# PRELAB

## **1.Function**

### [Q 1.2] Factorial

**[Tiếng Việt]**

Hiện thực hàm tính giai thừa của số N, sau đó gọi hàm vừa hiện thực trong hàm main để gán kết quả tính được vào biến result.

**Đầu vào:**

int N: số tự nhiên N

**[English]**

Implement a function that calculates the factorials of N. Then call that function inside the main function to assign the calculated value to the variable result.

**Input:**

* int N: a natural number N

**Template:**

#include <iostream>

using namespace std;  
// implement calculate factorial function in here  
# TODO  
  
int main(int narg, char\*\* argv)  
{  
    int N;  
    cin >> N;  
    long result;  
    // call function calculate factorial in here and assign value to the variable result  
    # TODO

    cout << result << endl;  
    return 0;  
}

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 5 | 120 |

### Answer q1.2

#include <iostream>

using namespace std;

// implement calculate factorial function in here

int calculateFactorial(int n)

{

if(n==0) return 1;

else return n\*calculateFactorial(n-1);

}

int main(int narg, char\*\* argv)

{

int N;

cin >> N;

long result;

// call function calculateFactorial in here and assign value to the variable result

result=calculateFactorial(N);

cout << result << endl;

return 0;

}

### [Q 1.3] Find max

**[Tiếng Việt]**

Viết hàm findMax để tìm giá trị lớn nhất trong mảng số nguyên.

**Đầu vào:**

int\* vals: mảng số nguyên

int numEls: số phần tử trong mảng vals

**Đầu ra:**

int: giá trị lớn nhất có trong mảng vals

**[English]**

Write function findMax to find the maximum of an input integer array.

**Input:**

int\* vals: array of integers

int numEls: the size of the array.

**Output:**

int: the maximum value in the array vals

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 10  -11 111 -1111 -112 -101 11 -19 1 145 12 | 145 |

int findMax(int \*vals, int numEls) {

// TODO

}

### Answer q1.3

int findMax(int \*vals, int numEls)

{

// TODO

int max=\*vals;

for(int i=1; i<numEls; i++)

{

if(\*(vals+i) >= max) max=\*(vals+i);

}

return max;

}

### [Q 1.4] Sum of elements

**[Tiếng Việt]**

Viết hàm sum2 để tính tổng giá trị các phần tử trong mảng số nguyên.

**Tham số:**

* int\* array: mảng số nguyên
* int size: số phần tử trong mảng
* int& result: tham số để lưu kết quả cuối cùng sau khi tính toán

**[English]**

Write the function sum2 that calculates the total of all elements in an integer array

**Parameters:**

* int\* array: an array of integers
* int size: the number of elements in the array
* int& result: a parameter to return the calculated value to the caller

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 10  -11 111 -1111 -112 -101 11 -19 1 145 12 | -1074 |

void sum2(int \* array, int numEls, int &result)

{

// TODO

}

### Answer q1.4

void sum2(int \* array, int numEls, int &result)

{

// TODO

result=0;

for(int i=0; i<numEls; i++)

{

result=result + \*(array+i);

}

}

### [Q 1.1] Check complete number

**[Tiếng Việt]**

Viết hàm bool completeNum(int N) để kiểm tra xem số nguyên dương N có phải là một số hoàn thiện hay không. N là một số hoàn thiện nếu N bằng tổng tất cả ước số nguyên dương (không bao gồm chính nó) của nó.

