**Basics of C++**

Structure of program

#include <iostream>

using namespace std;

int main (){

cout << "Hello World!";

return 0;}

**int main ()** 🡪

**Syntax:**

**#include <iostream> 🡪** header file. Increases functionality helps in input output.

**#include<climits>** 🡪includes these 2 constants - INT\_MAX (shows maximum int value that can be stored inside the integer), INT\_MIN (shows minimum int value that can be stored inside the integer)

**using namespace std;**

**int main () { 🡪** main is entry point. Int means gives integer value in last.

**return0;**

**}**

**Comments**

// 🡪 Single line comment

/\* \*/ 🡪 Multi line comment

**Variables**

a variable as a portion of memory to store a determined value.

a = 5;

b = 2;

a = a + 1;

result = a - b;

Int p, q; 🡪 declaring variables

Float r, s;

unsigned short int NumberOfSisters;

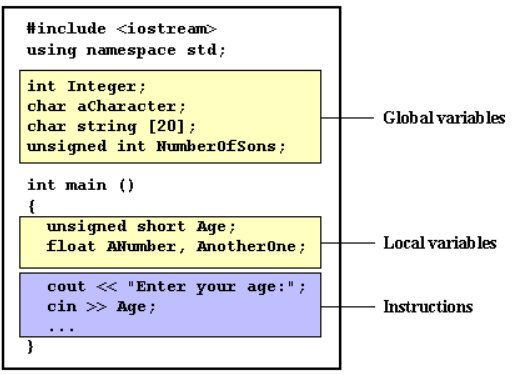
int p = 1; 🡪 assigning the values to the variables. / initialization of variables

float q = 2;

int a (0);

**Scope of variables**

A global variable is a variable declared in the main body of the source code, outside all functions, while a local variable is one declared within the body of a function or a block.



**Strings**

Variables that can store non-numerical values that are longer than one single character are known as strings.

string mystring = "This is a string";

**Data types**

int, bool, char, double, float

**Int** – only stores integers 🡪 int x = 100;

**Float** – stores decimals 🡪 float x = 10.14;

**Boolean** – stores true/false/0/1 🡪 bool x = true;

**Double** – stores bigger values than float 🡪 double x = 10.10294829824;

**Char** – stores characters 🡪 char ch = ‘A’;

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Size\*** | **Range\*** |
| char | Character or small integer. | 1byte | signed: -128 to 127 unsigned: 0 to 255 |
| short int (short) | Short Integer. | 2bytes | signed: -32768 to 32767 unsigned: 0 to 65535 |
| int | Integer. | 4bytes | signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295 |
| long int (long) | Long integer. | 4bytes | signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295 |
| bool | Boolean value. It can take one of two values: true or false. | 1byte | true or false |
| float | Floating point number. | 4bytes | +/- 3.4e +/- 38 (~7 digits) |
| double | Double precision floating point number. | 8bytes | +/- 1.7e +/- 308 (~15 digits) |
| long double | Long double precision floating point number. | 8bytes | +/- 1.7e +/- 308 (~15 digits) |
| wchar\_t | Wide character. | 2 *or* 4 bytes | 1 wide character |

**Sizeof** – tells how much memory does it occupy 🡪 cout<<”int”<<sizeof(x)<<endl;

\*\* dividing integer by integer gives output in integer only

**Int / int = int**

**Float / integer = Float**

**Integer / float = Float**

These are called typecasting\*\*

Typecasting is of 2 types

1. **Implicit** 🡪 which happens automatically
2. **Explicit 🡪**

**%** 🡪 modulus – gives the remainder

**\*Data type modifiers\*:**

**Unsigned int** – store only positive value, gives 1 bit more

(By default)**signed int** - stores both negative and positive values, gives one bit less

**Long** - expands the storage of the datatype

**Short** – decrease the storage of the datatype

**Wrap around** 🡪 whenever we try to exceed the maximum range of int we will end upon the opposite side.

PS C:\Users\shwet\Desktop\c++ codes> git add .

PS C:\Users\shwet\Desktop\c++ codes> git commit -m "new patttern code added"

PS C:\Users\shwet\Desktop\c++ codes> git push origin main

**Loops**

**While 🡪**  entry controlled //continues till the condition is true // in this we check the condition when loop is started

Int i = 1; // initial condition

While(i<=n){ //stopping criteria

i = i + 1; update condition

}

**Do while 🡪** exit controlled // it is executed at least once becoz it checks the condition after executing the loop once// it doesn’t check for initial condition is true or false

Do{

Update condition

}while(condition)

**If else 🡪**

If(n % 2 == 0 ){// if condition is true if block will be executed else else block will be executed.

