# PRELAB

## [Q 1.1] Zeros array

**Mô tả tiếng Việt:**

Hãy hiện thực hàm **int\* zeros(int n)** tạo một mảng có n phần tử 0.

Đầu vào: Kích thước mảng n.

Đầu ra: Con trỏ trỏ tới mảng vừa được cấp phát.

Lưu ý: Trong trường hợp cấp phát thất bại, hàm sẽ trả về nullptr.

**English version:**

Implement the function  **int\* zeros(int n)** which can create an array with n zero element.

Input: The array size n.

Output: The pointer that points to the allocated array.

Note: In the case of failed allocation, the function will return nullptr value.

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 1 | 0 |

int\* zeros(int n) {

// TODO

}

## Answer q1.1

int\* zeros(int *n*) {

    // TODO

    if(*n*>0){

        int\* tmp= **new** int[*n*];

        if(tmp) //check whether tmp is allocated

        {

            for(int i=0; i<*n*; i++){

                \*(tmp+i)=0;

            }

            return tmp;

        }

        else return nullptr;

    }

    else return nullptr;

}

## [Q 1.2] Shallow copy

**Mô tả tiếng Việt:**

Hãy hiện thực hàm **void shallowCopy(int\*& newArr, int\*& arr)**có chức năng tạo một bản sao của một mảng một chiều.

Đầu vào: Mảng một chiều arr cần được sao chép.

Đầu ra: Mảng đích một chiều newArr cần sao chép tới.

Lưu ý: sau thực thi mảng được sao chép và mảng cần sao chép đều sử dụng chung một vùng nhớ.

**English version:**

Implement the function **void shallowCopy(int\*& newArr, int\*& arr)**that can create a copy from a one-dimensional array.

Input:  The one-dimensional array that needs to be copied.

Output: The destination array.

Note: After finishing execution, both the one-dimensional array that needs to be copied and the destination array use the same data memory.

For example:

| **Test** | **Result** |
| --- | --- |
| int\* arr = new int[2];  arr[0] = 2; arr[1] = 3;  int\* newArr = nullptr;  shallowCopy(newArr, arr);  cout << newArr[0] << ' ' << newArr[1];  delete[] arr; | 2 3 |

void shallowCopy(int\*& newArr, int\*& arr) {

// TODO

}

## What is the difference between int\* & a and int\* b?

“int\*& a” is a reference to a pointer to integer. “int\* b” is a pointer to integer.

A reference to any object serves as an alias to that object. No explicit-dereferencing is required to access the object it refers to (here some pointer to integer) and it cannot be made to refer to a different pointer to integer. Once a reference is “seated” it is permanent for its lifetime. There is no way to access the “reference itself” rather than what it refers to. References are not objects — they are aliases, second names given to objects.

A pointer to integer is an object that may or may not “point to” an integer (it might have the value of nullptr which means it doesn’t currently point to anything). You can set its value to point to any integer whose address you can take.

You can use a pointer to integer that actually points to an integer to access the value of the integer it points to, but you have to use the explicit dereferencing operator \* to get to the value it points to.

EXAMPLE

#include <list>

    #include <iostream>

    #include <cstdint>

    #include <iomanip>

    #include <string>

    #include <sstream>

    #include <algorithm>

    constexpr auto newl = '\n';

    int main()

    {

       int y = 200;

       int x = 100;

       int\* a = &x; //a is pointer to integer that points to x

       int\*& b = a; //b is a reference to pointer to integer.  It returns to a.