**Đầu vào:**

* int N: số nguyên dương N cần kiểm tra

**Đầu ra:**

* bool: trả về true nếu N là số hoàn thiện, ngược lại trả về false

**[English]**

Write the function bool completeNum(int N) that checks if a positive integer N is a complete number. N is a complete number if and only if N is equal to the sum of all of its positive divisors (excluding itself)

**Input:**

* int N: positive integer N to be checked

**Output:**

* bool: return true if N is a complete number, otherwise return false

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 6 | true |

bool completeNum(int N)

{

// TODO

}

### Answer q1.1

bool completeNum(int N)

{

// TODO

int sum=0;

if(N==1) return false;

for(int i=1; i<=N; i++)

// **TH đặc biệt: 1 ko phải là complete number.**

{

sum=sum+i;

if(sum==N) return true;

else if(sum>N) return false;

}

}

**Mỗi số hoàn hảo có thể biểu diễn dưới dạng tổng các số tự nhiên liên tiếp**

**6 = 1+2+3**

**28 = 1+2+3+4+5+6+7**

**496 = 1+2+3+...+30+31**

**8128 = 1+2+3+...+126+127**

## **2.Recursion**

### [Q 2.2] Tìm UCLN bằng đệ quy và vòng lặp.

**[Tiếng Việt]**

Một hàm tìm ước số chung lớn nhất của 2 số nguyên dương có thể viết thông qua đệ quy và vòng lặp đơn giản. Bạn hãy viết hàm **gcdRecursion**để hiện thực tìm ước chung lớn nhất bằng đệ quy và hàm **gcdIteration**để tìm ước số chung lớn nhất bằng vòng lặp

**Đầu vào:**

Lần lượt 2 số nguyên p, q (1 ≤ p,q < 109 ).

**Đầu ra:**

Hàm **gcdRecursion**và **gcdIteration**lần lượt trả về giá trị là ước chung lớn nhất của p, q

**[English]**

A function that finds the greatest common divisor of two positive integers can be written through simple recursion and looping. You write the function gcdRecursion to perform the greatest common divisor by recursion and the function gcdIteration to find the greatest common divisor by loop.

**Input:**

Two integers p, q respectively (1 ≤ p,q < 109).

**Output:**

The gcdRecursion and gcdIteration functions return the greatest common divisor of p, q, respectively.

**Template of full code:**

#include<iostream>

#include<string>

#include <string>

#include <sstream>

#include <fstream>

#include <vector>

using namespace std;

/\* END of library \*/

bool gcdRecursion(int p, int q)

{

// BEGIN YOUR IMPLEMENTATION [1]

// TODO

// END YOUR EMPLEMENTATION [1]

}

int gcdIteration(int p, int q)

{

hiddenCheck();

// BEGIN YOUR IMPLEMENTATION [2]

// TODO

// END YOUR EMPLEMENTATION [2]

return 0;

}

int main()

{

hiddenCheck();

int p,q;

cin>>p>>q;

cout<<gcdRecursion(p,q)<< " "<<gcdIteration(p,q);

return 0;

}

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 5 5 | 5 5 |

int gcdRecursion(int p, int q)

{

// BEGIN YOUR IMPLEMENTATION [1]

// TODO

// END YOUR EMPLEMENTATION [1]

}

int gcdIteration(int p, int q)

{

// BEGIN YOUR IMPLEMENTATION [2]

// TODO

// END YOUR EMPLEMENTATION [2]

}

### [Q 2.5] Độ dài mảng một chiều.

**[Tiếng Việt]**

Cho một chuỗi, hiện thực hàm

int strLen(char\* str){}

để tính độ dài của chuỗi sử dụng đệ quy.

Lưu ý không được sử dụng các từ khóa như for, while, goto (thậm chí là tên biến, comment).

Trong bài tập này đã khai báo #include <iostream> và  using namespace std;

**[English]**

Given a string, implement function

int strLen(char\* str){}

to calculate length of the string using recursion.

Please note that you can't using key work for, while, goto (even in variable names, comment).

For this exercise, we have #include <iostream> and using namespace std;

For example:

| **Test** | **Result** |
| --- | --- |
| char str[] = "Truong DH Bach Khoa";  cout << strLen(str); | 19 |

int strLen(char\* str)

{

/\*

\* STUDENT ANSWER

\*/

}

### Answer q2.5

int strLen(char\* str)

{

/\*

\* STUDENT ANSWER

\*/

if(\*str=='\0') return 0;

else return 1+ strLen(str+1);

}

### [Q 2.6] In theo format

**[Tiếng Việt]**

Cho một số dương, in theo một mẫu mà không sử dụng bất kỳ vòng lặp nào.

