C++ is a high level programming language but it provide very easy way to work with computer memory so, it kinda works as low level also.

g++ has faster execution than other languages as it is a compiled language not interpreted.

It has richer library than C and also gives support of STL which is very crucial for CP.

So, C++ mainly used for writing codes of operating systems, games etc.

Since, C++ is a compiled language, it gives machine code which will be machine specific after compilation, so, you need to compile it everytime you want to run it.

Whereas in case of interpreted languages they gives machine code which are independent of specific machine, so, do not need to interpret it again when

you want to run it, At this point Java, Python like interpreted languages beat C++ like compiled languages.

But since C++ is compiled it is faster than interpreted languages like Java & Python.

Also C++ gives better way to maintain memory flow and control it than other languages.

→ Input/output in C++ :-

→ Cout <<

Standard output stream to the standard output devices like monitor & consol.

But 'Cout<<' can also be modified to give output stream to other output devices.

→ Cin >>

Standard input stream from standard input device like keyboard.

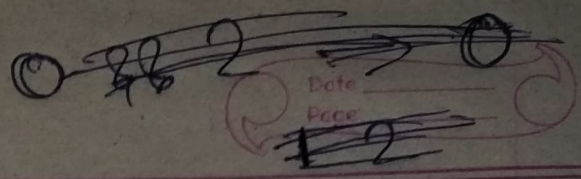
→ Data types and their ranges :-

C++ gives independence to the compiler to decide the size of a data type, it compiler decide the size of int will be 8 byte then it will be 8 byte, but to get know real size of data type in your computer, you can use 'sizeof()' operator.

Also given below the range the data types :-

char	-128 to 127
unsigned char	0 to 255
int	-2^{31} to $2^{31}-1$
long int	-2^{31} to $2^{31}-1$ (here size of long int is 4 byte)
long long int	-2^{63} to $2^{63}-1$
unsigned int	0 to $2^{32}-1$

NOTE :- You can't create two variable of same name in same scope, but they can be created in different scope.



ex -

```
int x;
int main()
{
    int x=10;
    {
        int x=20;
        cout<<x;
    }
}
```

output = 20

as 3 'x' variable are in different scope.

ex -

```
int main()
{
    int x=10;
    int x=20;
    cout<<x;
}
```

error will throw by compiler

→ Error in C++ :-

→ Sytax error
→ Semantic error → occurs when your code doesn't make sense
→ Linker error → if invoked function declaration is not available.
→ Runtime error
→ Logical error

most difficult (Cross)

NOTE:- ~~Associative~~ Assignment operator has associativity from right to left.

Also non-zero integer value at place of bool value consider as true, where as zero value at place of bool will consider as false.

NOTE:- Logical and operator returns true is both the statements are true and return false if one of them is false. So, if first statement is false then it will not execute second statement.

For Logical OR operator, it returns true is one of them is true, so if first statement is true, it will return true without executing second statement.

ex-

```
int main()
{
    x = 3
    bool r = (x==3) || (x++);
    cout<< r <<" "<< x;
    return 0;
}
```

Output :-

(1) 3

↑ for true

Value of x will be 3 as (x++) statement will not execute as (x=-3) statement is true.

NOTE:- ARITHMETIC and Relational operator has associativity from left to right.

→ auto keyword in C++ :-

auto keyword is used to define storage type of variable, but, since in C++ all variable are default automatic (created at the point of definition are destroyed when block exited), there remains no use of it.

But in modern C++ auto keyword has given new meaning 'Automatic Determination of Data-type during Assignment.

So, if we use auto keyword we don't need to specify the data type of the variable anymore.

cin.ignore()

ex-

```
auto d = 5.67;
auto i = (5+6);
typef struct node{
    int x;
    node(int x){this->x = x;}
};
auto root = new node(5);
```

here, we don't need to explicitly state the data-type of resultant expression on RHS during the assignment.

→ precision of floating point numbers in C++ :-

→ floor():-

floor() rounds off the given value to the closet integer, which is less than the given value.