       //I can print the value of a (which will be the memory address of x)

       std::cout << std::hex << "value of a: 0x" << a << ".  is memory address of x." << newl;

       std::cout << std::dec << "value pointed by a (i.e. the value at the memory address we just printed): " << \*a << newl;

       std::cout << std::hex << "printing b, which is an alias for a is exactly like printing a: 0x" << b << newl;

       std::cout << std::dec << "because b is an alias to a, anything we can do with a directly, "

                            <<  "we can do to a through b as well.  For example, "

                            <<  "we can dereference the pointer: " << \*b << newl;

        std::cout << "Making some changes..." << newl;

        //We can change what a points to like this

        a = &y;

        // and we can change the value of what it points to:

        \*a += 691;

        //Note that we can't change what B refers to, b will always refer to a

        std::cout << std::hex << "value of a: 0x" << a << ".  is memory address of Y." << newl;

        std::cout << std::dec << "value pointed by a (i.e. the value at the memory address we just printed): " << \*a << newl;

        std::cout << std::hex << "printing b, which is an alias for a is exactly like printing a: 0x" << b << newl;

        std::cout << std::dec << "because b is an alias to a, anything we can do with a directly, "

                            <<  "we can do to a through b as well.  For example, "

                            <<  "we can dereference the pointer: " << \*b << newl;

        std::cout << std::dec << "Value of y printed directly: " << y << newl;

        return 0;

    }

OUTPUT:

value of a: 0x0x7ffd076a6c44. is memory address of x.  
value pointed by a (i.e. the value at the memory address we just printed): 100  
printing b, which is an alias for a is exactly like printing a: 0x0x7ffd076a6c44  
because b is an alias to a, anything we can do with a directly, we can do to a through b as well. For example, we can dereference the pointer: 100  
Making some changes...  
value of a: 0x0x7ffd076a6c48. is memory address of Y.  
value pointed by a (i.e. the value at the memory address we just printed): 891  
printing b, which is an alias for a is exactly like printing a: 0x0x7ffd076a6c48  
because b is an alias to a, anything we can do with a directly, we can do to a through b as well. For example, we can dereference the pointer: 891  
Value of y printed directy: 891

## answer q1.2

void shallowCopy(int\*& *newArr*, int\*& *arr*) {

    // TODO

*newArr*=*arr*;

}

## [Q 1.5] Delete matrix

**Mô tả tiếng Việt:**

Hãy hiện thực hàm **void deleteMatrix(int\*\*& matrix, int r)** thực hiện giải phóng ô nhớ cho một mảng động 2 chiều có r hàng. **matrix** được gán bằng giá trị NULL sau khi thực hiện hàm.

Đầu vào: Mảng động hai chiều matrix có số hàng r cần giải phóng ô nhớ.

**English version:**

Implement the function **void deleteMatrix(int\*\*& matrix, int r)** that can free memory for a dynamic two-dimensional array consisting of r rows. **matrix** should be set to NULL after function's execution.

Input: The dynamic two-dimensional array, matrix, consists of r rows.

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 2 2  1 1  1 1 | SUCCESSFUL |

void deleteMatrix(int\*\*& matrix, int r) {

// TODO;

}

**An int\* is a pointer to an int, so int\*& must be a reference to a pointer to an int. Similarly, int\*\* is a pointer to a pointer to an int, so int\*\*& must be a reference to a pointer to a pointer to an int.**

## **answer q.15**

void deleteMatrix(int\*\*& *matrix*, int *r*) {

    // TODO;

    //int\*\*&: reference to pointer to pointer to integer

*matrix*=nullptr;

}

## [Q 1.3] Deep copy

Hãy hiện thực hàm **int\*\* deepCopy(int\*\* matrix, int r, int c)** trả về một bản sao của matrix gồm r hàng và n cột.

Đầu vào: Con trỏ matrix trỏ đến mảng hai chiều có kích thước r x c.

Đầu ra: Con trỏ trỏ đến mảng hai chiều được sao chép.

Lưu ý: sau thực thi, con trỏ trả về phải trỏ đến vùng nhớ được cấp phát mới và khi matrix truyền vào có kích thước 0, hàm trả về nullptr.

**English version:**

Implement the function **int\*\* deepCopy(int\*\* matrix, int r, int c)** that return a copy of a matrix consisting of  r rows and c colmuns.

Input: Pointer arr points to the one-dimensional array that needs to be copied.

Output: Pointer newArr points to the destination array.

Note: After finishing execution, the one-dimensional array that needs to be copied and the destination array use the two distinct data memory.

