**Reading Input from File**

It is very easy to read inputs from a file using C++.

**Program to read input from a file, then save the output in another file.**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

ifstream f("data.in");

ofstream g("data.out");

int main()

{

int a, b, sum;

f>>a>>b;

sum = a+b;

g<<sum;

return 0;

}

**Input File-** data.in



**Output File-** data.out



**What is STL?**

**STL** (**S**imple **T**emplate **L**ibrary) is a set of templates used to make the code **Simple** and **Easy to Write**.

**How simple?**

**Example of sorting an array:**

|  |  |
| --- | --- |
| **without STL** | **With STL** |
| void MergeSort(int st, int dr){  if(st < dr){  int m = st + rand () % (dr - st + 1);  MergeSort(st, m);  MergeSort(m + 1, dr);  int i = st, j = m + 1, k = 0;  while(i <= m && j <= dr){  if(v[i] < v[j])  tmp[++k] = v[i++];  else  tmp[++k] = v[j++];  }  while(i <= m){  tmp[++k] = v[i++];  }  while(j <= dr){  tmp[++k] = v[j++];  }  for(i = st, j = 1, i <= dr; i++, j++){  v[i] = tmp[j];  }  }  }  sort(a+n, a+n+1); |  |

**Example of swapping 2 values-**

|  |  |
| --- | --- |
| **without STL** | **With STL** |
| aux = a;  a = b;  b = aux;  swap(a, b); |  |

**Example of finding maximum and minimum of 2 values:**

|  |  |
| --- | --- |
| **without STL** | **With STL** |
| if(a > b){  maximum = a;  minimum = b;  }  else  maximum = b;  minimum = a;  }  maximum = max(a, b);  minimum = min(a, b); |  |

**How to use STL?**

By just including the library **#include <bits/stdc++.h>**.

**Example-**

|  |  |
| --- | --- |
| **Program** | **Output** |
| #include <bits/stdc++.h>  using namespace std;  int main(){  int a = 5, b = 8;  cout<<"Maximum = "<<max(a, b);  cout<<"\nMinimum = "<<min(a, b);  swap(a, b);  cout<<"\na = "<<a<<"\tb = "<<b<<'\n';  int number = 2;  double cubicRoot;  cubicRoot = pow((double) (number), 1.0/3);  cout<<cubicRoot<<'\n';  cout<<fixed<<setprecision(10)<<cubicRoot<<'\n';  cout<<fixed<<setprecision(3)<<cubicRoot<<'\n';  return 0;  }  Maximum = 8  Minimum = 5  a = 8 b = 5  1.25992  1.2599210499  1.260 |  |

**Global and Local Variables**

**Global variables:**

1. Defined on top of the program.
2. Initialized with 0.
3. Are accessible anywhere in the program.

**Local variables:**

1. Defined inside functions.
2. Initialized randomly.
3. Accessible just in the function.

|  |  |
| --- | --- |
| **Program** | **Output** |
| #include <iostream>  using namespace std;  int Aglobal[3];  int main(){  int Alocal[3];  cout<<"Aglobal is: "<<Aglobal[0]<<" "<<Aglobal[1]<<" "<<Aglobal[2]<<"\n";  cout<<"Alocal is: "<<Alocal[0]<<" "<<Alocal[1]<<" "<<Alocal[2]<<"\n";  return 0;  }  Aglobal is: 0 0 0  Alocal is: 0 16 0 |  |

|  |  |
| --- | --- |
| **Program** | **Output** |
| #include <iostream>  using namespace std;  int Aglobal[3];  void Printing(){  cout<<a;  }  int main(){  int Alocal[3];  int a;  a = 5;  Printing();  return 0;  }  Error: ‘a’ was not declared in this scope. |  |

|  |  |
| --- | --- |
| **Program** | **Output** |
| #include<iostream>  using namespace std;  int a;  void Printing(){  cout<<a<<'\n';  int a = 5;  cout<<a<<'\n';  }  int main(){  a = 100;  Printing();  int a = 60;  cout<<a;  return 0;  }  100  50 |  |

