**LAB 1**

**Name:** Nguyễn Hữu Khang

**Student’s id:** 2011365

**Question 1:**

The output of the code is “ABC”.

A screenshot of a computer

Description automatically generated

Explanation: “char str[5]” declares a char array which has the length of five. Therefore char ‘A’ is stored at &str[0], ‘B’ is stored at &str[1], ‘C’ is stored at &str[2]. Since &str[3] hold null value and &str[4] don’t hold any variables, so “cout<<str[3]”, “cout<<str[4]” print nothing. “cout<<str” print all three characters that are stored ‘A’,’B’,’C’.

**Question 2:**

The output of the code is “12”

**A picture containing text, screenshot, software, font

Description automatically generated**

“int \*arr[3] = {&a, &b, &c}” is a declaration of an integer pointer array which hold the addresses of a,b,c. “(\*arr[2]) -- ” return the value which is hold in address that is stored in arr[2] and then decreased the value, the value of c after this code is 2. Therefore “(\*arr[2]) –" return 3 because &c is stored in arr[2]. So on “(\*arr[2]) -- - 3” equal to 0 and “\*arr[(\*arr[2]) -- -3]” = “\*arr[0]” = 1.

**Question 3:**

a)

A picture containing text, font, screenshot, clock

Description automatically generated

b)

There isn’t any error when we delete a NULL pointer.

There isn’t any error when we call delete twice on the same pointer.

**Question 4:**

fun(3,2) 🡪 fun(2,5) 🡪 fun(1,7) 🡪 fun(0,8) 🡪 y =8

The result is 8

**Question 5:**

After call f(x) the value of x is 0 because program only pass by value to the function f().

After call g(x) the value of x is 1 because program pass by reference to the function g();

Function h() got an error because “const int &x” declares x as a read-only variable, cannot increase x.

**Question 6:**

a) The output is printing “1” five times

b) The output is also printing “1” infinitely

**Question 7:**

The program has two compile time errors. In default, every method and attribute in class are declared as private. Therefore Test() and x are private, we cannot call them out of class. The solution for these problems is adding “public:” before declare x and Test().

A screen shot of a computer program

Description automatically generated with low confidence

After fixing the output is “5”.

**Question 8:**

After “delete p” the value of “q[2]” is 2 and the value of “p[1][2]” is unknown because “delete p” dereferences the relationship between pointer p and the address of &p[0]. Pointer p now is no longer manage the address of p[0] so the value of p[1][2] is unpredictable.

After “delete q” the value of “q[2]” is unpredictable and the value of p[1][2] is unpredictable.

After “delete [] q” the value of q[2] and p[1][2] are empty.

**Question 9:**

Code:

void onePrime(int\* *arr*, int *n*) {

    if (*n*==1) {

        if (isPrime(*arr*[*n*-1])) {

            cout<<*arr*[*n*-1];

        }

    }

    else {

        if (isPrime(*arr*[*n*-1])) {

            cout<<*arr*[*n*-1];

        }

        else {

            onePrime(*arr*,*n*-1);

        }

    }

}

void allPrime(int\* *arr*, int *n*) {

    if (*n*==1) {

        if (isPrime(*arr*[*n*-1])) {

            cout<<*arr*[*n*-1]<<" ";

        }

    }

    else {

        if (isPrime(*arr*[*n*-1])) {

            allPrime(*arr*,*n*-1);

            cout<<*arr*[*n*-1]<<" ";

        }

        else {

            allPrime(*arr*,*n*-1);

        }

    }

}

Testing result:

A screenshot of a computer program

Description automatically generated with medium confidence

**Question 10:**

**a) Code**

#include<iostream>

using namespace std;

//input matrix size

void inputMatrixSize(int &*r1*, int &*c1*, int &*r2*, int &*c2*) {

    cout<<"Input matrix 1 size: ";

    cin>>*r1*>>*c1*;

    cout<<"Input matrix 2 size: ";

    cin>>*r2*>>*c2*;

}

//input matrix value

void inputMatrix(int *m*[10][10], int *r*, int *c*) {

    for (int i=0; i<*r*; i++) {

        for (int j=0; j<*c*; j++) {

            cout<<"m["<<i<<"]"<<"["<<j<<"]: ";

            cin>>*m*[i][j];

        }

    }

}

//print matrix

void printMatrix(int *m*[10][10], int *r*, int *c*) {

    for (int i=0; i<*r*; i++) {

        for (int j=0; j<*c*; j++) {

            cout<<*m*[i][j]<<" ";

        }

        cout<<endl;

