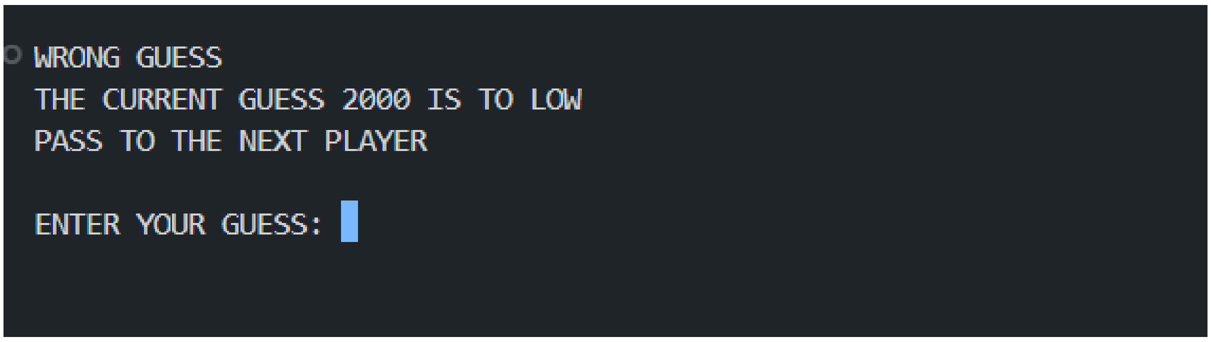
LAB 05

TASK 01:

```
# include <iostream>
# include <cstdlib>
# include <time.h>

using namespace std;

int input_num(void){
    int num;
    cout<<endl<<"ENTER YOUR GUESS: ";
    cin>>num;
    return num;
}

void guess_the_word(int random_num){
    int num=input_num();
    if(num==random_num){
        system("cls");
        cout<<"CONGRATULATIONS CORRECT GUESS "<<num<<endl;
    } else if(num<random_num){
        system("cls");
        cout<<"WRONG GUESS "<<endl<<"THE CURRENT GUESS "<<num<<" IS TO 
LOW"<<endl;
        cout<<"PASS TO THE NEXT PLAYER"<<endl;
        guess_the_word(random_num);
    } else{
        system("cls");
        cout<<"WRONG GUESS "<<endl<<"THE CURRENT GUESS "<<num<<" IS TO 
HIGH"<<endl;
        cout<<"PASS TO THE NEXT PLAYER"<<endl;
        guess_the_word(random_num);
    }

}



int main(){

    srand(time(0));
    guess_the_word(rand());


    return 0;
}
```

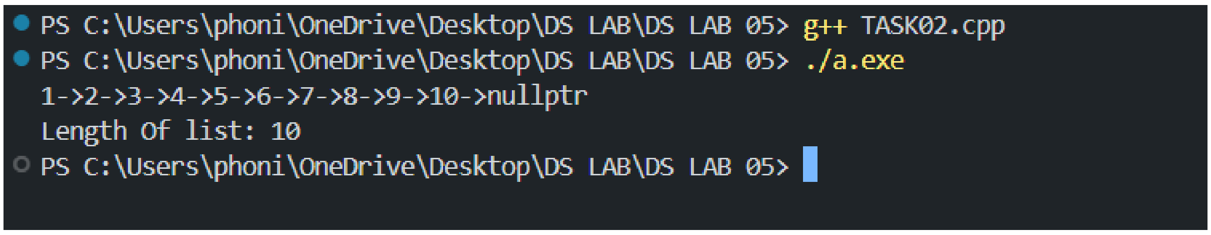


```
WRONG GUESS
THE CURRENT GUESS 2000 IS TO LOW
PASS TO THE NEXT PLAYER

ENTER YOUR GUESS:
```

TASK 02:

```cpp
# include <iostream>


using namespace std;

template <typename T>
class Node{
    public:
        T data;
        Node<T>* next=NULL;
        Node(int value=0):data(value){};
};


template <typename T>
class linklist{
    private:
        Node<T>* head;
        Node<T>* tail;
    public:
        linklist() : head(NULL), tail(head){}
        linklist(int value) : head(new Node<T>(value)), tail(head){}

        void add_node_at_tail(T value){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            tail->next=new_node;
            tail=new_node;
