Optional Hw

**Input and Output**

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

int n1,n2,n3;

cin>>n1>>n2>>n3;

cout<<n1+n2+n3<<endl;

return 0;

}

# Basic Data Types

#include <iostream>

#include <cstdio>

using namespace std;

int main() {

int in;

string num[10] = {"Greater than 9", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"};

cin >> in;

if(in > 9){

cout << num[0];

}

else{

cout << num[in];

}

return 0;

}

# Conditional Statements

#include <bits/stdc++.h>

using namespace std;

int main() {

int in;

string num[10] = {"Greater than 9", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"};

cin >> in;

if(in > 9){

cout << num[0];

}

else{

cout << num[in];

}

return 0;

}

# For Loop

#include <iostream>

#include <cstdio>

using namespace std;

int main() {

int in,in1;

string num[10] = {"Greater than 9", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"};

cin >> in>>in1;

int i;

for (i=in;i<=in1;i++){

if(i > 9){

if (i%2==0){

cout <<"even"<<endl;}

else{

cout <<"odd"<<endl;

}

}

else{

cout << num[i]<<endl;

}

}

return 0;

}

# Functions

#include <iostream>

#include <cstdio>

using namespace std;

/\*

Add `int max\_of\_four(int a, int b, int c, int d)` here.

\*/

int max\_of\_four(int a, int b, int c, int d){

return (a>b && a>c && a>d) ? a

: (b>c && b>d) ? b

: (c>d) ? c

: d;

}

int main() {

int a, b, c, d;

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

int ans = max\_of\_four(a, b, c, d);

printf("%d", ans);

return 0;

}

# Pointer

#include <stdio.h>

void update(int \*a,int \*b) {

int c =\*a;

\*a=\*a+\*b;

\*b= \*b>c ?(\*b-c):(c-\*b);

// Complete this function

}

int main() {

int a, b;

int \*pa = &a, \*pb = &b;

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

update(pa, pb);

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

return 0;

}

# Arrays Introduction

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

int N,i,j;

cin >>N;

if (N>1 & N<=1000){

int arra [N];

for (i=0;i<N;i++){

cin>>arra[i];

}

for(j=(sizeof(arra)/sizeof(\*arra))-1 ;j>=0;j--){

cout<<arra[j]<<" ";

}

}

return 0;

}

# Variable Sized Arrays

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int n,q;

cin >> n >> q;

vector< vector<int> > a(n);

// input each array

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

int k;

cin >> k;

for (int j=0;j<k;j++) {

int data;

cin >> data;

a[i].push\_back(data);

}

}

// do the queries

for (int i=0;i<q;i++) {

int x,y;

cin >> x >> y;

cout << a[x][y] << endl;

}

return 0;

}

# Attribute Parser

#include <cstdio>

#include <iostream>

#include <sstream>

#include <map>

using namespace std;

struct tagobj {

string name;

map<string, tagobj\*> tmap;

map<string, string> map\_att;

tagobj\* prev = NULL;

};

void delete\_tags(tagobj\* tag) {

for (map<string, tagobj\*>::iterator i = tag->tmap.begin() ; i != tag->tmap.end() ; i ++ ) {

delete\_tags(i->second);

}

delete(tag);

}

int main(){

//Setup a vector holding all first level tags

map<string, tagobj\*> main;

//Current depth so as to nest

tagobj\* masterTag = NULL;

int tlines, qlines;

string ints;

getline(cin, ints);

stringstream ss(ints);

ss >> tlines;

ss >> qlines;

string line;

size\_t del\_tag;

//Save the information

for(int i=0; i<tlines; i++) {

//Get Tag from line

getline(cin, line);

del\_tag = line.find\_first\_of(" >");

string tag = line.substr(1,del\_tag-1);

//opening tags

if(tag[0] != '/') {

//Create Struct

tagobj\* currTag = new tagobj();

currTag->prev = masterTag;

currTag->name = tag;

//check if we add to master vector or previous:

if(!masterTag) {

main[currTag->name] = currTag;

} else {

masterTag->tmap[currTag->name] = currTag;

}

masterTag = currTag;

//Add all attributes

size\_t found = line.find("=", del\_tag);

while(found != string::npos) {

//Get the attribute

string name = line.substr(del\_tag + 1, found-del\_tag-2);

//move past the =

del\_tag = found + 2;

found = line.find("\"", del\_tag + 1);

//Copy the value

string value = line.substr(del\_tag + 1,found - del\_tag - 1);

//add to map

currTag->map\_att[name]=value;

//keep going

del\_tag = line.find(" ", found);

found = line.find("=", found);

}

} else {

//If its a closing Tag return main tag to its previous

masterTag = masterTag->prev;

}

}

for(int i=0; i < qlines; i++) {

tagobj\* masterq = NULL;

getline(cin,line);

size\_t start = 0;

string val;

bool fnd;

while (true) {

size\_t found = line.find\_first\_of(".~", start);

string buff = line.substr(start,found-start);

fnd = false;

