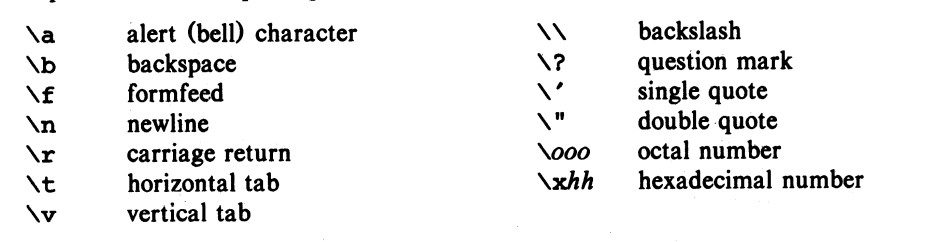
***Techniques for Programming in C***

|  |  |  |
| --- | --- | --- |
| Category | Operator | Associativity |
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)\* & sizeof | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Left to right |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %=>>= <<= &= ^= |= | Right to left |
| Comma | , | Left to right |

******

1. **Interchanging 2 Variables**

temp = x;

    x = y;

  y = temp;

// OTHER WAY Without using third variable;

x=x+y;

y=x-y;

x=x-y;

1. **Checking Prime Number or Not**

if((a ==2 || a == 3 || a == 5 || a== 7) || a%2!=0 && a%3!=0 && a%5!=0 && a%7!=0 && a!=1){

            printf("%d is a Prime Number \n", a);

    }

    else{

        printf("%d is not a Prime Number", a);

    }

**OR**

int a;

    printf("Enter the number if you want to know whether it is prime or not? : ");

    scanf("%d", &a);

    int div = 0;

    for(int i=1; i<=a; i++){

        if(a%i==0){

            div++;

        }

    }

    if(div==2){

        printf("%d is prime", a);

    }

    else{

        printf("%d is not prime", a);

    }

1. **Divisors of an integer**

for(int i=1; i<=n; i++){

        if(n%i==0){

            printf("%d \n", i);

        }

  }

1. **Counting Digits**

#include<stdio.h>

int main(){

    int n, count=0;

    printf("Enter Integer: ");

    scanf("%d", &n);

    while(n!=0){

        n/=10;

        count++;

    }

    printf("Count is %d", count);

  return 0;

}

1. **Condition for Leap Year (V.IMP)**

year%400==0 || (year%4==0 && year%100!=0)

1. **Reversing the DIGITS**

while(n!=0){

        remainder = n%10;

        rev = rev\*10 + remainder;

        n /= 10;

    }

1. **Sum of Digits**

while(n!=0){

        remainder = n%10;

        sum  = sum + remainder;

        n = n/10;

    }

1. **Getting first digit**

while(n>=10){

        n = n/10;

    }

    firstnum = n;

1. **Roots of Quadratic Equation**

float d = b\*b - 4\*a\*c;

if(d<0){

        printf("Roots are imaginary... \n");

        float RP = -b/(2\*a);

        float IMG = sqrt(-d)/(2\*a);

        printf("The roots are  %.2f+%.2fi and %.2f-%.2fi", RP, IMG, RP, IMG);

    }

    else if(d==0){

        printf("Roots are equal... \n");

        printf("Roots --> %.2f", -b/(2\*a));

    }

    else if(d>0){

        printf("Roots are distinct");

        float R\_1 = (-b+sqrt(d))/(2\*a);

        float R\_2 = (-b-sqrt(d))/(2\*a);

        printf("Root-1 = %.2f and Root-2 = %.2f", R\_1, R\_2);

    }

1. **Alphabet or Special Char**

char c;

    printf("Enter: ");

    scanf("%c", &c);

    if((c>=32 && c<=64) || (c>=91 && c<=96) || (c>=123 && c<=127)){

        printf("It is special Character");

    }

    else if((c>=65 && c<=90) || (c>=97 && c<=122)){

        printf("It is an alphabet");

    }

1. **Vowel or Consonant**

char c;

    printf("Enter any character to check whether entered character is a vowel or consonant --> ");

    scanf("%c", &c);

    if(c==65|| c==69 || c== 73 || c==79 || c==85 || c==97 || c==101 || c==105 || c==111 || c==117 ){

        printf("Entered character is a vowel");

    }

    else{

        printf("Entered character is a consonant");

    }

1. **Ascending Order**

for(int i=0; i<n; i++){             // n is the number of elements

        for(int j = i+1; j<n; j++){

            if(arr[j]<arr[i]){

                temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }

        }

  }

1. **Descending Order**

for(int i=0; i<n; i++){             // n is the number of elements

        for(int j = i+1; j<n; j++){

            if(arr[j]<arr[i]){

                temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }

        }

  }

1. **HCF (Highest Common Factor)**

#include<stdio.h>

int main(){

    int a, b, hcf;

    printf("Enter the value of a and b: ");

    scanf("%d %d", &a, &b);

    if(a<b){

        // min num --> a

        for(int i=1; i<=a; i++){

            if(a%i==0 && b%i==0){

                hcf = i;