            tail->next=nullptr;
        }
        void add_node_at_pos(T value,int pos){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            if(pos==0){
                add_node_at_head(value);
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1 && curr!=tail){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr==tail){
                add_node_at_tail(value);
                return;
            }

            Node<T>* prev_node=curr;
            Node<T>* next_node=curr->next;

            prev_node->next=new_node;
            new_node->next=next_node;

        }

        void add_node_at_head(T value){
            Node<T>* new_node=new Node<T>(value);
            new_node->next=head;
            head=new_node;
            return;
        }

        void delete_node_at_pos(int pos){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }
            if(pos==0){
                delete_node_at_head();
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr->next==tail){
                delete_node_at_tail();
                return;
            }
            Node<T>* prev_node=curr;
            Node<T>* tmp_node=curr->next;
            Node<T>* next_node=curr->next->next;
            prev_node->next=next_node;

            delete tmp_node;
        }


        void delete_node_at_head(void){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }

            Node<T>* tmp_node=head;
            head=head->next;
            delete tmp_node;
        }

        void delete_node_at_tail(void){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }

            Node<T>* tmp_node=tail;
            Node<T>* curr=head;
            while(curr->next!=tail){
                curr=curr->next;
            }
            tail=curr;
            tail->next=nullptr;
            delete tmp_node;
```

```cpp
        }

        Node<T>* get_head(void){
            return head;
        }

        Node<T>* get_tail(void){
            return tail;
        }


        void set_tail(Node<T>* tail){
            this->tail=tail;
            return;
        }
        void set_head(Node<T>* head){
            this->head=head;
            return;
        }

        void display(void){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr->next!=nullptr){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
            cout<<"nullptr"<<endl;
            return;
        }
};


template <typename T>
int getLength(Node<T>* head){
    static int count=0;
    if(!head) return count;
    count++;
    return getLength(head->next);
}



int main(){
    linklist<int> list;
    list.add_node_at_tail(1);
    list.add_node_at_tail(2);
    list.add_node_at_tail(3);
    list.add_node_at_tail(4);
    list.add_node_at_tail(5);
    list.add_node_at_tail(6);
    list.add_node_at_tail(7);
    list.add_node_at_tail(8);
    list.add_node_at_tail(9);
    list.add_node_at_tail(10);

    list.display();
    cout<<"Length Of list: "<<getLength(list.get_head());

    return 0;
}
```

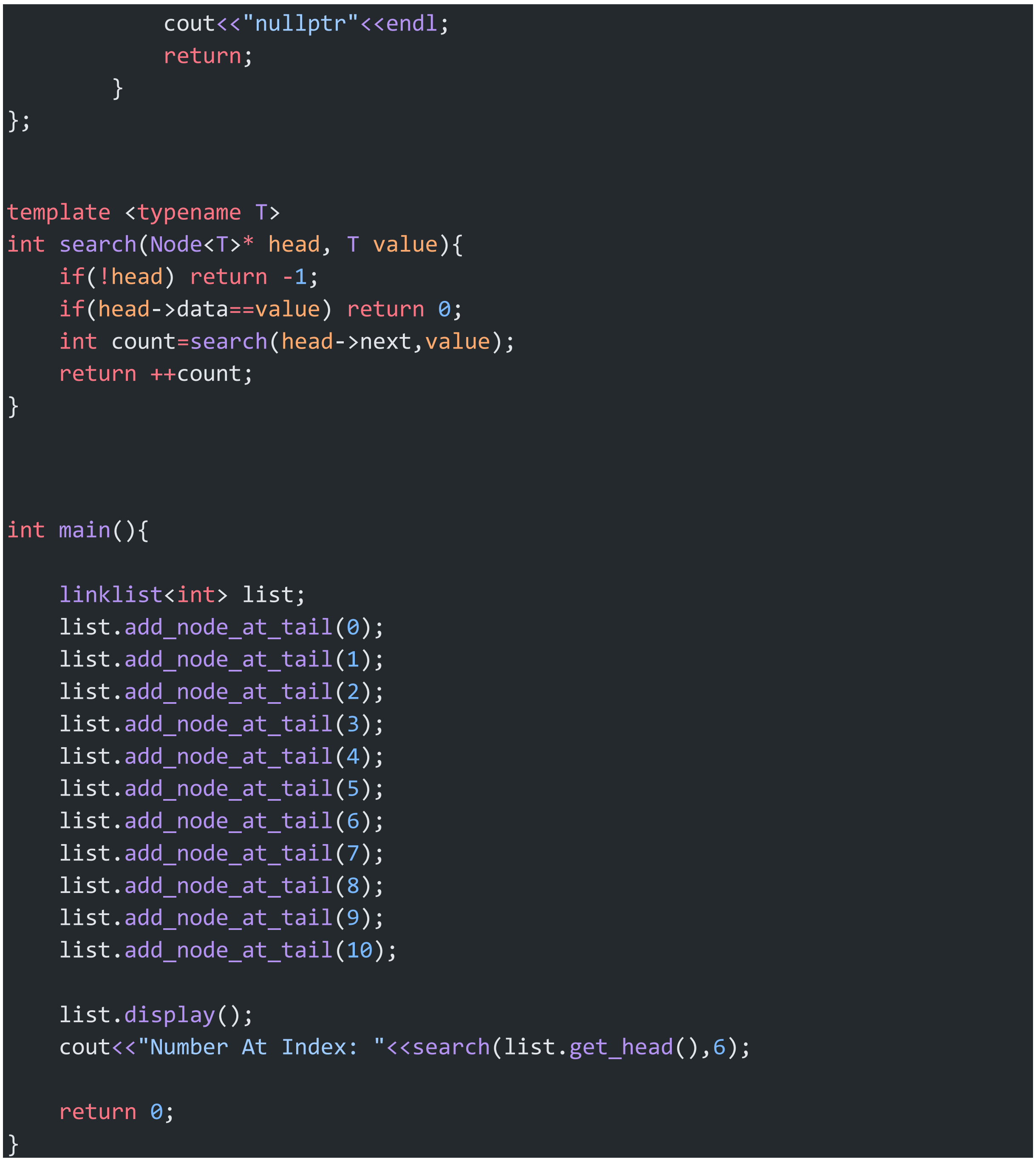


```
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05> g++ TASK02.cpp
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05> ./a.exe
1->2->3->4->5->6->7->8->9->10->nullptr
Length Of list: 10
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05>
```