//looking for attribute

if(start > 0 && line[start-1] == '~') {

//Attributes have a sentinal

//Check for bad input

if(masterq) {

val = masterq->map\_att[buff];

if(!val.empty()){

fnd = true;

}

break;

}

} else {

//Looking for a tag, if found, continue looking

map<string, tagobj\*> \*bufobj = NULL;

//Make sure to look in right place

if(masterq) {

bufobj = &(masterq->tmap);

} else {

bufobj = &main;

}

tagobj\* frommap = (\*bufobj)[buff];

if(frommap) {

masterq = frommap;

} else {

break;

}

}

start = found + 1;

}

//Printing Time:

if(fnd) {

cout << val << endl;

} else {

cout << "Not Found!" << endl;

}

}

for (map<string, tagobj\*>::iterator i = main.begin() ; i != main.end() ; i ++ ) {

delete\_tags(i->second);

}

}

# StringStream

#include <sstream>

#include <vector>

#include <iostream>

using namespace std;

vector<int> parseInts(string str) {

stringstream ss(str); //??

vector<int> result;

char ch;

int tmp;

while(ss >> tmp) { //??

result.push\_back(tmp);

ss >> ch; //??

}

return result;}

int main() {

string str;

cin >> str;

vector<int> integers = parseInts(str);

for(int i = 0; i < integers.size(); i++) {

cout << integers[i] << "\n";

}

return 0;

}

# Structs

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

struct Student

{

unsigned age{};

std::string first\_name{};

std::string last\_name{};

unsigned standard{};

};

int main() {

Student st;

cin >> st.age >> st.first\_name >> st.last\_name >> st.standard;

cout << st.age << " " << st.first\_name << " " << st.last\_name << " " << st.standard;

return 0;

}

# Class

#include <iostream>

#include <sstream>

using namespace std;

class Student{

int age;

int standard;

string first\_name;

string last\_name;

public:

Student()

{

age = 0;

standard = 0;

first\_name.clear();

last\_name.clear();

}

void set\_age(int newAge)

{

age = newAge;

}

void set\_standard(int newStandard)

{

standard = newStandard;

}

void set\_first\_name(string newFirst\_name)

{

first\_name = newFirst\_name;

}

void set\_last\_name(string newLast\_name)

{

last\_name = newLast\_name;

}

int get\_age() {return age;}

int get\_standard() {return standard;}

string get\_first\_name() {return first\_name;}

string get\_last\_name() {return last\_name;}

string to\_string()

{

stringstream ss;

char c = ',';

ss << age << c << first\_name << c << last\_name << c << standard;

return ss.str();

}

};

int main() {

int age, standard;

string first\_name, last\_name;

cin >> age >> first\_name >> last\_name >> standard;

Student st;

st.set\_age(age);

st.set\_standard(standard);

st.set\_first\_name(first\_name);

st.set\_last\_name(last\_name);

cout << st.get\_age() << "\n";

cout << st.get\_last\_name() << ", " << st.get\_first\_name() << "\n";

cout << st.get\_standard() << "\n";

cout << "\n";

cout << st.to\_string();

return 0;

}

# Classes and Objects

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

#include <cassert>

using namespace std;

class Student {

private:

int scores[5];

int sum;

public:

Student() : sum(0) {}

int calculateTotalScore() {return sum;}

void input() {

for(int i=0; i<5; i++) {

cin >> scores[i];

sum+=scores[i];

}

}

};

// Write your Student class here

int main() {

int n; // number of students

cin >> n;

Student \*s = new Student[n]; // an array of n students

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

s[i].input();

}

// calculate kristen's score

int kristen\_score = s[0].calculateTotalScore();

// determine how many students scored higher than kristen

int count = 0;

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

int total = s[i].calculateTotalScore();

if(total > kristen\_score){

count++;

}

}

// print result

cout << count;

return 0;

}

# Box It!

#include<bits/stdc++.h>

using namespace std;

class Box{

private:

int l, b, h;

public:

Box(){

l = 0;

b = 0;

h = 0;

}

Box(int length, int breadth, int height){

l = length;

b = breadth;

h = height;

}

Box(const Box& B){

l = B.l;

b = B.b;

h = B.h;

}

int getLenght(){

return l;

}

int getBreadth(){

return b;

}

int getHeight(){

return h;

}

long long CalculateVolume(){

return (long long)l\*b\*h;

}

friend bool operator < ( Box&A,Box& B){

if( (A.l < B.l) || ((A.b < B.b) && (A.l == B.l)) || ((A.h < B.h) && (A.l == B.l) && (A.b == B.b)) ){

return true;

}else{

return false;

}

};

friend ostream& operator<< (ostream& output, const Box& B){

output << B.l << " " << B.b << " " << B.h;

return output;

}

};