}

Else{

}

**For 🡪** is same as while loop // entry controlled loop //

For (initialization ; stopping criteria ; update){}

**Break 🡪** it is used to stop a loop based upon certain condition and end the loop.

**Continue 🡪** it skips the particular condition and goes to the starting of the loop.

**Prime no 🡪** if (n % i == 0) – not prime / if (i == n) – prime

**Global variable 🡪** ::x – used for getting value of global variable.

**Cin.get() 🡪**  it reads any single char including spaces new line from input

**Operators**

**Assignment 🡪 = --** assignsthe value

**Bitwise 🡪** operates on bits

4 types

1. & -- bitwise And
2. | -- bitwise Or
3. ^ -- bitwise Xor
4. ~ -- bitwise Not

**Shift 🡪**  << left shift , >> right shift

**Unary 🡪** they operate on a single operand

& - address of operator

\*- dereferencing

New

Delete

~ - logical

* - minus for negative

+ - plus for positive

**Arithmetic 🡪**  +, - , \* , /, %,

**Ternary 🡪** condition ? if true print this statement : if false print this statement

Condition ? true statement: false statement

**Comma 🡪**  , for separating something – int a, b;

**Relational 🡪** <, >, <=, >=, ==, !=

**Logical 🡪** &&, || -- for combining different expressions

**Post increment/decrement 🡪** x++, x- -

**Pre increment/decrement 🡪** ++x, - -x

**Compound assignment 🡪**  x +=5 / x= x+5

+=, -=,\*=,/=,%=, <<=,>>=

**Switch case 🡪**

Switch (ch){

Case ’a’ : \_\_\_\_\_; break;

Case ‘b’ :\_\_\_\_\_\_; break;

Default : \_\_\_\_\_\_\_\_;

}

**Function 🡪**

void sayhello() {}

Void – not returning anything (return type) sayhello – function name () – for arguments

Sayhello(); in main function means calling this function

**Stack 🡪** Last in first out(LIFO) in this there is a container in which we put any some functions the function we entered at last will be executed at first and the function we entered at first will be ended at last.

**Array 🡪** a collection of items stored in a continuous memory location

It contains only one type of data

Each item has index and index is started from 0

Datatype name [size];

Int arr[100]; -- creating array

Int a[10] = {0}; initialization of an array.

a[2] = {5}; updating array

cout<<a[2]; printing item at 3rd position;

last index of an array is n-1

size of the array = n\* bytes per bucket

**Linear search 🡪**  helps us to find a particular element in an array

in this we traverse the array elements one by one to find the element

for(i = 0; i<=n-1; i++){

if (a[i] == key) {cout<<key<<”found at”<<i<<”index”}

break;}

if(i==n){

cout<<key<<”key not present”}

**INT\_MIN 🡪**  smallest no in int

**INT\_MAX 🡪**  largest no in int

**Binary search 🡪** efficient way to search in a sorted array

In this we take a starting point and an ending point and then we have to divide the array in two parts then we find the mid element by adding start and end point and dividing the sum by 2 then we see that if the key is less than mid if yes then we will put end mid-1 else start will be mid+1 And if the key is = mid we will return the mid.

Int s = 0;

Int e = n-1;

While (s<=e){

Int mid = (s+e)/2;

If(a[mid] == key){

Return mid;}

Else If(a[mid]>key){-

E = mid -1;}

Else{

S = mid + 1;

}return -1

**Selection sort 🡪** arrange a randomly shuffled array to increasing or decreasing order.

At first we take first element and then check the minimum element in the array then we swap that min element with the first place then we move the first place to next place so that next min element should be stored on the next point.

For(int i = 0; i<n-1; i++){

Int min\_index = i;

For(int j = i; j<=n-1;j++){

If(a[j]<a[min\_index]){

Min\_index = j;

}

}

Swap(a[i],a[min\_index]);

}

**Bubble sort 🡪** take the larger element towards the end

In this we just take an element and check that is it bigger than the next element so we will swap the elements and that’s how we take it towards the end of the array. And decrease the end by 1 and also by no of iterations

For(int itr = 1; itr<=n-1;itr++){

For(int j = 0; j<=(n-itr-1);j++){

If(a[j]>a[j+1]){

Swap(a[j],a[j+1]);

}

}

}

**Insertion Sort 🡪**  Insert the current element in right position

In this we fix an element that is in right position then we check that whether the element is smaller or larger than the element if its smaller then we shift that element to the fixed elements place and then check that fixed element from next no. if its bigger then it stays on that place only.

