// 1. Viết định nghĩa lớp biểu diễn khái niệm thời gian với các thành phần dữ liệu giờ, phút, giây với các thao tác thích hợp.

#include <bits/stdc++.h>

using *namespace* std;

*class* *Time* {

*private:*

*int* hours, minutes, seconds;

*public:*

        Time(){

            hours = minutes = seconds = 0;

        }

        Time(const *Time*& *x*){

            hours = *x*.hours;

            minutes = *x*.minutes;

            seconds = *x*.seconds;

        }

        ~Time(){}

*int* getHours(){

            return this->hours;

        }

*int* getMinutes(){

            return this->minutes;

        }

*int* getSeconds(){

            return this->seconds;

        }

*void* setHours(*int* *h*){

            this->hours = *h*;

        }

*void* setMinutes(*int* *m*){

            this->minutes = *m*;

        }

*void* setSeconds(*int* *s*){

            this->seconds = *s*;

        }

        friend *istream*& operator >> (*istream*& *in*, *Time*& *x*){

*in* >> *x*.hours >> *x*. minutes >> *x*.seconds;

            return *in*;

        }

        friend *ostream*& operator << (*ostream*& *out*, *Time* *x*){

*out* << *x*.hours << " " << *x*.minutes << " " << *x*.seconds;

            return *out*;

        }

*bool* operator > (*Time* *x*) {

            return ChangeUnits(\*this) > ChangeUnits(*x*);

        }

*bool* operator < (*Time* *x*) {

            return ChangeUnits(\*this) < ChangeUnits(*x*);

        }

*bool* operator == (*Time* *x*) {

            return ChangeUnits(\*this) == ChangeUnits(*x*);

        }

*int* ChangeUnits(*Time* *x*){

*int* newHou = *x*.hours\*3600;

*int* newMin = *x*.minutes\*60;

*int* Change = *x*.seconds + newHou + newMin;

            return Change;

        }

};

*void* Plus(*Time* *x*, *Time* *y*){

*int* sumHours = *x*.getHours() + *y*.getHours();

*int* sumMinutes = *x*.getMinutes() + *y*.getMinutes();

*int* sumSeconds = *x*.getSeconds() + *y*.getSeconds();

    if(sumSeconds >= 60){

        sumSeconds = sumSeconds - 60;

        sumMinutes++;

    }

    if(sumMinutes >= 60){

        sumMinutes = sumMinutes - 60;

        sumHours++;

    }

    if(sumHours >= 24) sumHours = sumHours - 24;

    cout << "Addition results: " << sumHours << " " << sumMinutes << " " << sumSeconds << endl;

}

*void* Minus(*Time* *x*, *Time* *y*){

*int* newsec = *x*.getSeconds(), newmin = *x*.getMinutes(), newhours = *x*.getHours();

    if(*x*.getSeconds() < *y*.getSeconds()){

        newmin--;

        newsec += 60;

*x*.setMinutes(newmin);

*x*.setSeconds(newsec);

    }

    if(*x*.getMinutes() < *y*.getMinutes()){

        newhours--;

        newmin += 60;

*x*.setHours(newhours);

*x*.setMinutes(newmin);

    }

*int* minusHours = *x*.getHours() - *y*.getHours();

*int* minusMinutes = *x*.getMinutes() - *y*.getMinutes();

*int* minusSeconds = *x*.getSeconds() - *y*.getSeconds();

    if(minusHours < 0) minusHours = 0;

    if(minusMinutes < 0) minusMinutes = 0;

    if(minusSeconds < 0) minusSeconds = 0;

    cout << "Subtraction results: " << minusHours << " " << minusMinutes << " " << minusSeconds;

}

*void* ChangeTime(*Time*& *x*, *Time* *NewTime*){

*x*.setHours(*NewTime*.getHours());

*x*.setMinutes(*NewTime*.getMinutes());

*x*.setSeconds(*NewTime*.getSeconds());

}

*bool* Timeaccept(*Time* *x*){

    if(*x*.getHours() < 0 || *x*.getHours() >= 24) return true;

    if(*x*.getMinutes() < 0 || *x*.getMinutes() >= 60) return true;

    if(*x*.getSeconds() < 0 || *x*.getSeconds() >= 60) return true;

    return false;

}

*int* main(){

*Time* x, y;

    cout << "Enter the first time: ";

    do{

        cin >> x;

    } while(Timeaccept(x));

    cout << "Enter the second time: ";

    do{

        cin >> y;

    } while(Timeaccept(y));

*int* n;

    do{

        cout << "What is your choice: " << endl;

        cout << "1: Plus " << endl;

        cout << "2: Minus" << endl;

        cout << "3: Change Time" << endl;

        cout << "4: Compare" << endl;

        cout << "0: Exit" << endl;

        cin >> n;

        switch (n)