|  |  |
| --- | --- |
| **Programs** | **output** |
| #include<iostream>  using namespace std;  int a;  int X;  void Printing (int X){  cout<<X;  }  int main(){  a = 100;  Printing(a);  cout<<X;  return 0;  }  1000 |  |

|  |  |
| --- | --- |
| **Program** | **Output** |
| #include<bits/stdc++.h>  using namespace std;  int A[3][5], i;  void printingRow(int row){  for(i=0; i<5; i++){  cout<<A[row][i]<<" ";  }  cout<<'\n';  }  int main(){  for(i=0; i<3; i++){  printingRow(i);  }  return 0;  }  0 0 0 0 0 |  |

|  |  |
| --- | --- |
| **Program** | **Output** |
| #include<bits/stdc++.h>  using namespace std;  int A[3][5], i;  void printingRow(int row){  for(i=0; i<5; i++){  cout<<A[row][i]<<" ";  }  cout<<'\n';  }  int main(){  for(i=0; i<3; i++){  int B = 3;  printingRow(i);  }  cout<<B;  return 0;  }  error: ‘B’ was not declared in this scope |  |

**A Block of code** is the code between two braces + (header)

**Example-**

|  |  |  |
| --- | --- | --- |
| for(){    }  void printing(int a){  }  int main(){    } |  |  |

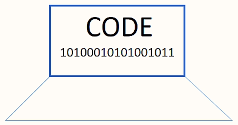
|  |  |
| --- | --- |
| **Program** | **Output** |
| #include<iostream>  using namespace std;  int Aglobal[3][3];  void addMatrix(int X){  for(int i=0; i<3; ++i){  for(int j=0; j<3; j++){  Aglobal[i][j] += X;  }  }  }  int sumMatrix(){  int sumElements = 0;  for(int i=0; i<3; i++){  for(int j=0; j<3; j++){  sumElements += Aglobal[i][j];  }  }  return sumElements;  }  int main(){  addMatrix(1);  addMatrix(10);  addMatrix(100);  int sumElements = sumMatrix();  cout<<sumElements;  return 0;  }  999 |  |

**How to analyse time complexity?**

**Running time depends upon:**

1. Single vs multiple processor.
2. Read/Write speed of memory.
3. 32-bit vs 64-bit.
4. Input.

**Model Machine**



1 unit of time for arithmetical and logical operations

1 unit of time for assignment and return

**Example-**

|  |  |
| --- | --- |
| **Program** | **Unit Time** |
| Sum(a, b){  return a+b;  }  2 |  |

|  |  |  |
| --- | --- | --- |
| **Code** | **Cost** | **Number of Times** |
| SumOfArray(A[], n){  Total = 0;  for(i=1; i<=n; i++){  Total = Total + A[i];  }  return Total;  }  Time Complexity = 1\*1 + 2\*n + 2\*n + 1\*1  Time Complexity = 4\*n + 2  1  2  2  1  1  n  n  1 |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Code**  SumOfArray(A[], n){  Total = 0;  for(i=1; i<=n; i++){  for (j=1; j<n; j++){  Total = Total + A[i][j];  }  }  return Total;  }  Time Complexity = 1 + 2\*n + 2\*n2 + 2\*n2 + 1  Time Complexity = 4\*n2 + 2\*n + 2  1  2  2  2  1  1  n  n\*n  n\*n  1 | **Cost** | **Number of Times** |
|  |  |  |

**We analyse complexity for:**

1. Worst case scenario.
2. Very large input size.

T(n) = n3 + 5n2 + 9n + 7 ≈ n3

T(n)= 5n2 + 3n + 7 ≈ n2

T(n)= 17n2 + 2n + 9 ≈ n2

T(n)= 2n + 998 ≈ n

N → ∞

**Big O Notation**

**Rules for Big O Notation**-

1. Drop lower order terms.  
   2. Drop any constant multiplier.

T(n) = 5n + 7 ≈ O(n)