**Đầu vào:** n = 16

**Đầu ra:** 16, 11, 6, 1, -4, 1, 6, 11, 16 (không có khoảng trắng ở cuối)

**Đầu vào:** n = 10

**Đầu ra:** 10, 5, 0, 5, 10 (không có khoảng trắng ở cuối)

Về cơ bản, đầu tiên chúng ta bớt 5 cái một cho đến khi chúng ta đạt đến số âm hoặc 0. Sau khi chúng tôi đạt đến 0 hoặc số âm, chúng tôi thêm 5 cho đến khi chúng tôi đạt đến n.

**Lưu ý:** Xin lưu ý rằng bạn không thể sử dụng key work for, while, goto (ngay cả trong tên biến, comment).

Bạn có thể triển khai các hàm đệ quy khác nếu cần.

Đối với bài tập này, chúng ta có #include <iostream> và sử dụng namespace std;

**[English ]**

Given a positive number, print following a pattern without using any loop.

**Input:** n = 16

**Output:** 16, 11, 6, 1, -4, 1, 6, 11, 16 (has no space at the end)

**Input:** n = 10

**Output:** 10, 5, 0, 5, 10 (has no space at the end)

We basically first reduce 5 one by one until we reach a negative or 0. After we reach 0 or negative, we one add 5 until we reach n.

**Note:** Please note that you can't using key work for, while, goto (even in variable names, comment).

You can implement other recursive functions if needed.

For this exercise, we have #include <iostream> and using namespace std;

For example:

| **Test** | **Result** |
| --- | --- |
| printPattern(16); | 16 11 6 1 -4 1 6 11 16 |
| printPattern(10); | 10 5 0 5 10 |

void printPattern(int n)

{

/\*

\* STUDENT ANSWER

\*/

}

### Answer q2.6 from vinh, note condition n>0

void printdown(int &n)

{

if(n>0)

**//thù ông thầy ra đề**

{

cout<<n<<" ";

n=n-5;

printdown(n);

}

}

void printup(int &n,int a)

{

if(n<a)

{

cout<<n<<" ";

n=n+5;

printup(n,a);

}

}

void printPattern(int n)

{

/\*

\* STUDENT ANSWER

\*/

int a=n;

printdown(a);

printup(a,n);

cout<<n;

}

## 3. **Pointer Basic**

### [Q 3.1] Truy xuất mảng theo chỉ số

Hiện thực hàm sau:  
**int getValueAtIndex(int \*ptr, int k);**  
Trả về giá trị của tại vị trí được chỉ định trong mảng qua con trỏ.  
Trong đó:  
**ptr** là con trỏ tới phần tử đầu tiên trong mảng.  
**k** là vị trí cần truy xuất phần tử (giá trị này không vượt quá độ dài của mảng).  
------------------------------------------------------------  
Implement the following function:  
**int getValueAtIndex(int \*ptr, int k);**Return value at the position of the index number through pointer.  
Where:  
**ptr** is a pointer to the first element in the array.  
**k** is the access position (this value does not exceed the length of the array).

For example:

| **Test** | **Result** |
| --- | --- |
| int arr[] = {1, 2, 3, 4, 5};  int k = 3;  cout << getValueAtIndex(arr, k); | 4 |

int getValueAtIndex(int \*ptr, int k)

{

}

### Answer q3.1

int getValueAtIndex(int \*ptr, int k)

{

return \*(ptr + k);

}

### [Q 3.2] Hoán đổi giá trị bằng con trỏ

Hiện thực hàm sau:  
**void swap(int \*ptr2, int \*ptr2);**  
Thực hiện hoán đổi giá trị tại vị trí của 2 con trỏ trỏ tới.  
Trong đó:  
**ptr1, ptr2** lần lượt là là các con trỏ trỏ tới vị trí cần hoàn đổi.  
------------------------------------------------------------  
Implement the following function:  
**void swap(int \*ptr2, int \*ptr2);**  
Swap the value at the position that 2 pointers point to.  
Where:  
**ptr1, ptr2** respectively are pointers to swapping positions.