            }

        }

    }

    else if(a>b){

        // min num --> b

        for(int i=1; i<=b; i++){

            if(a%i==0 && b%i==0){

                hcf = i;

            }

        }

    }

    printf("The HCF is %d", hcf);

1. **LCM (Lowest Common Multiple)**

For this simply use the HCF program and use the formula that a x b = HCF x LCM

1. **Smallest Element and its position**

int arr[100];

    int position = 0;

    int n;

    printf("Enter how many elements u gonna put in the array: ");

    scanf("%d", &n);

    for(int i=0; i<n; i++){

        printf("Enter element %d --> ", i+1);

        scanf("%d", &arr[i]);

    }

    int small = arr[0];

    for(int i=0; i<n; i++){

        if(arr[i]<small){

            small = arr[i];

            position = i+1;

        }

    }

    printf("The smallest element entered is %d \n", small);

    printf("Position of that element is %d", position);

1. **Getting 2nd last digit**

 int n, num;

    printf("Enter the value of n: ");

    scanf("%d", &n);

    while(n>=10){

        n = n/10;

    }

    num = n;

    printf("firstnum is %d \n", num);

1. **Printing nth digit of number**

#include<stdio.h>

#include<math.h>

int main(){

    int n, rem;

    printf("Enter the value of n: ");

    scanf("%d", &n);

    int k;

    printf("Which digit u wanna know: ");

    scanf("%d", &k);

    while(n>=pow(10, (k-1))){

        rem = n%10;

        n = n/10;

    }

    if(k==1){

        printf("%dst digit is %d", k, rem);

    }

    else if(k==2){

        printf("%dnd digit is %d", k, rem);

    }

    else if(k==3){

        printf("%drd digit is %d", k, rem);

    }

    else{

        printf("%dth digit is %d", k, rem);

    }

    return 0;

}

1. **Printing Value in sin(x), cos(x) or other trigon**

float x;

    printf("Enter Value of x: ");

    scanf("%f", &x);

    if(x==0)

     printf("Value of sin(1/x) is Not Possible");

    else{

      printf("Value of sin(1/x): %.4f", sin((1/x)\*3.14159/180)); // Always use \*3.14159/180 to convert radian to degree

    }

1. **Taking Modulus of a Number**

#include<stdio.h>

#include<stdlib.h>

int main(){

    int a;

    printf("Enter a number: ");

    scanf("%d", &a);

    printf("The value of 'a' is %d", abs(a));

    return 0;

}

1. **Printing Random Number**

#include<stdio.h>

#include<stdlib.h>

**int** main(){

**printf**(" %d ", **rand**());

**return** 0;

}

1. **Printing Random Number after Every Run!**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

**int** main(){

**srand**(**time**(0));

**printf**(" %d ", **rand**());

**return** 0;

}

1. **Printing Random Number in a range**

int lower = 1, upper = 10, count = 1;   //  You can also take limit using scanf

    int num;

    srand(time(0));

    for (int i = 0; i < count; i++) {

        num = (rand() % (upper - lower + 1)) + lower;

    }

1. **Decimal Number to Binary Number**

#include<stdio.h>

#include<stdlib.h>

int main(){

    int a[10],n,i;

    printf("Enter the number jo aap convert karna chahte ho: ");

    scanf("%d", &n);

    for(i=0;n>0;i++){

        a[i]=n%2;

        n=n/2;

    }

    printf("Binary of Given Number is = ");

    for(i=i-1;i>=0;i--){

    printf("%d", a[i]);

    }

    return 0;

}

**OR**

#include<stdio.h>

#include<stdlib.h>

int main(){

    int a[10],n,i;

    printf("Enter the number jo aap convert karna chahte ho: ");

    scanf("%d", &n);

    for(i=0;n>0;i++){

        a[i]=n%2;

        n=n/2;

    }

    printf("Binary of Given Number is = ");

    for(i=i-1;i>=0;i--){

    printf("%d", a[i]);

    }

    return 0;

}

1. **Binary to Decimal**

#include<stdio.h>

#include<math.h>

int main(){

    int rem, dec = 0, i = 0;

    int n;

    printf("Enter the binary number: ");

    scanf("%d", &n);

    while(n!=0){

        rem = n%10;

        n/=10;

        dec = dec + rem\*pow(2, i);

        i++;

    }

    printf("Decimal mein ans --> %d", dec);

    return 0;

}

**OR**

#include<stdio.h>

void main(){

    int n , p=1;

    int dec = 0, i = 1, d;

    printf("Input a binary number: ");

    scanf("%d", &n);

    int n1 = n;

    for (int j = n; j>0; j= j/10){

        d = j % 10;

            if(i==1){

                  p=p\*1;