T(n) = n4 + 3n3 + n2 + 9998 ≈ O(n4)

**Rule:** Running time = sum of all code fragments.

|  |  |  |
| --- | --- | --- |
| for(int i=7; i<=n; ++i){  //simple statement  }  for(int i; i<=n; ++i){  for(j=1; j<=n; ++j){  //simple statement  }  }  for(int i; i<=n; ++i){  for(j=1; j<=n; ++j){  //simple statement  }  }  **Simple Statements**  **O(1)** | **Single Loop**  **O(n)** | **Nested Loop**  **O(n2)** |

function(){

int a;

a = 5;

a = a + 6;

++a;

for(int i=7; i<=n; ++i){

//simple statement

}

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

for(j=1; j<=n; ++j){

//simple statement

}

}

}

O(1)

O(n)

O(n2)

T(n) = O(n2) + O(n) + O(1)

T(n) = O(n2)

**Appearance Array**

|  |  |
| --- | --- |
| **Program** | **Result** |
| #include<bits/stdc++.h>  using namespace std;  ifstream f("data.in");  ofstream g("data.out");  int main(){  int i, n, element, elementCount;  f>>n;  int arr[n];  for(i=0; i<=n-1; i++){  f>>arr[i];  }  f>>element;  elementCount = count(arr, arr+n, element);  if(elementCount > 0){  g<<"Element is present "<<elementCount<<" times.";  }else{  g<<"Element not present.";  }  return 0;  }  10  1 7 9 2 7 3 10 7 9 4  7  Element is present 3 times. | **data.in**  **data.out** |

**STACK**

A stack is a list with the restriction that insertion and deletion can be performed only from one end, called the top of the stack.

A stack is also called a **LIFO Data Structure**(Last In, First Out), which means the last element that enters the stack will be the first element to be removed from the stack.

There are 4 operations that can be performed on a stack-  
1. **push(x)** - It pushes an element (x) at the top of the stack.

1. **pop() -** It removes an element at the top of the stack.
2. **isEmpty() -** Returns a boolean value whether the stack is empty or not.
3. **top() -** Returns the element at the top of the stack.

**Example-**

|  |  |
| --- | --- |
| **Program** | **Output** |
| #include<bits/stdc++.h>  using namespace std;  int Stack[100], ind;  void push(int x){  ++ind;  Stack[ind] = x;  }  bool isEmpty(){  if(ind >= 1)  return false;  else  return true;  }  void pop(){  Stack[ind] = 0;  --ind;  }  int top(){  return Stack[ind];  }  int main(){  ind = 0;  push(1);  push(2);  if(!isEmpty())  cout<<top();  pop();  pop();  return 0;  }  2 |  |

**QUEUE**

A queue is a list with the restriction that insertion can be performed from one end (called back) and deletion can be performed from the other end (called front).

A queue is also called a **FIFO Data Structure**(First In, First Out), which means the first element that enters the queue will be the first element to be removed from the queue.

There are 4 operations that can be performed on a queue-  
1. **push(x)** - It pushes an element (x) at the back of the queue.

1. **pop() -** It removes an element at the front of the queue.
2. **isEmpty() -** Returns a boolean value whether the queue is empty or not.
3. **front() -** Returns the element at the front of the queue.

|  |  |
| --- | --- |
| **Program** | **Output** |
| #include<bits/stdc++.h>  using namespace std;  int backInd = -1;  int frontInd = 0;  int Queue[100];  void push(int x){  ++backInd;  Queue[backInd] = x;  }  void pop(){  Queue[frontInd] = 0;  ++frontInd;  }  bool isEmpty(){  if(backInd < frontInd)  return true;  else  return false;  }  int Front(){  return Queue[frontInd];  }  int main(){  push(1);  push(2);  if(!isEmpty())  cout<<Front();  pop();  pop();  return 0;  }  1 |  |

**Complexity?**

Let n = 100.000

Complexity of Linear search → O(n) → 100,000

Complexity of Binary search → O(log n) → 18

thus 0.02% of the time