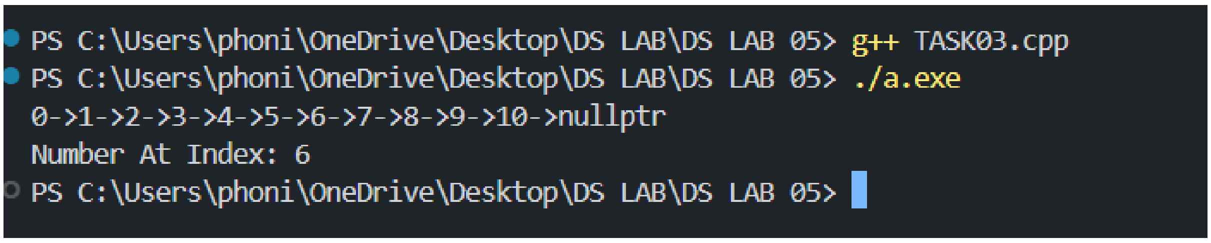
TASK 03:

```cpp
# include <iostream>

using namespace std;

template <typename T>
class Node{
    public:
        T data;
        Node<T>* next=NULL;
        Node(int value=0):data(value){};
};


template <typename T>
class linklist{
    private:
        Node<T>* head;
        Node<T>* tail;
    public:
        linklist() : head(NULL), tail(head){}
        linklist(int value) : head(new Node<T>(value)), tail(head){}

        void add_node_at_tail(T value){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            tail->next=new_node;
            tail=new_node;
            tail->next=nullptr;
        }

        void add_node_at_pos(T value,int pos){
            Node<T>* new_node=new Node<T>(value);
            if(head==NULL){
                head=new_node;
                tail=head;
                return;
            }
            if(pos==0){
                add_node_at_head(value);
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1 && curr!=tail){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr==tail){
                add_node_at_tail(value);
                return;
            }

            Node<T>* prev_node=curr;
            Node<T>* next_node=curr->next;

            prev_node->next=new_node;
            new_node->next=next_node;
        }

        void add_node_at_head(T value){
            Node<T>* new_node=new Node<T>(value);
            new_node->next=head;
            head=new_node;
            return;
        }

        void delete_node_at_pos(int pos){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }
            if(pos==0){
                delete_node_at_head();