    }

}

//multiply matrix

void multiplication(int *m1*[10][10], int *m2*[10][10], int *m3*[10][10], int *r1*, int *c1*, int *c2*) {

    for (int i =0; i< *r1*; i++) {

        for (int j =0; j<*c2*; j++) {

*m3*[i][j] = 0;

        }

    }

    for (int i=0; i<*r1*; i++) {

        for (int j=0; j<*c2*; j++) {

            for (int k=0; k<*c1*; k++) {

*m3*[i][j] += (*m1*[i][k] \* *m2*[k][j]);

            }

        }

    }

}

int main() {

    int m1[10][10];

    int m2[10][10];

    int m3[10][10]; //Result

    int r1,c1,r2,c2;

    inputMatrixSize(r1,c1,r2,c2);

    while (c1!=r2) {

        cout<<"Please input matrix size again !"<<endl;

        inputMatrixSize(r1,c1,r2,c2);

    }

    cout<<"Input matrix 1: " <<endl;

    inputMatrix(m1,r1,c1);

    cout<<"Input matrix 2: " <<endl;

    inputMatrix(m2,r2,c2);

    multiplication(m1, m2,m3, r1, c1, c2);

    cout<<"RESULT: "<<endl;

    printMatrix(m3,r1,c2);

}

**Testing Result:**

**A screen shot of a computer

Description automatically generated with medium confidence**

**b) Code**

#include<iostream>

using namespace std;

void inputMatrixSize(int &*r1*, int &*c1*, int &*r2*, int &*c2*) {

    cout<<"Input matrix 1 size: ";

    cin>>*r1*>>*c1*;

    cout<<"Input matrix 2 size: ";

    cin>>*r2*>>*c2*;

}

void inputMatrix(int \*\**m*, int *r*, int *c*) {

    for (int i=0; i<*r*; i++) {

        for (int j=0; j<*c*; j++) {

            cout<<"m["<<i<<"]"<<"["<<j<<"]:";

            cin>>*m*[i][j];

        }

    }

}

void printMatrix(int \*\**m*, int *r*, int *c*) {

    for (int i=0; i<*r*; i++) {

        for (int j=0; j<*c*; j++) {

            cout<<*m*[i][j]<<" ";

        }

        cout<<endl;

    }

}

int\*\* multiplication(int\*\* *m1*, int\*\* *m2*, int *r1*, int *c1*, int *c2*) {

    int\*\* m3 = **new** int\*[*r1*];

    for (int i=0; i<*r1*; i++) {

        m3[i]= **new** int[*c2*];

    }

    for (int i=0; i<*r1*; i++) {

        for (int j=0; j<*c2*; j++) {

            m3[i][j] =0;

            for (int k=0; k<*c1*; k++) {

                m3[i][j] += *m1*[i][k]\**m2*[k][j];

            }

        }

    }

    return m3;

}

int main() {

    int r1, c1, r2, c2;

    inputMatrixSize(r1,c1,r2,c2);

    if(c1!=r2) {

        cout<<"Please input the matrix size again !!"<<endl;

        inputMatrixSize(r1,c1,r2,c2);

    }

    int \*\* m1 = **new** int\* [r1];

    int \*\* m2 = **new** int\* [r2];

    for (int i=0; i<r1; i++) {

        m1[i] = **new** int[c1];

    }

    for (int i=0; i<r2; i++) {

        m2[i] = **new** int[c2];

    }

    cout<<"Please input matrix 1:"<<endl;

    inputMatrix(m1,r1,c1);

    cout<<"Please input matrix 2:"<<endl;

    inputMatrix(m2,r2,c2);

    int \*\* m3 = multiplication(m1,m2,r1,c1,c2);

    printMatrix(m3,r1,c2);

    return 0;

}

**Testing result:**

**A screen shot of a computer

Description automatically generated with medium confidence**

**Question 11:**

**Code:**

“powRepetition” is a function using repetition way, “powRecursion” use recursion way.

#include<iostream>

using namespace std;

int powRepetition(int *a*, int *n*) {

    if (*n*==0) return 1;

    int res = *a*;

    for (int i=0; i<*n*-1; i++) {

        res\*=*a*;

    }

    return res;

}

int powRecursion(int *a*, int *n*) {

    if (*n* == 0) return 1;

    return *a*\*powRecursion(*a*, *n*-1);

}

int main() {

    int a, n;

    cout<<"Please input a number: ";

    cin>>a;

    cout<<"Please input a positive exponent: ";

    cin>>n;

    cout<<"Result using repetition: "<<powRepetition(a,n)<<endl;

    cout<<"Result using recursion: "<<powRecursion(a,n)<<endl;