            }

            else{

                 p=p\*2;

            }

       dec= dec + (d\*p);

       i++;

    }

        printf("The equivalent Decimal  Number: %d", dec);

}

1. **Binary to Octal**

#include<stdio.h>

#include<math.h>

int main(){

    int sum = 0;

    int count = 0;

    int n;

    printf("Enter the binary number: ");

    scanf("%d", &n);

    int k = n;

    while(n!=0){

        count++;

        n = n/10;

    }

    int arr[count];

    for(int i=0; i<count; i++){

        int l = k%10;

        arr[i] = l;

        k = k/10;

    }

    for(int i=0; i<count; i++){

        // printf("%d", arr[i]);

        sum = (int)(sum + arr[i]\*pow(2, i));

    }

    // printf("Decimal --> %d \n", sum);

    int octal = sum;

    int sum\_count = 0;

    while(sum!=0){

        sum\_count++;

        sum = sum/10;

    }

    int oct[sum\_count];

    for(int i=0; i<sum\_count; i++){

        int l = octal%8;

        oct[i] = l;

        octal = octal/10;

    }

    for(int i=(sum\_count-1); i>=0;i--){

        printf("%d", oct[i]);

    }

    return 0;

}

1. **Octal to Decimal**

#include<stdio.h>

#include<math.h>

int main(){

    int sum = 0;

    int count = 0;

    int n;

    printf("Enter the octal Number: ");

    scanf("%d", &n);

    int k = n;

    while(n!=0){

        count++;

        n = n/10;

    }

    int arr[count];

    for(int i=0; i<count; i++){

        int l = k%10;

        arr[i] = l;

        k = k/10;

    }

    for(int i = 0; i<count; i++){

        // printf("%d", arr[i]);

        sum = (int)(sum + arr[i]\*pow(8, i));

    }

    printf("The decimal is %d", sum);

    return 0;

}

1. **Perfect Number:**

An integer number is said to be “perfect number” if its factors, including 1(but not the number itself), sum to the number.

E.g., 6 is a perfect number because 6=1+2+3].

1. **Abundant Number:**

A number n is said to be Abundant Number if sum of all the proper divisors of the number denoted by sum(n) is greater than the value of the number n. And the difference between these two values is called the abundance.   
Mathematically, if below condition holds the number is said to be Abundant number:

sum(n)> n

abundance = sum(n) – n

1. **Hexadecimal Number**

Hexadecimal Digits are a-f, A-F or 0-9

1. **Majority Element**

A majority element in an array A[] of size n is an element that appears more than n/2 times (and hence there is at most one such element).

1. **Palindrome**

If Number entered is equal to reversed number, then it is palindrome

1. **Harshad Number**

If a number is divisible by the sum of its digits, then it will be known as a Harshad Number. Some Harshad numbers are 8, 54, 120 etc.

1. **Pronic Number**  A Pronic number is a number which is the product of two consecutive integers, that is, a number of the form n(n + 1).
2. **Ugly Number** Ugly numbers are numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, … shows the first 11 ugly numbers. By convention, 1 is included

if((n==1)||(n%2==0 || n%3==0 || n%5==0) && (n%4!=0 && n%6!=0 && n%7!=0 && n%8!=0 && n%9!=0)){

        printf("%d is ugly", n);

    }

1. **Deficient Number**

Deficient Number if sum of all the divisors of the number denoted by divisors Sum(n) is less than twice the value of the number n. The Difference between these two values is called deficiency

1. **Fermat Number**

Fermat numbers are non-negative odd numbers which is valid for all values of k>=0. Only the first seven terms of the sequence are known till date. 2^2^k+1

1. **Mersenne Number**  A Mersenne prime is a prime number that is one less than a power of two. That is, it is a prime number of the form Mn = 2^n − 1 for some integer n.
2. **Hexadecimal Digit** Hexadecimal Digits are a-f, A-F or 0-9
3. **Abundant Number**

An Abundant Number (also known as excessive number) is a number in the number theory. which itself is smaller than the sum of all its proper divisors.

1. **Friendly Pair** Friendly Pair or Amicable numbers are two different numbers related in a way such that the sum of the proper divisors of each is equal to the other number.
2. **Pell Number**

Pn = 2\*Pn-1 + Pn-2

with seeds P0 = 0 and P1 = 1

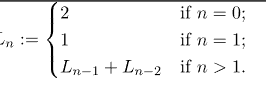
1. **Disarium Number**

A number is Disarium when sum of digits raised to power of respective positions is equal to number itself. E.g. 89, 135, 518 1^1 + 3^2 + 5^3 = 135

1. **Armstrong Number**

abcd... = a^n + b^n + c^n + d^n + …. (n is number of digits) E.g. 153 = 1\*1\*1 + 5\*5\*5 + 3\*3\*3

1. **Lucas Number**

****



1. **Automorphic Number**

A number is called Automorphic number if and only if its square ends in the same digits as the number itself.