For example:

| **Test** | **Result** |
| --- | --- |
| int a = 5;  int b = 6;  swap(&a, &b);  cout << a << ' ' << b; | 6 5 |

void swap(int \*ptr1, int \*ptr2)

{

}

### Answer q3.2

void swap(int \*ptr1, int \*ptr2)

{

int tmp=\*ptr2;

\*ptr2=\*ptr1;

\*ptr1=tmp;

}

# INLAB

## **1.Function**

### [Q 1.5] Check if a string is palindrome

**[Tiếng Việt]**

Một chuỗi được gọi là palindrome nếu chuỗi đó giống với chuỗi được đảo ngược từ chính nó. Ví dụ: “eye”, “noon”, “abcba”...

Hãy viết hàm kiểm tra xem một chuỗi có là palindrome hay không?

**Đầu vào:**

* const char\* str: chuỗi cần kiểm tra palindrome. str chỉ bao gồm chữ cái thường

**Đầu ra:**

* bool: true nếu chuỗi str là palindrome, ngược lại false

**[English]**

A string is a palindrome if it reads the same forward and backward. For example: "eye", "noon", "abcba", ...

Write a function to check if a given string is a palindrome

**Input:**

* const char\* str: the string to be checked. str only contains lowercase letters

**Output:**

* bool: true if str is a palindrome, false otherwise

For example:

| **Test** | **Result** |
| --- | --- |
| const char\* str = "abba";  cout << isPalindrome(str); | 1 |
| const char\* str = "axz";  cout << isPalindrome(str); | 0 |

bool isPalindrome(const char\* str) {

// Write your code here

}

### Answer q1.5

bool isPalindrome(const char\* *str*) {

    // Write your code here

    const char\* p = *str*;

    int length = 0;

    while(\*p != '\0') {

        length ++;

        p=p+1;

    }

    for(int i = 0; i < length; i++){

        if(*str*[i] != *str*[length - 1-i]) return false;

    }

    return true;

}

### [Q 1.6] Check if a natural number is special

**[Tiếng Việt]**

Một số tự nhiên n được gọi là đặc biệt khi và chỉ khi n là số nguyên tố và tổng các chữ số của n cũng là số nguyên tố. Viết hàm kiểm tra một số tự nhiên có đặc biệt hay không.

**Đầu vào:**

* int n: số tự nhiên cần kiểm tra có phải số đặc biệt không

**Đầu ra:**

* bool: trả về true nếu n là số đặc biệt, ngược lại trả về false

**[English]**

A natural number n is special if and only if n is a prime number and the sum of all the digits of n is also a prime number. Write a function that determines if a natural number is a special or not.

**Input:**

int n: a natural number n. 0 <= n <= 1000

**Output:**

bool: return true if n is special, return false otherwise

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| int n;  cin >> n;  cout << isSpecialNumber(n); | 23 | 1 |
| int n;  cin >> n;  cout << isSpecialNumber(n); | 7 | 1 |

bool isSpecialNumber(int n) {

// Write your code

}

### Answer q.16

bool isSpecialNumber(int *n*) {

    // Write your code

    //number is prime number

    for(int i = 1; i <*n*; i++) {

        if(i>1 && (*n*%i)==0) return false;

    }

    //digit is prime number

    int tmp=0;

    while(*n*!= 0){

        tmp=tmp+*n*%10;

*n*=*n*-*n*%10;

*n*=*n*/10;

    }

    for(int i = 1; i <tmp; i++) {

        if(i>1 && tmp%i == 0) return false;