                return;
            }
            int count=0; Node<T>* curr=head;
            while(count<pos-1){
                if(curr==tail){
                    cout<<"Error Wrong Position";
                    return;
                }
                curr=curr->next;
                count++;
            }
            if(curr->next==tail){
                delete_node_at_tail();
                return;
            }
            Node<T>* prev_node=curr;
            Node<T>* tmp_node=curr->next;
            Node<T>* next_node=curr->next->next;
            prev_node->next=next_node;

            delete tmp_node;
        }


        void delete_node_at_head(void){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }

            Node<T>* tmp_node=head;
            head=head->next;
```

```cpp
            delete tmp_node;
        }

        void delete_node_at_tail(void){
            if(head==NULL){
                cout<<"Linklist Does Not Rush"<<endl;
                return;
            }

            Node<T>* tmp_node=tail;
            Node<T>* curr=head;
            while(curr->next!=tail){
                curr=curr->next;
            }
            tail=curr;
            tail->next=nullptr;
            delete tmp_node;
        }

        Node<T>* get_head(void){
            return head;
        }

        Node<T>* get_tail(void){
            return tail;
        }


        void set_tail(Node<T>* tail){
            this->tail=tail;
            return;
        }
        void set_head(Node<T>* head){
            this->head=head;
            return;
        }

        void display(void){
            if(head==NULL){