1. **Fibonacci Series**

Fn = Fn-1 + Fn-2

F0 = 0 and F1 = 1

1. **Happy Number**

A **happy number** is a number which eventually reaches 1 when replaced by the sum of the square of each digit. For instance, 13 is a happy number because 1^2 + 3^2 = 1 and 1^2 + 0^2 = 1{\displaystyle 1^{2}+3^{2}=10}1^2{\displaystyle 1^{2}+0^{2}=1}1^21ff

A number which is not happy is called sad or unhappy number

1. **Keith Number**

Write a program in C to check if a number is Keith or not(with explanation). A n digit number x is called Keith number if it appears in a special sequence (defined below) generated using its digits. The special sequence has first n terms as digits of x and other terms are recursively evaluated as sum of previous n terms.

Input : x = 197 Output : Yes

197 has 3 digits, so n = 3

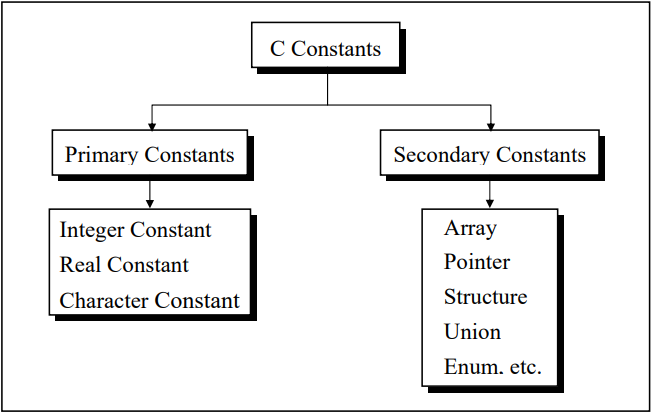
The number is Keith because it appears in the special sequence that has first three terms as 1, 9, 7 and remaining terms evaluated using sum of previous 3 terms. E.g. 1, 9, 7, 17, 33, 57, 107, 197, .....

1. **Narcissistic Number**

Narcissistic Number is a number that is the sum of its own digits each raised to the power of the number of digits

1. **Karpekar Number**

Number whose square in that base can be split in 2 parts that add upto the original number again!

 ***(IMP POINTS)***

**Rules for Constructing Integer Constants**

(a) An integer constant must have at least one digit.

(b) It must not have a decimal point.

(c) It can be either positive or negative. If no sign precedes an integer constant it is assumed to be positive.

(d) No commas or blanks are allowed within an integer constant.

(e) The allowable range for integer constants is -32768 to 32767.

**Rules for Constructing Real Constants**

Real constants are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form.

Following rules must be observed while constructing real constants expressed in fractional form:

* A real constant must have at least one digit.
* It must have a decimal point.
* It could be either positive or negative.
* Default sign is positive.
* No commas or blanks are allowed within a real constant.

In exponential form of representation, the real constant is represented in two parts. The part appearing before ‘e’ is called mantissa, whereas the part following ‘e’ is called exponent.

* + The mantissa part and the exponential part should be separated by a letter e.
  + The mantissa part may have a positive or negative sign. Default sign of mantissa part is positive.
  + The exponent must have at least one digit, which must be a positive or negative integer.
  + Default sign is positive.
  + Range of real constants expressed in exponential form is -3.4e38 to 3.4e38. **Ex.: +3.2e-5 4.1e8 -0.2e+3 -3.2e-5**

**Rules for Constructing Character Constants**

A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.

Both the inverted commas should point to the left. For example, ’A’ is a valid character constant whereas ‘A’ is not. The maximum length of a character constant can be 1 character. Ex.: 'A' , 'I' , '5'

**Variable names are names given to locations in memory**.

These locations can contain integer, real or character constants. In any language, the types of variables that it can support depend on the types of constants that it can handle. This is because a particular type of variable can hold only the same type of constant. For example, an integer variable can hold only an integer constant, a real variable can hold only a real constant and a character variable can hold only a character constant.

**Rules for Constructing Variable Names**

* A variable name is any combination of 1 to 31 alphabets, digits or underscores. Some compilers allow variable names whose length could be up to 247 characters. Still, it would be safer to stick to the rule of 31 characters.
* Do not create unnecessarily long variable names as it adds to your typing effort.
* The first character in the variable name must be an alphabet or underscore.
* No commas or blanks are allowed within a variable name.
* No special symbol other than an underscore can be used in a variable name.

**C Keywords**

Keywords are the words whose meaning has already been explained to the C compiler (or in a broad sense to the computer). The keywords cannot be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer. Some C compilers allow you to construct variable names that exactly resemble the keywords. However, it would be safer not to mix up the variable names and the keywords. The keywords are also called ‘Reserved words’.