    }

    return true;

}

### [Q 1.7] Check if every elements in an array is unique

**[Tiếng Việt]**

Viết hàm kiểm tra các phần tử trong mảng có duy nhất hay không

**Đầu vào:**

* int\* arr: mảng số tự nhiên
* int n: số lượng phần tử trong mảng

**Đầu ra:**

* bool: trả về true nếu các phần tử trong mảng là duy nhất, ngược lại trả về false

**Chú ý:** arr[i] nằm trong khoảng từ [0, 1000]

**[English]**

Write a function that determines if the elements in the given array is unique

**Input:**

* int\* arr: array of integer
* int n: the size of the array

**Output:**

* bool: return true if the elements in arr is unique, otherwise return false

**Note:** arr[i] is in the range of [0, 1000]

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| int n;  cin >> n;  int\* arr = new int[n];  for(int i = 0; i < n; i++) {  cin >> arr[i];  }  cout << checkElementsUniqueness(arr, n);  delete[] arr; | 5  2 5 13 5 2 | 0 |
| int n;  cin >> n;  int\* arr = new int[n];  for(int i = 0; i < n; i++) {  cin >> arr[i];  }  cout << checkElementsUniqueness(arr, n);  delete[] arr; | 3  17 10 25 | 1 |

bool checkElementsUniqueness(int\* arr, int n) {

// Write your code

}

### Answer 1.7

bool checkElementsUniqueness(int\* *arr*, int *n*) {

    // Write your code

    for(int i=0; i<*n*; i++){

        for(int j=i+1; j<*n*; j++){

            if(*arr*[j] == *arr*[i]) return false;

        }

    }

    return true;

}

### [Q 1.10] Caesar Cipher

**[Tiếng Việt]**

Viết hàm để mã hóa và giải mã một đoạn text theo phương pháp Caesar Cipher. Để mã hoá và giải mã một chuỗi ký tự text, ta cần một tham số có giá trị nguyên là shift.

Hàm mã hóa sẽ thay đổi từng chữ cái trong text bằng cách dịch chuyển chữ cái đó sang phải shift lần trong bảng chữ cái. Ví dụ với shift = 3. Khi đó 'a' được mã hoá thành 'd', 'b' được mã hoá thành 'e',... 'z' được mã hoá thành 'c'

Hàm giải mã sẽ nhận một chuỗi ký tự text và giá trị nguyên shift và giải mã chuỗi ký tự này thành chuỗi ban đầu (tức dịch chuyển từng chữ cái sang trái shift lần trong bảng chữ cái)

**Đầu vào:**

* char\* text: chuỗi ký tự cần được mã hoá hoặc giải mã, chỉ bao gồm chữ cái thường và hoa
* int shift: giá trị dịch chuyển trong Caesar Cipher

**[English]**

Write a function to encrypt a text string using Caesar Cipher technique.

In this technique, to encrypt a string we need a parameter of type integer called 'shift'.

The encrypt method will shift each letter by some fixed number of position (determined by the parameter 'shift') to the right in the alphabet. For example, when 'shift' is 3, 'a' will be replaced by 'd', 'b' will become 'e', ... , 'z' will become 'c'.

The decrypt method will receive an encoded string and a shift value and it will decode this string to get the original string, which means shifting each character to the left in the alphabet.

**Input:**

* char\* text: the text string that needs to be encrypted or decrypted. text only contains lowercase and uppercase ASCII letters
* int shift: the shift value in Caesar Cipher technique