For example:

| **Test** | **Result** |
| --- | --- |
| int\*\* m = new int\*[2];  m[0] = new int[2]; m[0][0] = 1; m[0][1] = 2;  m[1] = new int[2]; m[1][0] = 1; m[1][1] = 3;  int\*\* n = deepCopy(m, 2, 2);  cout << n[0][0] << ' ' << n[0][1] << '\n' << n[1][0] << ' ' << n[1][1]; | 1 2  1 3 |

int\*\* deepCopy(int\*\* matrix, int r, int c) {

// TODO

}

## Answer q1.3

int\*\* deepCopy(int\*\* *matrix*, int *r*, int *c*) {

    // TODO

    if(*r*<=0 || *c*<=0) return nullptr;

    return *matrix*;

}

## Answer q1.3 from mrvinh

int\*\* deepCopy(int\*\* *matrix*, int *r*, int *c*) {

    // TODO

    if(*r*<=0 || *c*<=0)

    {

        return NULL;

    }

    if(*r*==*c*==1)

    {

        int \*\* tmp=**new** int\* [1] ;

        tmp[1]=**new** int [1];

        tmp=*matrix*;

        return tmp;

    }

    int\*\*tmp=**new** int\* [*r*];

    for(int i=0;i<*r*;i++)

    {

        tmp[i]=**new** int [*c*];

    }

    for(int j=0;j<*r*;j++)

    {

        for(int k=0;k<*c*;k++)

        {

            tmp[j][k]=*matrix*[j][k];

        }

    }

    return tmp;

}

## [Q 1.7] Insert matrix row

**Mô tả tiếng Việt:**

Cho một mảng động hai chiều matrix có kích thước r x c. Hiện thực hàm **void insertRow(int\*\*& matrix, int r, int c, int\* rowArr, int row)** tiến hành chèn mảng rowArr (có kích thước c) vào hàng thứ row của mảng matrix.

Đầu vào: Mảng 2 chiều matrix có kích thước r x c, hàng cần chèn rowArr và vị trí chèn row.

Đầu ra: Mảng 2 chiều matrix sau khi được chèn.

**English version:**

Given a dynamic two-dimensional array of size r x c. Implement the function **void insertRow(int\*\*& matrix, int r, int c, int\* rowArr, int row)** that can insert the rowArr array (with the size c) into the row position, row, of the matrix.

Input: The two-dimensional matrix of size r x c, the insert row rowArr and the insert position row.

Output: The two-dimensional matrix after insert.

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 2 3  1 2 3  4 5 6  2  7 8 9 | 1 2 3  4 5 6  7 8 9 |

void insertRow(int\*\*& matrix, int r, int c, int\* rowArr, int row) {

// TODO

}

## Answer q1.7

void insertRow(int\*\*& *matrix*, int *r*, int *c*, int\* *rowArr*, int *row*) {

    // TODO

    int\*\* temp=**new** int\*[*r*+1];

    for(int i=0; i<*r*+1; i++){

        temp[i]=**new** int[*c*];

    }

    if(*row*==*r*)

    {

        //copy r rows to temp

        for(int i=0; i<*r*; i++){

            for(int j=0; j<*c*; j++){

                temp[i][j]=*matrix*[i][j];

            }

        }

        //insert new row at the last row

        temp[*row*]=*rowArr*;

    }

    else{

        //case: row: [0,r)

        //copy the first row rows to temp

        for(int i=0; i<*row*; i++){

            for(int j=0; j<*c*; j++){

                temp[i][j]=*matrix*[i][j];

            }

        }

        //insert rowArr

        temp[*row*]=*rowArr*;

        //insert  from row (th) to r(th) of matrix into temp

        for(int i=*row*+1; i<*r*+1; i++) {

            for(int j=0; j<*c*; j++) {

                temp[i][j]=*matrix*[i-1][j];

            }

        }

    }

*matrix*=temp;

}

## [Q 2.1] SCP Struct Declaration

Kho lưu trữ của tổ chức SCP chứa hàng loạt các vật thể dị thường. Mỗi vật thể dị thường được lưu trữ dưới struct SCP với các thông tin như sau:

* id: kiểu int, là mã định danh (hay mã vật thể), phân biệt giữa các vật thể với nhau.
* objClass: kiểu int, là phân loại của vật thể đó.
* speConProcedures: kiểu string, mô tả quy trình quản thúc đặc biệt của vật thể đó.
* description: kiểu string, mô tả về các đặc điểm của vật thể đó.
* addendums: kiểu string\*, là một tập hợp của các phụ lục đính kèm, mô tả các thông tin bổ sung cho vật thể đó.
* numAddendums: kiểu int, là số lượng phụ lục đính kèm.