**; acts as a statement terminator.**

\* and / are the arithmetic operators. The arithmetic operators available in C are +, -, \* and /. C is very rich in operators. There are about 45 operators available in C. Surprisingly there is no operator for exponentiation... a slip, which can be forgiven considering the fact that C has been developed by an individual, not by a committee.

**C Instructions**

There are basically three types of instructions in C:

* Type Declaration Instruction 🡪 To declare the type of variables used in a C program
* Arithmetic Instruction 🡪 To perform arithmetic operations between constants and variables.
* Control Instruction 🡪 To control the sequence of execution of various statements in a C program.

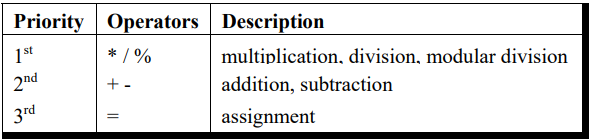
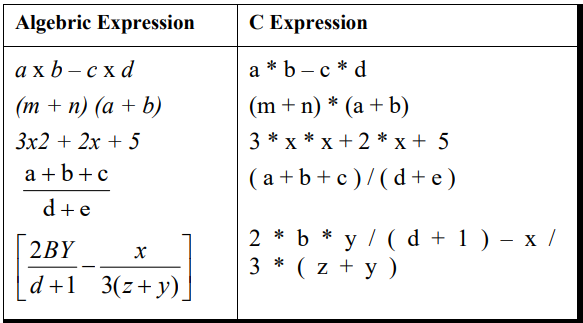
\*, /, -, + are the arithmetic operators. = is the assignment operator. 2, 5 and 3200 are integer constants. 3.2 and 0.0056 are real constants.

The variables and constants together are called ‘operands’

**A C arithmetic statement could be of three types**.

These are as follows:

* **Integer mode arithmetic statement** - This is an arithmetic statement in which all operands are either integer variables or integer constants. Ex.: int i, king, issac, noteit ; i = i + 1 ; king = issac \* 234 + noteit - 7689 ;
* **Real mode arithmetic statement** - This is an arithmetic statement in which all operands are either real constants or real variables. Ex.: float qbee, antink, si, prin, anoy, roi ; qbee = antink + 23.123 / 4.5 \* 0.3442 ; si = prin \* anoy \* roi / 100.0 ;
* **Mixed mode arithmetic statement** - This is an arithmetic statement in which some of the operands are integers and some of the operands are real. Ex.: float si, prin, anoy, roi, avg ; int a, b, c, num ; si = prin \* anoy \* roi / 100.0 ; avg = ( a + b + c + num ) / 4 ;
* It is very important to understand how the execution of an arithmetic statement takes place. Firstly, the right-hand side is evaluated using constants and the numerical values stored in the variable names. This value is then assigned to the variable on the left-hand side.

** **

**Control Instructions in C**

There are four types of control instructions in C. They are:

(a) Sequence Control Instruction

(b) Selection or Decision Control Instruction

(c) Repetition or Loop Control Instruction

(d) Case Control Instruction

The Sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program. Decision and Case control instructions allow the computer to take a decision as to which instruction is to be executed next. The Loop control instruction helps computer to execute a group of statements repeatedly.

A decision control instruction can be implemented in C using:

(a) The if statement

(b) The if-else statement

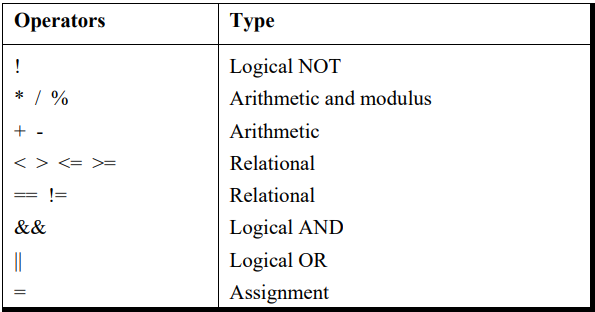
(c) The conditional operators

**🡪** Use of Logical Operators C allows usage of three logical operators, namely, &&, || and !. These are to be read as ‘AND’ ‘OR’ and ‘NOT’ respectively.

* **The ! Operator**

!(y < 10 ) This means “not y less than 10”. In other words, if y is less than 10, the expression will be false, since ( y < 10 ) is true. We can express the same condition as ( y >= 10 ).

The NOT operator is often used to reverse the logical value of a single variable, as in the expression if ( ! flag ) This is another way of saying if ( flag == 0 ) Does the NOT operator sound confusing? Avoid it if you want, as the same thing can be achieved without using the NOT operator.



**The Conditional Operators**

The conditional operators ? and : are sometimes called ternary operators since they take three arguments. In fact, they form a kind of foreshortened if-then-else.