}

**Testing result:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**Question 12:**

**Code:**

#include<iostream>

using namespace std;

class *candidate* {

    private:

    double math;

    double physics;

    double chemistry;

    public:

    int id;

    char\* name;

    candidate(double *math*, double *physics*, double *chemistry*, int *id*, char\* *name*) {

*this* -> math = *math*;

*this* -> physics = *physics*;

*this* -> chemistry = *chemistry*;

*this* -> id = *id*;

*this* -> name = *name*;

    }

    ~ candidate() {}

    double totalGrade() {

        return math + physics + chemistry;

    }

};

int main() {

*candidate* a(10,9,8,2011365,"Khang");

    cout<< a.totalGrade();

}

**Testing result:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**Question 13:**

**Code:**

#include<iostream>

using namespace std;

struct *node* {

    int data;

*node*\* next = NULL;

};

*node*\* ConvertToLinkedList(int *List*[],int *Size*) {

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

    head -> data = *List*[0];

*node*\* current = head;

    for (int i=1; i<*Size*; i++) {

*node*\* newNode = **new** *node*;

        newNode -> data = *List*[i];

        current -> next = newNode;

        current = newNode;

    }

    return head;

}

void PrintLinkedList(*node*\* *Node*) {

    while(*Node*!=nullptr) {

        cout<<*Node* -> data<<" ";

*Node* = *Node* -> next;

    }

}

int main() {

    int List[5]={1,2,3,4,5};

    int Size = 5;

*node*\* linkedList = ConvertToLinkedList(List,Size);

    PrintLinkedList(linkedList);

}

**A screenshot of a computer program

Description automatically generated with medium confidence**

**Question 14:**

#include<iostream>

using namespace std;

struct *node* {

    int data;

*node*\* next = NULL;

};

*node*\* ConvertToLinkedList(int *List*[],int *Size*) {

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

    head -> data = *List*[0];

*node*\* current = head;

    for (int i=1; i<*Size*; i++) {

*node*\* newNode = **new** *node*;

        newNode -> data = *List*[i];

        current -> next = newNode;

        current = newNode;

    }

    return head;

}

*node*\* swap(*node* \**ptr1*, *node*\* *ptr2*) {

*node*\* tmp = *ptr2* -> next;

*ptr2* -> next = *ptr1*;

*ptr1* -> next = tmp;

    return *ptr2*;

}

void PrintLinkedList(*node*\* *Node*) {

    while(*Node*!=NULL) {

        cout<<*Node* -> data<<" ";

*Node* = *Node* -> next;

    }

}

*node*\* SortHelper(*node*\* *head*) {

     if (*head* -> next == NULL) return *head*;

     if (*head* -> data > *head* -> next -> data) {

*head* = swap (*head*, *head* -> next);

     }

*head* -> next = SortHelper(*head*->next);

     return *head*;

}

void SortLinkedList(*node*\* &*head*, int *Size*) {

*node*\* p = *head*;

    while(*Size*!=0) {

        p = SortHelper(p);

*Size*--;

    }

*head* = p;

}

int main() {

    int List[8]={5,8,3,2,9,1,10,32};

    int Size = 8;

*node*\* LinkedList = ConvertToLinkedList(List,Size);

    cout<<"Original LinkedList: ";

    PrintLinkedList(LinkedList);

    cout<<endl;

    SortLinkedList(LinkedList, Size);

    cout<<endl;

    cout<<"Sorted LinkedList: ";

    PrintLinkedList(LinkedList);

}

**Testing result:**

**A screen shot of a computer program

Description automatically generated with low confidence**

**Question 15:**

**Code: I think “maxVal” is not necessary, we can ignore it.**

int myMaxFunc(*node*\* *head*, int *maxVal*) {

    if (*head* -> next == nullptr) return *head* -> data;

    return (*head* -> data > myMaxFunc(*head*->next,*maxVal*)) ? *head* -> data : myMaxFunc(*head* -> next, *maxVal*);

}

**Testing result:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**Question 16:**

*node*\* AddPoly(*node*\* *head1*, *node*\* *head2*) {

*node*\* res = **new** *node*;

*node*\* p = res;

    while(*head1* !=NULL || *head2* != NULL) {

*node*\* newNode = **new** *node*;

        p -> next = newNode;

        p = p -> next;

        if(*head1* == NULL) {

            p -> data = *head2* -> data;

*head2* = *head2* -> next;

        }

        else if(*head2* == NULL) {

            p -> data = *head1* -> data;

*head1* = *head1* -> next;

        }

        else {

            p -> data = *head1* -> data + *head2* -> data;

*head1* = *head1* -> next;

*head2* = *head2* -> next;

        }

    }

    return res->next;

}

**Testing result:**

**A screen shot of a computer

Description automatically generated with medium confidence**