                cout<<"Linklist is empty"<<endl;
                return;
            }
            Node<T>* curr=head;
            while(curr->next!=nullptr){
                cout<<curr->data<<"->";
                curr=curr->next;
            }
            cout<<curr->data<<"->";
```

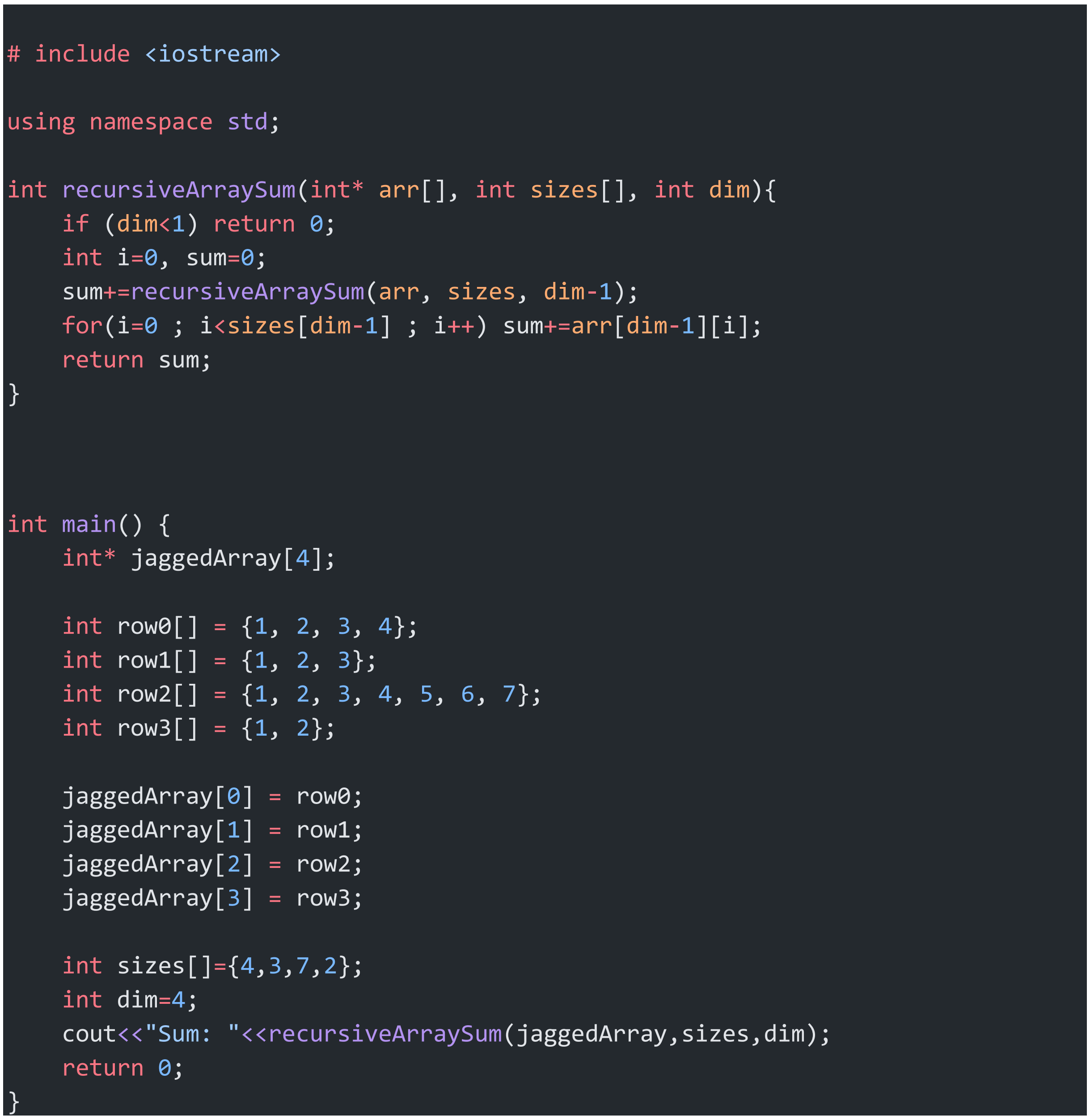


```cpp
            cout<<"nullptr"<<endl;
            return;
        }
};


template <typename T>
int search(Node<T>* head, T value){
    if(!head) return -1;
    if(head->data==value) return 0;
    int count=search(head->next,value);
    return ++count;
}



int main(){

    linklist<int> list;
    list.add_node_at_tail(0);
    list.add_node_at_tail(1);
    list.add_node_at_tail(2);
    list.add_node_at_tail(3);
    list.add_node_at_tail(4);
    list.add_node_at_tail(5);
    list.add_node_at_tail(6);
    list.add_node_at_tail(7);
    list.add_node_at_tail(8);
    list.add_node_at_tail(9);
    list.add_node_at_tail(10);

    list.display();
    cout<<"Number At Index: "<<search(list.get_head(),6);

    return 0;
}
```

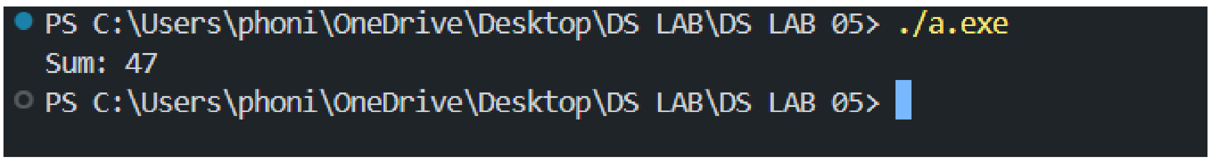


```
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05> g++ TASK03.cpp
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05> ./a.exe
0->1->2->3->4->5->6->7->8->9->10->nullptr
Number At Index: 6
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05>
```

TASK 04:



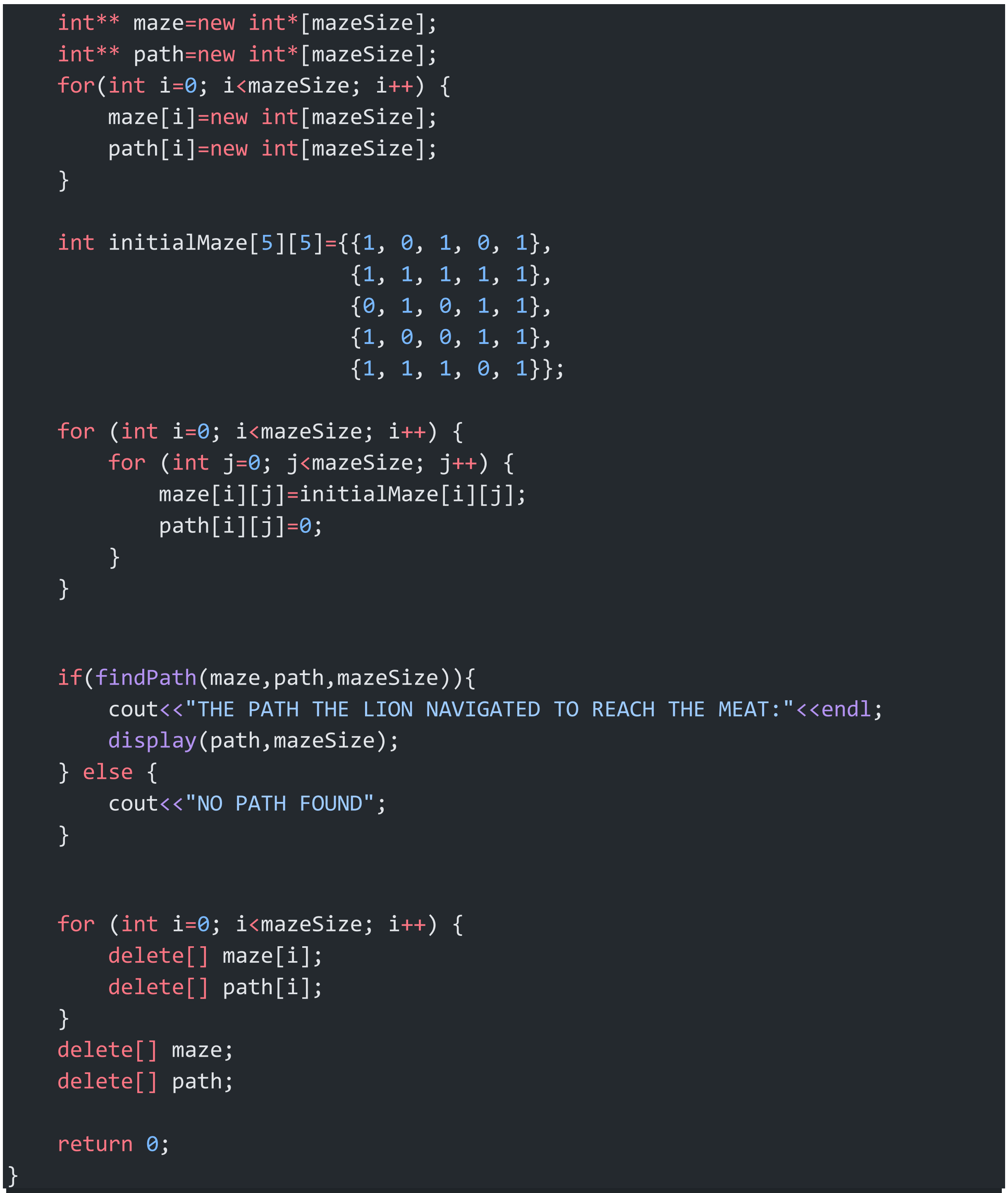
```
# include <iostream>

using namespace std;

int recursiveArraySum(int* arr[], int sizes[], int dim){
    if (dim<1) return 0;
    int i=0, sum=0;
    sum+=recursiveArraySum(arr, sizes, dim-1);
    for(i=0 ; i<sizes[dim-1] ; i++) sum+=arr[dim-1][i];