**expression 1 ? expression 2 : expression 3**

What this expression says is: “if expression 1 is true (that is, if its value is non-zero), then the value returned will be expression 2, otherwise the value returned will be expression 3”

1. It’s not necessary that the conditional operators should be used only in arithmetic statements. This is illustrated in the following examples:

Ex.: **int i ;**

**scanf ( "%d", &i ) ;**

**( i == 1 ? printf ( "Amit" ) : printf ( "All and sundry" ) ) ;**

Ex.: **char a = 'z' ;**

**printf ( "%c" , ( a >= 'a' ? a : '!' ) ) ;**

1. The conditional operators can be nested as shown below.

**int big, a, b, c ;**

**big = ( a > b ? ( a > c ? 3: 4 ) : ( b > c ? 6: 8 ) ) ;**

1. Check out the following conditional expression: a > b ? g = a : g = b ; This will give you an error ‘Value Required’. The error can be overcome by enclosing the statement in the : part within a pair of parenthesis. This is shown below: a > b ? g = a : ( g = b ) ; In absence of parentheses the compiler believes that b is being assigned to the result of the expression to the left of second =. Hence it reports an error.
2. The limitation of the conditional operators is that after the ? or after the : only one C statement can occur. In practice rarely is this the requirement. Therefore, in serious C programming conditional operators aren’t as frequently used as the if-else.

**Why Use Functions**

1. Writing functions avoids rewriting the same code over and over.
2. Using functions it becomes easier to write programs and keep track of what they are doing. If the operation of a program can be divided into separate activities, and each activity placed in a different function, then each could be written and checked more or less independently. Separating the code into modular functions also makes the program easier to design and understand.

In the programs the moment closing brace ( } ) of the called function was encountered the control returned to the calling function. No separate return statement was necessary to send back the control. This approach is fine if the called function is not going to return any meaningful value to the calling function. In the above program, however, we want to return the sum of x, y and z. Therefore, it is necessary to use the return statement.

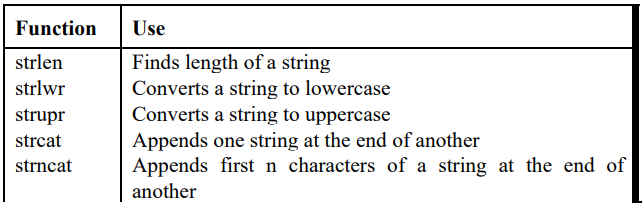
**ARRAYS**

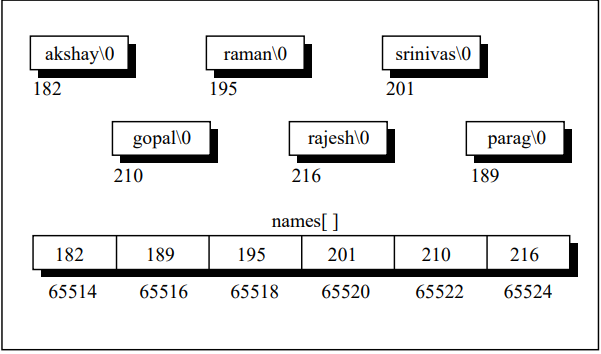
**Array[p][q][r]**

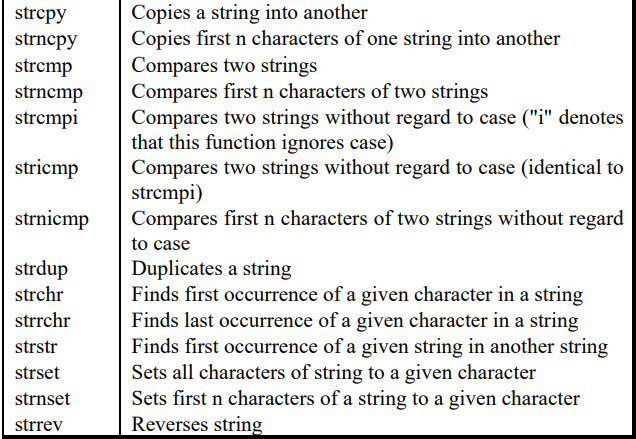
**\*( \*( \*( Array + p ) + q ) + r )**

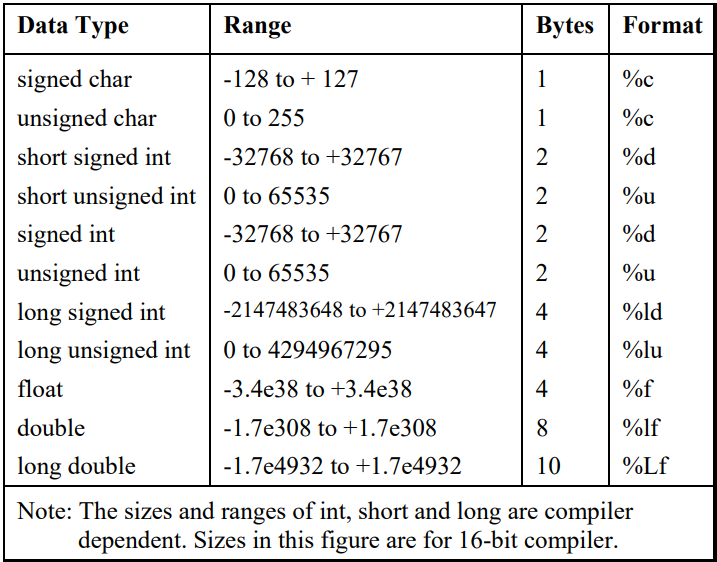
arr[2][3][1]