- floor(1.4) = 1
  floor(1.8) = 1
  floor(-1.7) = -2 ; floor(-1.3) = -2

→ ceil() :-

ceil() rounds off the given value to the closest integer which is more than the given value.

ceil(.2) = 1 ; ceil(1.7) = 2

ceil(-1.2) = -1 ; ceil(-1.9) = -1

→ trunc() :-

trunc() removes digits after decimal point.

trunc(1.7) = 1 trunc(2.1) = 2
trunc(1.9) = 1 trunc(-1.7) = -1

→ round() :-

round() round off the digit to the closest integer.

round(1.4) = 1 . round(1.7) = 2
round(-1.7) = -2 . round(-1.2) = -1

→ Setprecision ():-

When setprecision() used with fixed precise the digit after decimal up to number mentioned in brackets of setprecision()

ex:-

```
double pi = 3.141597
cout<<fixed <<setprecision(1)<< pi <<endl;
cout<<fixed <<setprecision(3)<<pi <<endl;
```

Output- 3.1
3.141

→ Jump statements in C++:-

→ Continue:- Skip current iteration of loop.
→ break:- come out of loop

→ return :- Come out of function.
→ goto :- To jump to any lable.

NOTE:- During function overloading you should remember that two function can't differ by their return type.

NOTE:- To know element in array use can use formula.

```
int n = sizeof(arr)/sizeof(arr[0]);
```

→ Arrays and vectors in C++ :-

As we know arrays are used to store same data types at contiguous location.

Because of this they provide easy and fast access to data element and there is less burden on chache memory.

There are many disadvantages of array too-

We have to provide size of array at time of declaration or initialization and we can't easily change size of array and do operations like deletion or insertion.

To solve this problem we have vectors in C++, which allocate array dynamically and you not need to provide size of array at time of declaration.

Vectors have all features of array + it provides dynamic memory allocation and feature like insertion and deletion.

→ Vectors :-

To use vectors you have to use

```
#include<vector>
```

It is used to make array of dynamic size.

ex -

```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
```

// This will make same thing as

10	20	30

v

```
for(int i=0; i<v.size(); i++)
{
    cout<<v[i]<<" ";
}
    return 0;
}
```

Now instead of writing this for loop for printing we can write this loop like that also,

```
for(auto x : v)
{
    cout<<x<<" ";
}
```

This for loop works same as above discussed in for loop.

Here, we have learnt two methods to traverse through vector, there is one methode by iterators which we will learn later.

Another methode to initialize a vector.-

→ size of vector

vector<int> v(10,5);

→ value in each place.

So it will create an array equivalent as,

5	5	5	5	5	5	5	5	5	5

One methode to initialize a vector is

vector<int> v{10,20,30};

NOTE:- Along with vectors there are many built-in functions in C++ STL vector, by using using them it makes very easy to use and operate on vectors.

We know more about vectors in later classes.

Some questions on array/vectors:-

- Finding maximum element in

Array :-

Method 1:-

```
int main()
{
  int arr[]={10,200,30,500,600,80};

  int n= sizeof(arr)/sizeof(arr[0]);
  int x=arr[0];
 for(int i=1;i<n;i++)
  {
    x=max(x,arr[i]);
  }
 cout<<x<<endl;

  return 0;
}
```

Method 2:-

by using built in function

```
*max_element(arr,arr+n)
```

It can be used only for 1-D array/vector.

size of array

```
int main()
{
 int arr[]={10,20,100,40,500,8};
```

```
int n= sizeof(arr)/sizeof(arr[0]);
int x= *max_element(arr, arr+n);
cout<<x;
return 0;
}
```

- Finding maximum element in an ~~array~~ vector:-

For this also we use * max_element() function.

```
int main()
{
    vector<int>v{10,20,100,40,500,79};
    int x= *max_element(v.begin(),
                        v.end());
    cout<<x;
    return 0;
}
```

// begin() and end() are also built in function in vector STL to keep hold of

first and end marker element (special end element in vector beyond its size) of vector. and they return iterators to that elements.

- Finding sum of array using function :-

We use accumulate() function to find sum of elements in array/vector.