For (int i = 1; i<=n-1; i++){  
 int e – arr[i];

Int j = i-1;

While(j>=0 and arr[j]>e){

a[j+1] = a[j];

j = j-1;

}

a[j+1] = e;

}

**Sort() 🡪** it is a function comes under the library (#include<algorithm>) it has many parameters (Start point of container a, ending point a + n) // similar to bubble sort

Sort(a, a + n);

\*\*If we want to get decreasing order we have to create a comparator function which compares the elements

Bool compare(int a, int b){return a>b}

Sort(a, a + n, compare);

**Kadane’s algo for max subarray sum 🡪** in this we find the max sum of sub array

In this we search add every element of array and if the starting element is in negative we take it as 0.we don’t go in negative we take the no till 0 and then add another and the max sum is the last added sum we got.

Int cs = 0;

Int ms = 0;

For (int i = 0; i<n;i++){

Cs = cs + a[i];

If(cs<0){cs=0;}

ms = max(cs,ms);

}

cout<<”maximumsum is “<<ms<<endl;

**2D Array 🡪**  it has two dimensions rows and cols, it allows us to make matrix.

M\*n array have m-1,n-1 as last box index

Int a[rows][cols]; creation

Stored as linear or we can say as rows form or cols form

For traversing 2d array

Int val = 1;

For (int row = 0; row <=4; row++){

For(int col = 0;col<=2; col++){

a[row][col] = val;

val = val+1;

cout<< a[row][col]<<” ”;

}

Cout<<endl;

}

**Rotate image 🡪** in this we move rows to cols

For reversing each row

For(int row=0; row<n; row++){

Int startcol = 0;

Int endcol = n-1;

While (startcol <endcol){

Swap(a[i][startcol],a[i][endcol]);

Startcol++;

Endcol--;

}

}

To take transpose

For(int i=0; i<n; i++){

For (int j = 0;j<n; j++){

if(i<j){

swap(a[i][j],a[j][i]);

}

}

}

And we can do the same thing using stl reverse method and we have to do the transpose in both

For(int i =0; i<n; i++){

Reverse(a[i],a[i]+n);

}

**2d Character arrays 🡪**  in this we store characters

Char a[][10] = {{‘a’,’b’,’\0’},{‘d’,’e’,’f’,’\0’}}; -- initialisation

Char a[][10] = {“ab”, “Def”,”ghi”};

Char a[10][10]; c[0][0] = ‘A’;

Storing and reading char array

Int n;

Cin >>n;

Cin.get();

For(int i=0;i<n;i++){

Cin.getline(a[i],1000);}

To print all strings

For (int i=0; i<n;i++){

Cout<<a[i]<<endl;}

Cout<<a<<endl; // when we print a in any integer array we get the address and when wee print it in char array we get the element

**Strlen() 🡪** returns the length of the c sting

**Strcpy() 🡪** it copies the string from one array to another and it comes under the #include <cstring>

**String class 🡪**  it is an alternative for character array. It wraps our array in a string object. String is a container. We have to include the header file #inlcude <string>

String s; -- initialisation

String s1(“Hello”);

String s2= “hello world”;

String s3(s2); // it copies the data of s2 in s3

String s4 = s4; //it will be same as s3

If (S1.**empty()**) // this empty function returns a Boolean value

S1.**append**(“I love c++”); // aapend is used to add this string to the end of the previous

S1 **+=** “and python”; // it also add this at last of the string

S1.**clear()**; // it removes all the elements

S1.**compare**(s2);// it compare the string it returns an integer == equal, > 0 or < 0

Int idx = s.**find**(“love”); // for finding the substrings

Cout>>idx<<endl;

String word = “love”;

Int len = word.**length()**;// for getting the word length

s.**erase**(idx,length+1); // for removing particular word from the string

**auto🡪** it defines the datatype automatically which is in right

for(auto it = s1.**begin()**; it!s1.**end()**; it++){ begin and end are the iterators  
cout<<(\*it)<<”,”; //\*it is the pointe here it points to the value

//for each loop for iterating

For(auto c:s1){cout<<c<<”,”; }

Char **\*ptr** = **Strtok(s,” ”);** // \*ptr refers to the pointer and strtok refers to the string tokenisation this breaks the string where space comes in the string and returns the ptr string.

Ptr = strtok(NULL,” “); // it refers that it will store the next part of the string every time we call it.

**Address of operator 🡪 ‘&’** finds the address of the variable

Cout<<&x<<endl; // gives hexadecimal no.

It doesn’t work for char variables

(Void\*)&ch -- explicit typecasting from char\* to void\*

**Pointer 🡪**  it is a variable that stores address of another variable.

Int \*ptr – to declare pointer variable

Int \*ptr = &y – to declare and initialize

&bucket – address

\*address – bucket

\*(&x) – bucket

**Dereference 🡪** we can access the variable to which we are pointing

**Compile time /static memory allocation🡪**  in this memory for named variable is allocated by the compiler and exact size and type of storage must be known at compile time. It is for standard array declaration that is why the size has to be constant

**Run time / Dynamic memory allocation🡪** in this memory is allocated during run time. Dynamically allocated space usually placed in a program segment known as the heap or the free store. Exact amount of space or number of items does not have to be known by compiler in advance. In this pointer are crucial.