\*( \*( \*( arr + 2 ) + 3 ) + 1 )



****



****

**See Storage Classes on pg-238 and Conditional and Miscellaneous Directives on pg-275 in Let Us C**

**SUMMARY:**

1. The three primary constants and variable types in C are integer, float and character.
2. A variable name can be of maximum 31 characters. Do not use a keyword as a variable name.
3. An expression may contain any sequence of constants, variables and operators.
4. Operators having equal precedence are evaluated using associativity. Left to right associativity means that the left operand of a operator must be unambiguous whereas right to left associativity means that the right operand of an operator must be unambiguous.
5. Input/output in C can be achieved using scanf( ) and printf( ) functions.
6. There are three ways for taking decisions in a program. First way is to use the if-else statement, second way is to use the conditional operators and third way is to use the switch statement.
7. The default scope of the if statement is only the next statement. So, to execute more than one statement they must be written in a pair of braces.
8. An if block need not always be associated with an else block. However, an else block is always associated with an if statement.
9. If the outcome of an if-else ladder is only one of two answers then the ladder should be replaced either with an else-if clause or by logical operators.
10. && and || are binary operators, whereas, ! is a unary operator.
11. In C every test expression is evaluated in terms of zero and non-zero values. A zero value is considered to be false and a non-zero value is considered to be true.
12. Assignment statements used with conditional operators must be enclosed within a pair of parentheses.
13. A break statement takes the execution control out of the loop
14. A continue statement skips the execution of the statements after it and takes the control to the beginning of the loop.
15. A do-while loop is used to ensure that the statements within the loop are executed at least once.
16. The operators +=, -=, \*=, /=, %= are compound assignment operators. They modify the value of the operand to the left of them.
17. When we need to choose one among number of alternatives, a switch statement is used.
18. The switch keyword is followed by an integer or an expression that evaluates to an integer.
19. The case keyword is followed by an integer or a character constant.
20. The control falls through all the cases unless the break statement is given.
21. The usage of the goto keyword should be avoided as it usually violets the normal flow of execution.
22. An array is also known as a subscripted variable.
23. An array is similar to an ordinary variable except that it can store multiple elements of similar type.
24. Compiler doesn’t perform bounds checking on an array.
25. The array variable acts as a pointer to the zeroth element of the array. In a 1-D array, zeroth element is a single value, whereas, in a 2-D array this element is a 1-D array.
26. On incrementing a pointer it points to the next location of its type.
27. Array elements are stored in contiguous memory locations and so they can be accessed using pointers. Only limited arithmetic can be done on pointers.
28. If the array is declared as a global one or as static in a function, then all elements are initialized to zero if they aren't initialized already.
29. **Address of a floating-point variable is always a whole number.** True.
30. malloc( ) function can be used to allocate space in memory on the fly during execution of the program.
31. Though scanf( ) can be used to receive multi-word strings, gets( ) can do the same job in a cleaner way.
32. Both printf( ) and puts( ) can handle multi-word strings
33. Strings can be operated upon using several standard library functions like strlen( ), strcpy( ), strcat( ) and strcmp( ) which can manipulate strings. More importantly we imitated some of these functions to learn how these standard library functions are written.
34. Though in principle a 2-D array can be used to handle several strings, in practice an array of pointers to strings is preferred since it takes less space and is efficient in processing strings.
35. A structure is usually used when we wish to store dissimilar data together. Structure elements can be accessed through a structure variable using a dot (.) operator.
36. Structure elements can be accessed through a pointer to a structure using the arrow (->) operator.
37. All elements of one structure variable can be assigned to another structure variable using the assignment (=) operator. It is possible to pass a structure variable to a function either by value or by address.
38. It is possible to create an array of structures.
39. We can use different variations of the primary data types, namely signed and unsigned char, long and short int, float, double and long double. There are different format specifications for all these data types when they are used in scanf( ) and printf( ) functions.
40. The maximum value a variable can hold depends upon the number of bytes it occupies in memory. By default all the variables are signed. We can declare a variable as unsigned to accommodate greater value without increasing the bytes occupied.
41. We can make use of proper storage classes like auto, register, static and extern to control four properties of the variable—storage, default initial value, scope and life.
42. The preprocessor directives enable the programmer to write programs that are easy to develop, read, modify and transport to a different computer system.
43. We can make use of various preprocessor directives such as #define, #include, #ifdef - #else - #endif, #if and #elif in our program. The directives like #undef and #pragma are also useful although they are seldom used.
44. C is a Middle Level Language.
45. Tokens are the smallest elements of a program, which are meaningful to the compiler. The following are the types of tokens: Keywords, Identifiers, Constant, Strings, Operators, etc.
46. Max Length of variables in C 🡪 32
47. Integer division results in Truncating the fractional part.
48. Explicit type conversion is known as Casting.
49. p++ executes faster than p+1 because p++ is a single instruction
50. Header files in C contain Library functions.
51. The printf() function retunes which value when an error occurs Negative value
52. Symbolic constants can be defined using constant.
53. Null character is represented by \0
54. C supports 3 looping constructs.
55. Number of relational operators in C are 6
56. A link is an analysing tool in C.
57. The continue command cannot be used with for.
58. A multidimensional array can be expressed in terms of array of pointers rather than as pointers to a group of a contiguous array.
59. A pointer to a pointer is a form of multiple indirections and a chain of pointers
60. Pointers are of unsigned integer data types.
61. Using C language programmers can write their own library functions
62. C language is used in development for Databases, Graphic Applications, Word Processors etc.
63. 31 characters used to distinguish Identifier or Names of Functions and Global variables.