Hiện thực struct SCP với các yêu cầu trên.

Ghi chú: (Các) thư viện iostream và string đã được khai báo, và namespace std đã được sử dụng.

**English version:**

SCP Foundation's classified archives consist of records of paranormal objects. The information of each object is stored using struct SCP with following requirements:

* id: integer, the identifier (or item number) of the object.
* objClass: integer, the object class.
* speConProcedures: string, specification of the object's special containment procedures.
* description: string, description of the object.
* addendums: string\*, array of addendums describing additional information about the object.
* numAddendums: integer, the number of addendums attached.

Declare struct SCP with mentioned requirements.

Note: Libraries iostream and string have been imported, and namespace std has been used.

struct SCP {

// Student answer

};

## Answer q2.1

struct *SCP* {

    // Student answer

    int id;

    int objClass;

*string* speConProcedures;

*string* description;

*string*\* addendums=nullptr;

    int numAddendums;

};

## [Q 2.4] SCP Struct Pointer Members Accessing

Cho struct SCP lưu trữ thông tin các vật thể dị thường được mô tả như sau:

struct SCP {

int id;

int objClass;

string speConProcedures;

string description;

string\* addendums;

int numAddendums;

};

Hiện thực một hàm với prototype sau:

void addAddendum(SCP &obj, string addendum);

Hàm thực hiện bổ sung một phụ lục addendum vào cuối danh sách phụ lục (addendums) của obj.

Ghi chú: (Các) thư viện iostream và string đã được khai báo, và namespace std đã được sử dụng.

**English version:**

Struct SCP used to store information about paranormal objects is declared as below:

struct SCP {

int id;

int objClass;

string speConProcedures;

string description;

string\* addendums;

int numAddendums;

};

Implement the function with the following prototype:

void addAddendum(SCP &obj, string addendum);

The function appends a new addendum into the array of addendums of the object obj.

Note: Libraries iostream and string have been imported, and namespace std has been used.

For example:

**Test**

string\* addendums = new string[1];

addendums[0] = "Document #055-1: An Analysis of SCP-055\nThe author puts forward the hypothesis that SCP-055 was never formally acquired.";

SCP obj {55, 2, "Object is kept within a five (5) by five (5) by two point five (2.5) meter square room.", "All known information about the object is that it is not round.", addendums, 1};

addAddendum(obj, "Document #055-2: Report of Dr. John Marachek\nSurvey team #19-055-127BXE was successfully able to enter SCP-055's container and ascertain the appearance.");

cout << obj.addendums[1];

delete [] obj.addendums;

**Result**

Document #055-2: Report of Dr. John Marachek

Survey team #19-055-127BXE was successfully able to enter SCP-055's container and ascertain the appearance.

void addAddendum(SCP &obj, string addendum) {

// Student answer

}

## Answer q2.4

void addAddendum(*SCP* &*obj*, *string* *addendum*) {

    // Student answer

*string*\* temp = **new** *string*[*obj*.numAddendums+1];

        for(int i=0; i<*obj*.numAddendums; i++){

            temp[i] = *obj*.addendums[i];

        }

        temp[*obj*.numAddendums] = *addendum*;

*obj*.numAddendums++;

*obj*.addendums=temp;

}

## [Q 3.1] Compare two linked list

Cho chương trình khởi tạo, trong đó:

- struct node: đại diện cho một node của linked list  
- Hàm createLinkedList:  
    + Nhận vào số phần tử (>0) của link list  
    + Xây dựng một linked list với dữ liệu của các node được nhập từ đầu vào chuẩn (stdin), node mới được thêm vào **CUỐI**linked list  
    + Trả về con trỏ đến node đầu tiên của linked list.  
- Hàm isEqual so sánh hai linked list:  
   + Nhận vào con trỏ head1 của linked list thứ nhất, con trỏ của một head2 của linked list thứ 2  
   + Hàm trả về true khi và chỉ khi hai linked list có cùng chiều dài và giá trị của node ở vị trí tương ứng luôn bằng nhau. Ngược lại, hảm trả về false.  
- Hàm main đọc vào số phần tử của linked list, gọi hàm createLinkedList để khởi tạo linked list, sau đó gọi hàm print để in ra các phần tử của linked list.  
Bạn hãy hoàn thành hàm **isEqual**   
**Đầu vào**:   
Các giá trị nhập vào từ đầu vào chuẩn (stdin) đều có giá trị trong khoảng (0; 5000)

Write a program where:  
- struct node: represents a node in linkedlists  
- functions createLinkedList:  
    + Receive the size of a linked list (>0)  
    + create a linked list with value from standard input (stdin) a new node is add to the  **END** of the linked list  
    + Return a pointer which points to the first node of the linked list.  
- Function isEqual compare two linked list:  
   + Receive the pointer head1 of the 1st linked list , pointer head2 of the 2nd linked list  
   + Return true if and only if 2 the linked lists have indentical size, and node value. Otherwise, return false.  
- Function main reads the size of the linked list, calls function createLinkedList to Initialize the linked list, then call functdion print to print all the node's values of the linked list.  
Complete function **isEqual**   
**Input**:   
The value input from standard input (stdin) with value in (0; 5000)

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 5  1 3 5 7 9  5  1 3 5 7 9 | 1 |

#include <iostream>

using namespace std;

struct node

{

int data;

node \*next;

};

node \*createLinkedList(int n); // The implementation is provided implicitly

bool isEqual(node \*head1, node \*head2)

{

// TO DO

}

int main()

{

int n = 0;

cin>> n;

node \*head1 = createLinkedList(n);

int m = 0;

cin>> m;

node \*head2 = createLinkedList(m);

cout << isEqual(head1, head2) << endl;

return 0;

}

## Answer q3.1

bool isEqual(*node* \**head1*, *node* \**head2*)

{

  // TO DO

    if(*head1* == nullptr && *head2* == nullptr) return true;

    else{

        if(*head1*== nullptr || *head2*== nullptr) return false;

        //case: head1 and head2 not nullptr

        if(*head1*->data != *head2*->data) return false;

        //explain: head1->data means (\*head1).data

        else{

            return isEqual(*head1*->next,*head2*->next);

        }

    }

}

**//explain: head1->data means (\*head1).data**

## [Q 3.2] Count node

Cho chương trình khởi tạo, trong đó:  
- struct node: đại diện cho một node của linked list  
- Hàm print: nhận vào con trỏ head của linked list và in ra từng phần tử của linked list  
- Hàm createLinkedList:  
    + Nhận vào số phần tử (>0) của link list  
    + Xây dựng một linked list với dữ liệu của các node được nhập từ đầu vào chuẩn (stdin), node mới được thêm vào **CUỐI**linked list  
    + Trả về con trỏ đến node đầu tiên của linked list.  
- Hàm countNode:  
    + Nhận đầu vào là con trỏ head của linked list  
    + Trả về số lượng nodes trong linked list  
- Hàm main đọc vào số phần tử của linked list, gọi hàm createLinkedList để khởi tạo linked list, sau đó gọi hàm print để in ra các phần tử của linked list.  
Bạn hãy hoàn thành hàm **countNode**  
**Đầu vào:**Số n là số lượng phần tử trong linked list (0 < n < 5000)  
n số tiếp theo là giá trị của mỗi node trong linked list, giá trị là một số nguyên có giá trị trong khoảng (-5000; 5000)  
**Đầu ra:**Thỏa yêu cầu bài toán