        {

        case 1:

            Plus(x, y);

            break;

        case 2:

            Minus(x, y);

            break;

        case 3:

        {

*Time* New; *int* m;

            cout << "The time you want to change: ";

            cin >> New;

            cout << "Choose time to change: " << endl;

            cout << "1: First time" << endl;

            cout << "2: Second time" << endl;

            cin >> m;

            switch (m)

            {

            case 1:

                ChangeTime(x, New);

                cout << "New frist time: " << x << endl;

                break;

            case 2:

                ChangeTime(y, New);

                cout << "New second time: " << y << endl;

                break;

            }

        }

            break;

        case 4:

        {

            if(x < y) cout << "The first time is smaller!" << endl;

            else if (x > y) cout << "The second time is smaller!" << endl;

            else cout << "Two equal times!" << endl;

        }

            break;

        default:

            cout << "Exit!";

            break;

        }

    } while(n != 0);

    return 0;

}

// 2. Viết định nghĩa lớp Stack để biểu diễn khái niệm một Stack các số nguyên với thao tác tương ứng.

#include <bits/stdc++.h>

using *namespace* std;

*class* *Stack* {

*private:*

*int* maxSize, currentSize;

*int*\* arr;

*public:*

        Stack(){

            currentSize = -1;

        }

*void* importSize(){

            cout << "Enter max size of the stack: ";

            cin >> maxSize;

            arr = new *int*[maxSize];

        }

*void* output(){

            if(!isEmpty()){

                for(*int* i = currentSize; i>=0; i--){

                    cout << arr[i] << " ";

                }

            }

            else cout << "Empty stack!" << endl;

        }

*void* convert(){

            for(*int* i = currentSize; i>=0; i--){

                if(arr[i] < 10) cout << arr[i];

                else cout << *char*(arr[i] + 55);

            }

        }

*void* deleteStack(){

            currentSize = -1;

        }

*bool* isEmpty(){

            if(currentSize == -1) return true;

            return false;

        }

*bool* isFull(){

            if(currentSize == maxSize - 1) return true;

            return false;

        }

*void* Push(*int* *element*){

            if(!isFull()){

                currentSize++;

                arr[currentSize] = *element*;

            }

            else cout << "Full stack!" << endl;

        }

*void* Pop(){

            if(!isEmpty()){

                cout << "The retrieved element is: " << arr[currentSize] << endl;

                currentSize--;

            }

            else cout << "Empty stack!" << endl;

        }

*void* getvalatTop(){

            if(!isEmpty()){

                cout << arr[currentSize];

            }

            else cout << "Empty stack!" << endl;

        }

};

*void* addElement(*Stack*& *stack*){

*int* element;

    cout << "Enter the element you want to add: " << endl;

    cin >> element;

*stack*.Push(element);

}

// 3. Viết chương trình phân tích một số thành thừa số nguyên tố rồi in ra theo thứ tự ngược sử dụng Stack ở câu trên.

// Ví dụ:

// Nhập vào: 750

// In ra: 750 = 5 \* 5 \* 5 \* 3 \* 2

*bool* isPrime(*int* *x*) {

    if (*x* < 2)

        return false;

    for (*int* i = 2; i\*i <= *x*; i++)

        if (*x* % i == 0)

            return false;

    return true;

}

*void* Primefactors(*int* *n*, *Stack*& *stack*){

*int* origin = *n*;

    while(*n* != 1){

*int* i;

        for(i = 2; i<=origin ; i++){

            if(*n*%i == 0 && isPrime(i)){

*stack*.Push(i);

                break;

            }

        }

*n* /= i;

    }

}

// 4. Viết chương trình đổi một số sang hệ thập lục phân, hệ bát phân, hệ nhị phân sử dụng Stack ở câu trên.

*void* Hexadecimalsystem(*int* *n*, *Stack*& *stack*){

    if(*n* == 0) cout << 0;

    else {

        while(*n* != 0){

*stack*.Push(*n*%16);

*n* = *n*/16;

        }

    }

}

*void* Octalsystem(*int* *n*, *Stack*& *stack*){

    if(*n* == 0) cout << 0;

    else {

        while(*n* != 0){

*stack*.Push(*n*%8);

*n* = *n*/8;

        }

    }

}

*void* Binarysystem(*int* *n*, *Stack*& *stack*){

    if(*n* == 0) cout << 0;

    else {

        while(*n* != 0){

*stack*.Push(*n*%2);

*n* = *n*/2;

        }

    }

}

*int* main(){

*int* choice;