**Find function 🡪** for searching any element

It = find(arr, arr+n, key);

Index = it – arr;

**Binary search function 🡪** for searching any element and occurrence of element in a sorted array

Bool present = binary\_search(arr,arr+n,key);

**Lower bound function 🡪**  it will return the first element that will compare that key is >=key

auto lb = lower\_bound(arr, arr + n, 41);

**Upper bound function 🡪** it will return the first element that is strictly greater than key. >key

auto ub = upper\_bound(arr, arr + n, 41);

occurrences of element 🡪 ub-lb

**Sort function 🡪** Sort (starting point of a container a, ending point a + n)

**Rotate function 🡪** this function rotates the array going mid to right then coming left to mid.

Rotate(start, mid,end); rotate(arr,arr+2,arr+n)

**Next permutation function 🡪** for rearranging the elements of the array and getting lexicographically bigger number.

Next\_permutation(start,end);

**Swap function 🡪** for interchanging two numbers in an array.

Swap(a,b);

**Min function 🡪** for getting the smallest no in the container

Min(a,b);

**Max function 🡪** for getting biggest no in the container.

Max(a,b);

**Reverse function 🡪** for reversing the elements

Reverse(arr,arr+4/arr+n);

**Pair class🡪** for making pair of any type of two elements

Pair<int,char>p; // format of creating pair. Here p is the name of the pair

p.first = 10; // giving value to the pair

p.second = ‘b’;

pair<int,char> p2(p); // another format. Here p2 is also p means p2 is having the values of p

pair<int,string>p3 = make\_pair(100,’audi’); // here we used make pair function it makes the pair here only.

\*\* we can make array, vector of pairs. And we can also make pair of pair

Pair<pair<int,int>,string>p5

P5.second = “hi”;

P5.first.first = 10;

P5.first.second = 20;

Cout<<p5.first.first<<endl;

**Make pair function 🡪** it makes the pair

Make\_pair = (a,b);

**First occurrence in an array 🡪** when we are able to find the key as mid by binary search then we will go and search on left of mid for first occurrence

**last occurrence in an array 🡪** when we are able to find the key as mid by binary search then we will go and search on right of mid for last occurrence.

**Bitwise operators 🡪**

**AND &**  1 &1 = 1, 1&0 = 0, 0&1 = 0, 0&0 = 0

**OR |** 1|1 = 1 , 1|0 = 1, 0|1 = 1, 0|0 = 0

**NOT ~** ~1 = 0, ~ 0 = 1

**XOR ^** 1^1 = 0, 1^0 = 1, 0^1 = 1, 0^0 = 0

**Left shift operator <<** a<<1 = a\*2

**Right shift operator >>** a>>b = a/2b

for checking the given no is odd or even we do (a & 1) if first digit of this is 1 then it id odd if it is 0 then it is even.

**Find ith bit 🡪**

We do & with n and (1<<i) if this is greater than 0 then it is set bit it will return 1 otherwise 0.

Int mask = (1<<i)

Int bit = (n&mask) >0?1:0;

Return bit;

**Set ith bit 🡪**

Int mask = (1<<i);

Int bit = n|mask;

Return bit;

**Clear bit 🡪**

We have to create a mask where the ith bit should be 0 and all other are 1. Then we will flip all the bits.

Int mask = ~(1<<i);

N = n&mask;

**update bit 🡪**

first clear the ith bit. Then create a mask where ith bit is value and other are 0. Then we do OR of this

int mask= ~(1<<i);

int cleared\_n = n&mask;

n = cleared\_n|(v<<i);

**clear last i bits 🡪**

first we will take a no. then we create a mask of all zeroes and flip the mask. (~ 0). And then we will left shift the mask by i.

int mask = (-1<<i);

return n&mask;

**clear a range of bits from i to j 🡪**

we will create a mask having range 0 and other bits 1 from left and right.

We will take A which is having all 1 till jth position after that all zeroes.

Then we will take B which will start 1 from i+1th position to 0.

Then we will do OR of these two we will get the final mask.

Int A (left)= (~0)<<(j+1);

Int B (right)= (1<<i)-1;

Int Mask = A|B;

int ans = n & mask;

**Modulus properties**

(a+b)%m = ((a%m)+(b%m))%m

(a-b)%m = ((a%m)-b%m + m)%m

(a\*b)%m = ((a%m)\*(b%m))%m

(a/b)%m = ((a%m)\*(b-1 %m))%m -- b-1 – multiplicate modulo inverse.