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| int n, shift;  cin >> n >> shift;  char\* text = new char[n+1];  for(int i = 0; i < n; i++) cin >> text[i];  text[n] = 0;  encrypt(text, shift);  cout << text << '\n';  decrypt(text, shift);  cout << text;  delete[] text; | 6 3  aczDYZ | dfcGBC  aczDYZ |
| int n, shift;  cin >> n >> shift;  char\* text = new char[n+1];  for(int i = 0; i < n; i++) cin >> text[i];  text[n] = 0;  encrypt(text, shift);  cout << text << '\n';  decrypt(text, shift);  cout << text;  delete[] text; | 16 25  programmingisfun | oqnfqzllhmfhretm  programmingisfun |

void encrypt(char\* text, int shift) {

// Write your code

}

void decrypt(char\* text, int shift) {

// Write your code

}

### Answer q1.10

void encrypt(char\* *text*, int *shift*) {

    // Write your code

*shift*=*shift*%26;

    for(int i=0; i<strlen(*text*); i++){

        if(*text*[i]>=65 && *text*[i]<=90)

        {

            //case: A to Z: 65 to 90

            int tmp=*text*[i];

            tmp=tmp+*shift*;

            if(tmp<65) tmp=tmp+26;

            if(tmp>90) tmp=tmp-26;

*text*[i]=tmp;

        }

        else{

            //case: a to z: 97 to 122

            int tmp=*text*[i];

            tmp=tmp+*shift*;

            if(tmp<97) tmp=tmp+26;

            if(tmp>122) tmp=tmp-26;

*text*[i]=tmp;

        }

    }

}

void decrypt(char\* *text*, int *shift*) {

    // Write your code

*shift*=*shift*%26;

    for(int i=0; i<strlen(*text*); i++){

        if(*text*[i]>=65 && *text*[i]<=90)

        {

            //case: A to Z: 65 to 90

            int tmp=*text*[i];

            tmp=tmp-*shift*;

            if(tmp<65) tmp=tmp+26;

            if(tmp>90) tmp=tmp-26;

*text*[i]=tmp;

        }

        else{

            //case: a to z: 97 to 122

            int tmp=*text*[i];

            tmp=tmp-*shift*;

            if(tmp<97) tmp=tmp+26;

            if(tmp>122) tmp=tmp-26;

*text*[i]=tmp;

        }

    }

}

## **2.Recursion**

### [Q 2.7] Chuyển thập phân sang nhị phân

**[Tiếng Việt ]**

Cho một số thập phân dương làm đầu vào, chúng ta cần triển khai hàm

long int decimalToBinary(int decimal\_number){}

để chuyển đổi số thập phân dương đã cho thành số nhị phân tương đương.

Xin lưu ý rằng bạn không thể sử dụng từ khóa for, while, goto (ngay cả trong tên biến, comment).

Đối với bài tập này, chúng ta có #include <iostream> và sử dụng namespace std;

**[English ]**

Given a positive decimal number as input, we need to implement function

long int decimalToBinary(int decimal\_number){}

to convert the given positive decimal number into equivalent binary number.

Please note that you can't using key work for, while, goto (even in variable names, comment).

For this exercise, we have #include <iostream> and using namespace std;

For example:

| **Test** | **Result** |
| --- | --- |
| cout << decimalToBinary(20); | 10100 |

long int decimalToBinary(int decimal\_number)

{

/\*

\* STUDENT ANSWER

\*/

}

### Answer q2.7

long int decimalToBinary(int *decimal\_number*)

{

    /\*

    \* STUDENT ANSWER

    \*/

    if (*decimal\_number* == 0)

        return 0;

    else

        return (*decimal\_number* % 2 + 10 \*decimalToBinary(*decimal\_number* / 2));

}

[Q 2.1] Kiểm tra xem tất cả phần tử của một dãy số nguyên có duy nhất hay không

**[Tiếng Việt ]**

Viết một hàm **đệ quy**

bool checkDuplicate(int\* ar,int size)

{

  //Implement content in function  here

}

kiểm tra xem tất cả phần tử của một dãy số nguyên có duy nhất hay không (tức là dãy sẽ không có hai phần tử riêng biệt nào có cùng giá trị). Ví dụ dãy {1,2,3,4,5} là duy nhất, còn dãy {1,2,3,2,5} là không duy nhất vì lặp lại giá trị 2.