**Object-Oriented-Programming**

An object-oriented programming approach is a collection of objects and each object consists of corresponding data structures and procedures. The program is reusable and more maintainable. The important aspect in OOP is a class which has similar syntax that of structure.

**Class** 🡪 It is a collection of data and member functions that manipulate data. The data components of class are data members and functions that manipulate the data are **member functions**.

It can also called as blue print or prototype that defines the variables and functions common to all objects of certain kind. It is also known as user defined data type or ADT(abstract data type) A class is declared by the keyword class.

**Object 🡪** Instance of a class is called as object.

**Access Control 🡪** Access specifier or access modifiers are the labels that specify type of access given to members of a class. These are used for data hiding. These are also called as visibility modes. There are three types of access specifiers

1. **Private** - If the data members are declared as private access, then they cannot be accessed from other functions outside the class. It can only be accessed by the functions declared within the class. It is declared by the key word **private**.

2. **Public** - If the data members are declared public access then they can be accessed from other functions out side the class. It is declared by the key word **public**.

3. **Protected** -The access level of protected declaration lies between public and private. This access specifier is used at the time of inheritance.

**IMP POINTS**

1. **Accessing Members: -** Dot operator is used to access members of class
2. If the access specifier is not specified in the class the default access specifier is private
3. All member functions are to be declared as public if not they are not accessible outside the class.
4. There are 2 types of scope. First there is **Local Scope** which is local to the function of the program and the other one is **Global Scope** which has visibility to all functions of the program. **Scope Resolution Operator is “::”**
5. **Inline Functions 🡪** An inline function is a function that is expanded in line when it is invoked. Inline expansion makes a program run faster because the overhead of a function call and return is eliminated. It is defined by using key word “**inline**”.

**Why Inline Functions?**

1. One of the objectives of using functions in a program is to save some memory space, which becomes appreciable when a function is likely to be called many times.
2. Every time a function is called, it takes a lot of extra time in executing a series of instructions for tasks such as jumping to the function, saving registers, pushing arguments into the stack, and returning to the calling function.
3. When a function is small, a substantial percentage of execution time may be spent in such overheads. One solution to this problem is to use macro definitions, known as macros. Pre-processor macros are popular in C. The major drawback with macros is that they are not really functions and therefore, the usual error checking does not occur during compilation.
4. C++ has different solution to this problem. To eliminate the cost of calls to small functions, C++ proposes a new feature called inline function.

**Why not Inline Functions?**

1. A function that is returning value, if it contains switch, loop or both then it is treated as normal function.
2. if a function is not returning any value and it contains a return statement then it is treated as normal function
3. if a function is not returning any value and it contains a return statement then it is treated as normal function
4. If the inline function is declared as recursive function, then it is executed as normal function.
5. **Memory Allocation for Objects:** Memory for objects is allocated when they are declared but not when class is defined. All objects in a given class uses same member functions. The member functions are created and placed in memory only once when they are defined in class definition
6. **Static Class Members 🡪 a) Static Data Members b) Static Member Functions**

**Static Data Members** 🡪 A data member of a class can be qualified as static. A static member variable has certain special characteristics:

1. It is initialized to zero when the first object of its class is created. No other initialization is permitted.
2. Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
3. It is visible only within the class, but its lifetime is the entire program.
4. Static data member is defined by keyword **static**

**Static Member Functions 🡪** Like static member variable, we can also have static member functions. A member function that is declared static has the following properties: A static function can have access to only other static members (functions or variables) declared in the same class. A static member function is to be called using the class name (instead of its objects) as follows: **class-name :: function-name;**

1. **Arrays of Objects 🡪** Arrays of variables of type "class" is known as "Array of objects". An array of objects is stored inside the memory in the same way as in an ordinary array.