*Stack* stack;

    stack.importSize();

    do{

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

    cout << "0. Exit!" << endl;

    cout << "1. Push (them phan tu tren dau ngan xep)" << endl;

    cout << "2. Pop (lay ra phan tu tren dau ngan xep" << endl;

    cout << "3. In ra phan tu tren dau ngan xep" << endl;

    cout << "4. In ra ngan xep hien tai" << endl;

    cout << "5. Xoa ngan xep da tao" << endl;

    cout << "6. Phan tich thua so nguyen to" << endl;

    cout << "7. Doi sang he thap luc phan" << endl;

    cout << "8. Doi sang he bat phan" << endl;

    cout << "9. Doi sang he nhi phan" << endl;

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

    cin >> choice;

        switch (choice)

        {

        case 0:

            cout << "Exited!";

            break;

        case 1:

            addElement(stack);

            break;

        case 2:

            stack.Pop();

            break;

        case 3:

            stack.getvalatTop();

            break;

        case 4:

            stack.output();

            break;

        case 5:

            stack.deleteStack();

            break;

        case 6:

        {

*int* n;

            cout << "Enter the number you want to factorize into prime factors: ";

            cin >> n;

            Primefactors(n, stack);

            stack.output();

            stack.deleteStack();

            cout << endl;

        }

            break;

        case 7:

        {

*int* n;

            cout << "Enter the number you want to convert to hexadecimal: ";

            cin >> n;

            Hexadecimalsystem(n, stack);

            stack.convert();

            cout << endl;

            stack.deleteStack();

            cout << endl;

        }

            break;

        case 8:

        {

*int* n;

            cout << "Enter the number you want to convert to octal: ";

            cin >> n;

            Octalsystem(n, stack);

            stack.convert();

            cout << endl;

            stack.deleteStack();

            cout << endl;

        }

            break;

        case 9:

        {

*int* n;

            cout << "Enter the number you want to convert to binary: ";

            cin >> n;

            Binarysystem(n, stack);

            stack.convert();

            cout << endl;

            stack.deleteStack();

            cout << endl;

        }

        default:

            cout << "Invalid selection!" << endl;

            break;

        }

    } while(choice != 0);

    return 0;

}

// 5. Viết định nghĩa lớp Queue để biểu diễn khái niệm hàng đợi các số nguyên với thao tác tương ứng.

#include <bits/stdc++.h>

using *namespace* std;

*class* *Queue* {

*private:*

*int* size;

*int* \*arr;

*int* head, tail;

*public:*

        Queue(){

            this->head = -1;

            this->tail = -1;

        }

*void* input(){

            cout << "Input size of queue: ";

            cin >> size;

            arr = new *int*[size];

        }

*void* output(){

            if(isEmpty()) cout << "Empty Queue!" << endl;

            else{

                for(*int* i = head; i<=tail; i++){

                    cout << arr[i] << " ";

                }

                cout << endl;

            }

        }

*void* EnQueue(*int* *n*){

            if(head == -1)

                head++;

            tail++;

            arr[tail] = *n*;

        }

*void* DeQueue(){

            cout << "The element at the head of the queue to be deleted is: " << arr[head] << endl;

            head++;

        }

*bool* isEmpty(){

            if(head == -1 && tail == -1) return true;

            return false;

        }

*bool* isFull(){

            if(tail - head + 1 == size) return true;

            return false;

        }

*void* deleteQueue(){

            head = tail = -1;

            cout << "Queue has been deleted!" << endl;

        }

*int* getcurrentsize(){

            return tail - head + 1;

        }

};

*void* addelement(*Queue*& *queue*){

    if(*queue*.isFull()) cout << "Full Queue!" << endl;

    else{

        cout << "The element you want to add to the queue is: ";

*int* n; cin >> n;

*queue*.EnQueue(n);

    }

}

*void* getsize(*Queue*& *queue*){

    if(*queue*.isEmpty()) cout << "Empty Queue!" << endl;

    else{

        cout << "Current Queue is: " << *queue*.getcurrentsize() << endl;

    }

}

*int* main(){

*Queue* queue;

    queue.input();

*int* m;

    do{

    cout << "=======================================" << endl;

    cout << "0. Exit" << endl;

    cout << "1. EnQueue (them 1 phan tu vao cuoi hang doi)" << endl;

    cout << "2. DeQueue (xoa phan tu o dau hang doi)" << endl;

    cout << "3. In ra hang doi hien tai" << endl;

    cout << "4. In ra so phan tu trong hang doi" << endl;

    cout << "5. Xoa hang doi hien tai" << endl;

    cout << "=======================================" << endl;

    cin >> m;

    switch (m)

        {

        case 1:

            addelement(queue);

            break;

        case 2:

            queue.DeQueue();

            break;

        case 3:

            queue.output();

            break;

        case 4:

            getsize(queue);

            break;

        case 5:

            queue.deleteQueue();

            break;

        default:

            cout << "Exit!";

            break;

        }

    }while(m != 0);

    return 0;

}