**Đầu vào:**

Hàm **checkDuplicate**sẽ nhận vào một dãy các số nguyên có giá trị trong khoảng (-109; 109) và dãy đó có kích thước trong đoạn [1; 104] cùng với kích thước của mảng.

**Đầu ra:**

Trả về **true**nếu dãy số nguyên không có 2 phần tử nào có cùng giá trị, ngược lại trả về **false**

**[English ]**

Write a **recursive**function

bool checkDuplicate(int\* ar,int size)

{

  //Implement content in function  here

}

to check if all elements in an array are unique (meaning there are no identical value element). I.e {1,2,3,4,5} is unique, and {1,2,3,2,5} is not unique because it has two element with value 2.

**Input:**

Function **checkDuplicate**will receive an integer array with value between (-109; 109) and the size in [1; 104] and array size.

**Output**:

Return **true**if there is no identical element, else return **false**

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 5  1 2 3 4 5 | 1 |

### [Q 2.1] Kiểm tra xem tất cả phần tử của một dãy số nguyên có duy nhất hay không

**[Tiếng Việt ]**

Viết một hàm **đệ quy**

bool checkDuplicate(int\* ar,int size)

{

  //Implement content in function  here

}

kiểm tra xem tất cả phần tử của một dãy số nguyên có duy nhất hay không (tức là dãy sẽ không có hai phần tử riêng biệt nào có cùng giá trị). Ví dụ dãy {1,2,3,4,5} là duy nhất, còn dãy {1,2,3,2,5} là không duy nhất vì lặp lại giá trị 2.

**Đầu vào:**

Hàm **checkDuplicate**sẽ nhận vào một dãy các số nguyên có giá trị trong khoảng (-109; 109) và dãy đó có kích thước trong đoạn [1; 104] cùng với kích thước của mảng.

**Đầu ra:**

Trả về **true**nếu dãy số nguyên không có 2 phần tử nào có cùng giá trị, ngược lại trả về **false**

**[English ]**

Write a **recursive**function

bool checkDuplicate(int\* ar,int size)

{

  //Implement content in function  here

}

to check if all elements in an array are unique (meaning there are no identical value element). I.e {1,2,3,4,5} is unique, and {1,2,3,2,5} is not unique because it has two element with value 2.

**Input:**

Function **checkDuplicate**will receive an integer array with value between (-109; 109) and the size in [1; 104] and array size.

**Output**:

Return **true**if there is no identical element, else return **false**

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 5  1 2 3 4 5 | 1 |

### Answer q2.1

//Implement content in function here

if(size <= 1)

{

return true;

}

else

{

if(\*ar != \*(ar+size-1) && (size-1>0))

{

return checkDuplicate(ar,size-1)\*checkDuplicate(ar+1,size-1);

}

else if(size-1>0) return false;

}

return true;

### [Q 2.4] recursiveSearch

**[Tiếng Việt ]**

Viết một hàm **recursiveSearch**nhận đầu vào là một dãy có N phần tử, và giá trị M. Hàm recursiveSearch sẽ trả về vị trí đầu tiên của giá trị M trong dãy (giả sử phần tử này có chỉ số là **k**) và xóa phần tử ở vị trí **k** ra khỏi dãy. Nếu không tìm thấy thì sẽ trả về -1

**Đầu vào:**

Hàm recursiveSearch nhận các giá trị đầu vào sau:

* Số phần tử của dãy N (0 ≤ N ≤ 108)
* Dãy N phần tử arr và mỗi phần tử có giá trị trong khoảng (0; 109)
* Số M (0 < M < 109)
* Giá trị khởi đầu của biến index từ hàm main là 0

**Đầu ra:**

Hàm recursiveSearch sẽ trả về theo yêu cầu của bài toán.