We also have to include <numeric> header file to use this function.

```
#include<numeric>
#include <iostream>

using namespace std;

int main()
{
    int arr[] = {10, 20, 5, 40};
    int n = sizeof(arr)/sizeof(arr[0]);
    int S=0;

        S=accumulate(arr, arr+n, S);
        cout<<S;
        return 0;
}
```

To get sum of vector elements we use

S= accumulate(v.begin(), v.end(), S);

- Find the element that appears once in an array where every other element appears twice :-

There are may methods to solve this question but we learn most effective one and simple here,

Methode 1:- using XOR(^) operator :-

XOR of all elements of array gives us the number with a single occurance. It is due to two facts of XOR(^)

- XOR of a number with itself is 0.
- XOR of a number with 0 is number itself.

So, for example we have an array

arr[5, 7, 5, 6, 6, 8, 8]

Since, XOR is associative and commutative So, ~~we can~~ it is same as

int x= 7^(5^5)^(6^6)^(8^8)

= 7^0^0^0 =7

```
#include <iostream>
using namespace std;

int find_single(int arr[], int ar_size);

int main(){
    int arr[]={5, 7, 5, 6, 6, 8, 8};
    int n= sizeof(arr)/ sizeof(arr[0]);

    cout<<" single element in array is: "
        << find_single(arr, n);

    return 0;

}

int find_single (int ar[], int ar_size)
{
    int res =0
    for(int i=0; i<ar_size; i++)
    {
        res = res^ ar[i];
    }
    return res;
}
```

Methode 2:-

This approach is not as effective as previous one. but another way to get desired results.

If we add each distinct element and multiply 2 with sum and then subtract sum of array to get desired result.

→ String :-

Strings can be considered as array of characters.

There are two types of string in C++ :-

→ C-type String :-

C-type string are defined using character array. There is an extra character called NULL (\0) character appended to the character array to mark it's end.

Declaration & initialization of string :-

```
char str[size];
char str[] = "Geeks";
char str[5] = "Geeks";
```

In this type of initialization we don't need to put \0 at last.

```
char str[] = {'G', 'e', 'e', 'k', 's'};
```

but in this type of initialization it is prefered to put '\0' as a character at last.

So, we prefer 2nd and 3rd declaration type in C type string.

There are many function provided by <cstring> header file to work on C type strings, some of them are

- strlen()

→ used to calculate length of given string.

ex-

```
int x = strlen(str);
```

- strcat()

→ used to concatinate two strings.

```
strcat(str1, str2);
```

ex-

```
char str1[] = "Source";
char str2[] = "destination";

cout<<strcat(str1, str2);
```

- StrCmp():-

-> Used to Compare two strings which lexicographical order. string word comes first in alphabatical order.
Unlike integer and floating point number we can't compare two c type string. we must have to use this function to do so.

It returns

0 if both are equal/same.
(+ve) if 1st string appears after 2nd string
(-ve) if 1st string appears before 2nd string.

Ex.-

```
char str1[] = "abc";
char str2[] = "abcd";

cout<< StrCmp(str1, str2);
```

output:-
-100 (-ve) value as 'abc' is smaller than 'abcd'.

- strcpy()

→ Used to copy one string value to other string.

It is also used when we have already declared a Ctype string and later want to initialize it, we can't do it as integer, we must need strcpy() function.

ex-

```
Char str1[10];
strcpy(str1, "Anuj");
Str str2[] = "Akash";
strcpy(str1, str2);
```

- strstr()

→ Used for string searching. Finds and returns pointer to the first occurance of one string in another. Here also 2nd string is searched in the 1st string.

```
int main()
{
    char s1[] = "Geeks for Geeks";
    char s2[] = "for";
    char *p;

    p = strstr(s1, s2);

    if (p)
    {
    cout << "First occurrence of " << s2 << " in "
         << s1 << " is " << p << endl;
    }
    else
    {
        cout << "string not found";
    }

    return 0;
}
```

NOTE:- now-a-day it is more appropriate to use C++ type string over C type string, as, C type string has not as much functionalities and operations as C++ type string have.

In C type string we can't directly assigne value to already declared string, we need strcpy() for that but in c++ type string can do so, as integers and other data types.