Write a program where:  
- struct node: represents a node in linkedlists  
- functions createLinkedList:  
    + Receive the size of a linked list (>0)  
    + create a linked list with value from standard input (stdin) a new node is add to the  **END** of the linked list  
    + Return a pointer which points to the first node of the linked list.  
- function countNode:  
    + Input is the head pointer of the linked list  
    + return the number of nodes in linked list  
- function main reads the size of the linked list, calls createLinkedList to Initialize the linked list, then calls print to print the linked list.  
Complete function **countNode**  
**Input:**  
n is size of the linked list (0 < n < 5000)  
n following numbers are values of each node in the linked list, each value is an integer in (-5000; 5000)  
**Output:**  
Satisfy the requirements

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 5  1 3 5 7 5 | 1  3  5  7  5  5 |

#include <iostream>

using namespace std;

struct node

{

int data;

node \*next;

};

node \*createLinkedList(int n); // The implementation is provided implicitly

int countNode(node\* head)

{

//TODO

}

void print(node \*head)

{

while (head != nullptr)

{

cout << head->data << endl;

head = head->next;

}

}

int main()

{

int n = 0;

cin >> n;

node \*head = createLinkedList(n);

print(head);

cout<<endl;

cout<<countNode(head);

return 0;

}

## Answer q3.2

int countNode(*node*\* *head*)

{

  //TODO

*node*\* temp=*head*;

    int count=0;

    while(temp!=nullptr){

        count++;

        temp=temp->next;

    }

    return count;

}

## [Q 3.4] Create linked list 2

Cho chương trình khởi tạo, trong đó:  
- struct node: đại diện cho một node của linked list  
- Hàm print: nhận vào con trỏ head của linked list và in ra từng phần tử của linked list  
- Hàm createLinkedList:  
    + Nhận vào số phần tử (>0) của link list  
    + Xây dựng một linked list với dữ liệu của các node được nhập từ đầu vào chuẩn (stdin), node mới được thêm vào **ĐẦU**linked list  
    + Trả về con trỏ đến node đầu tiên của linked list.  
- Hàm main đọc vào số phần tử của linked list, gọi hàm createLinkedList để khởi tạo linked list, sau đó gọi hàm print để in ra các phần tử của linked list.  
Bạn hãy hoàn thành hàm createLinkedList  
**Đầu vào:**Số n là số lượng phần tử trong linked list (0 < n < 5000)  
n số tiếp theo là giá trị của mỗi node trong linked list, giá trị là một số nguyên có giá trị trong khoảng (-5000; 5000)  
**Đầu ra:**Thỏa yêu cầu bài toán  
**English version:**Write a program where:  
- struct node: represents a node in linkedlists  
- functions createLinkedList:  
    + Receive the size of a linked list (>0)  
    + create a linked list with value from standard input (stdin) a new node is add to the  **First** position of the linked list  
    + Return a pointer which points to the first node of the linked list.  
- Function main reads the size of the linked list, calls function createLinkedList to Initialize the linked list, then call function print to print the linked list.  
Complete function createLinkedList  
**Input:**size n of the linked list (0 < n < 5000)  
following by n numbers, each number is a value of a node in the linked list, each number is an integer in (-5000; 5000)  
**Output:**Satisfy the requirements

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 5  1 3 5 7 9 | 9  7  5  3  1 |

#include <iostream>

using namespace std;

struct node

{

int data;

node \*next;

};

node \*createLinkedList(int n)

{

// TO DO

}

void print(node \*head)

{

while (head != nullptr)

{

cout << head->data << endl;

head = head->next;

}

}

int main()

{

int n = 0;

cin >> n;

if (n > 0)

{

node \*head = createLinkedList(n);

print(head);

}

else

{

cout << "Invalid n" << endl;

}

return 0;

}

## Answer q3.4

*node* \*createLinkedList(int *n*)

{

  // TO DO

*node*\* head=**new** *node*;

    cin>>head->data;

    for(int i=1; i<*n*; i++){

*node*\* newnode = **new** *node*;

        cin>>newnode->data;

        newnode->next=head;

        head=newnode;

    }

    return head;

}

# INLAB

# POSTLAB