**[ English ]**

Write a **recursiveSearch** function that takes an array of N elements as input, and the value M. The recursiveSearch function returns the first position of the value M in the sequence (assuming this element has an index of k) and deletes the part. the element at position k out of the sequence. If not found, will return -1

**Input:**

The recursiveSearch function takes the following input values:

Number of elements of the sequence N (0 N ≤ 108)

The sequence of N elements arr and each element has a value in the range (0; 109)

Number M (0 < M < 109)

The initial call for index variable in main() is 0

**Output:**

The recursiveSearch function will return the request of the problem.

**[ Template code ]**

int recursiveSearch(int& n , int m, int arr[], int index)

{

// Implement function content here

}

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 5 1  1 2 3 4 5 | 0  2 3 4 5 |

### Answer q2.4

int recursiveSearch(int& *n* , int *m*, int *arr*[], int *index*)

{

   // Implement function content here

    //begin from main function: index=0

    //index: from 0 to n-1

    if(*n*==0) return -1; //IMPORTANT TESTCASE

    if(*m*!= *arr*[*index*]){

        if(*index*==*n*-1) return -1;

        return recursiveSearch(*n*,*m*,*arr*,*index*+1);

    }

    else{

        //delete index'th element

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

*arr*[i]=*arr*[i+1];

        }

*n*--;

        return *index*;

    }

}

## 3. **Pointer Basic**

### [Q 3.4] Thêm phần tử vào mảng 1 chiều sử dụng con trỏ

Hiện thực hàm sau:  
**void add(int \*ptr, int n, int k);**  
Thực hiện thêm phần tử vào cuối của mảng 1 chiều được cho bởi con trỏ.  
Trong đó:  
**ptr** là con trỏ tới phần tử đầu tiên trong mảng.  
**n, k** lần lượt là kích thước của mảng và phần tử cần được thêm vào.  
------------------------------------------------------------  
Implement the following function:  
**void add(int \*ptr, int n, int k);**  
Insert element to the end of the 1-dimension array given by a pointer.  
Where:  
**ptr** is a pointer to the first element in the array.  
**n, k** respectively is the size of the array and the element that need to be added.

For example:

| **Test** | **Result** |
| --- | --- |
| int arr[100] = {1, 2, 3, 4, 5};  int n = 5;  int k = 7;  add(arr, n, k);  cout << arr[n]; | 7 |

void add(int \*ptr, int n, int k)

{

}

### Answer q3.4

void add(int \**ptr*, int *n*, int *k*)

{

    \*(*ptr*+*n*) =*k*;

}

### [Q 3.3] Tính tổng phần tử mảng 1 chiều bằng con trỏ

Hiện thực hàm sau:  
**int calcSum(int \*ptr, int n);**  
Tính và trả về tổng của các phần tử trong mảng 1 chiều được cho bởi con trỏ.  
Trong đó:  
**ptr** là con trỏ tới phần tử đầu tiên trong mảng.  
**n** là kích thước của mảng.  
**Lưu ý:** Bạn cần phải dùng dereference operator (\*) để lấy giá trị của các phần tử trong mảng. Không được dùng subscript operator ([]).  
------------------------------------------------------------  
Implement the following function:  
**int calcSum(int \*ptr, int n);**  
Calculate and return the sum of elements of a 1-dimension array given by a pointer.  
Where:  
**ptr** is a pointer to the first element in the array.  
**n** is the size of the array.

**Note**: You need to use the dereference operator (\*) to get the values of the elements in the array. The subscript operator ([]) cannot be used.

For example:

| **Test** | **Result** |
| --- | --- |
| int arr[] = {1, 2, 3, 4, 5};  cout << calcSum(arr, sizeof(arr) / sizeof(arr[0])); | 15 |

int calcSum(int \*ptr, int n)

{

}

### Answer q3.3

int calcSum(int \**ptr*, int *n*)

{

    int result =0;

    for(int i=0; i<*n*; i++){

        result += \*(*ptr*+i);

    }

    return result;

}

# POSTLAB

## **1.Function**

## **2.Recursion**

## 3. **Pointer Basic**