdio.h>

# include <conio.h>

Void main ()

{

int a,b,c;

printf (“Enter any 3 numbers”);

scanf (“%d%d%d”,&a,&b,&c);

if (a>b)

{

If (a>c)

Printf (“ a ia largest”);

Else

Printf (“c is largest”);

}

Else if (b>a)

{

If (b>c)

Printf (“b is largest”);

Else

Printf (“c is largest”);

}

Else

{

Printf (“Two of the three numbers are equal”);

}

Getch ();

}

Output: Enter any 3 numbers 1 2 4

C is largest

Enter any 3 numbers 4 6 1

b is largest

Enter any 3 numbers 5 2 4

C is largest

Enter any 3 numbers 1 2 2

C is largest

Enter any 3 numbers 2 2 1

Two of the three numbers are equal

**The Else if Ladder**:

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

Syn: if (condition 1)

statement-1;

else if (condition 2)

statement-2;

else if (condition 3)

statement-3;

else if (condition n)

statement-n;

else

default-statement;

statement-x;

This construct is known as the else if ladder. The conditions are evaluated from the top (of the ladder), downwards. As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x (skipping the rest of the ladder). When all the n conditions become false, then the final else containing the default-statement will be executed. The following figure shows the logic of execution of else if ladder statements.

Entry

Statement-2

Next statement

Statement-3

Cond-n

Statement-n

Default Statement

Statement-1

Cond-2

Cond-3

Cond-1

Statement-x

False True

False True

False True

True

False

Ex: # include <stdio.h>

# include <conio.h>

void main ( )

{

int num;

printf (“Enter any number”);

scanf (“%d”,&num);

if ( num > 0 )

printf ("\n Number is Positive");

else if ( num < 0 )

printf ("\n Number is Negative");

else

printf ("\n Number is Zero");

}

Output: Enter any number 6

Number is positive

Enter any number -5

Number is negative

Enter any number 0

Number is Zero

**The Switch Statement:**

We can use an if statement to control the selection. Then, the complexity of such a program increases dramatically when the number of alternatives increases. So, C has a built-in multiway decision statement known as a switch. The switch statement tests the value of a given variable (or expression) aganist a list of case values and when a match is found, a block of statements associated with that case is executed.

Syn: switch (expression)

{

case value-1:

block-1;

break;

case value-2:

block-2;

break;

......................

.......................

default:

default-block;

break;

}

Statement-x;

The expression is an integer expression or characters. Value-1, value-2,...... are constants or constant expressions and are known as case labels. Each of these values should be unique with in a switch statement. block-1,block-2,....... are statement 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 successfully compared against the valuesvalue-1, value-2,........ 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 takes place if all matches fail and the control goes to the statement-x.

The selection process of switch statement is represented in the following flow chart.

Next statement

Block - 1

Block-2

Default Block

Switch exp

Ex : # include <stdio.h>

# include <conio.h>

main ()

{

char ch;

int a,b;

clrscr ();

printf ("\nEnter the values for a and b");

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

printf ("Enter any option");

flushall();

scanf (" %c",&ch);

switch (ch)

{

case '+':printf ("%d + %d = %d",a,b,a+b);break;

case '-':printf ("%d - %d = %d",a,b,a-b);break;

case '\*':printf ("%d \* %d = %d",a,b,a\*b);break;

case '/':printf ("%d / %d = %d",a,b,a/b);break;

case '%':printf ("%d %% %d = %d",a,b,a%b);break;

default:printf ("Invalid option");

}

getch ();

}

Output: Enter the value of a & b : 2 3

Enter any option: +

2 + 3 = 5

**The ? :Operator:**

C language has an unusual operator, useful for making two-way decisions. This operator is a combination of ? and :, and takes three operands. This operator is popularly known as the conditional operator.

Syn: conditional expression ? expression1 : expression2

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

Ex: if (x>0)

Flag=0;

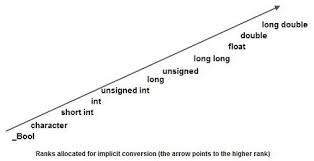
Else

Flag=1;

Can be written as flag= (x<0)?0:1;

**Type Conversion in C or Type Casting:**

Type casting refers to changing an variable of one data type into another. The compiler will automatically change one type of data into another if it makes sense. For instance, if you assign an integer value to a floating-point variable, the compiler will convert the int to a float. Casting allows you to make this type conversion explicit, or to force it when it wouldn’t normally happen. There are two types of type conversion:

1. **Implicit Type Conversion:**

When the type conversion is performed automatically by the compiler without programmers intervention, such type of conversion is known as implicit type conversion or type promotion.

Ex: int x;

for(x=97; x<=122; x++)

{

printf("%c", x); /\*Implicit casting from int to char thanks to %c\*/

}

1. **Explicit Type Conversion:**  
    The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion. The explicit type conversion is also known as type casting. Type casting in c is done in the following form:

Syn: (data\_type) expression;

where, data\_type is any valid c data type, and expression may be constant, variable or expression.

Ex: int x;

for(x=97; x<=122; x++)

{

printf("%c", (char)x); /\*Explicit casting from int to char\*/

}