    return sum;
}



int main() {
    int* jaggedArray[4];

    int row0[] = {1, 2, 3, 4};
    int row1[] = {1, 2, 3};
    int row2[] = {1, 2, 3, 4, 5, 6, 7};
    int row3[] = {1, 2};

    jaggedArray[0] = row0;
    jaggedArray[1] = row1;
    jaggedArray[2] = row2;
    jaggedArray[3] = row3;

    int sizes[]={4,3,7,2};
    int dim=4;
    cout<<"Sum: "<<recursiveArraySum(jaggedArray,sizes,dim);
    return 0;
}
```



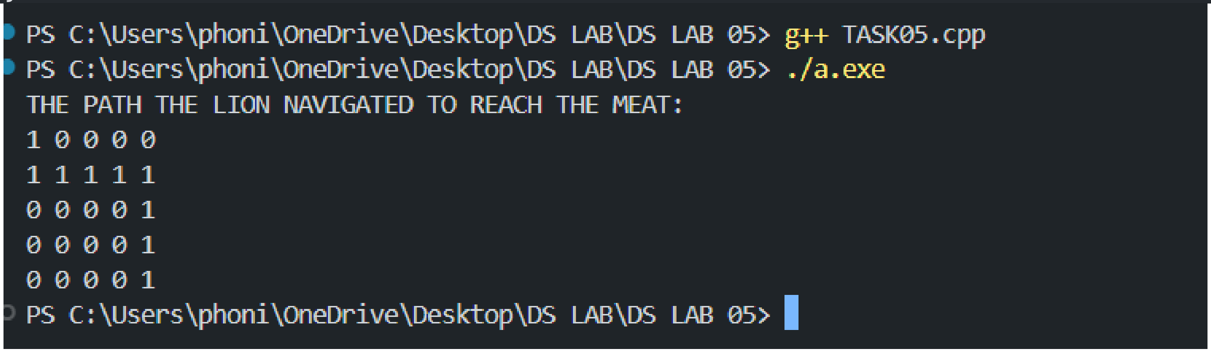
```
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05> ./a.exe
Sum: 47
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05>
```

TASK 05:

```
# include <iostream>

using namespace std;

bool isSafe(int** maze, int i, int j, int mazeSize){
    return (i>-1 && j>-1 && i<mazeSize && j<mazeSize && maze[i][j]==1);
}


bool findPath(int** maze,int** path, int mazeSize,int i=0,int j=0){

    if(i==mazeSize-1 && j==mazeSize-1){
        path[i][j]=1;
        return true;
    }

    if(isSafe(maze, i, j, mazeSize)){
        path[i][j]=1;

        //Forward Direction
        if(findPath(maze,path,mazeSize,i,j+1)) return true;
        //Downward Direction
        if(findPath(maze,path,mazeSize,i+1,j)) return true;
        //Backward Direction
        if(findPath(maze,path,mazeSize,i,j-1)) return true;
        //Upward Direction
        if(findPath(maze,path,mazeSize,i-1,j)) return true;

        path[i][j]=0;
        return false;
    }
    return false;
}

void display(int** matrix,int size){
    for(int i=0 ; i<size ; i++){
        for(int j=0 ; j<size ; j++){
            cout<<matrix[i][j]<<" ";
        }
        cout<<endl;
    }
}

int main(){

    int mazeSize=5;
```



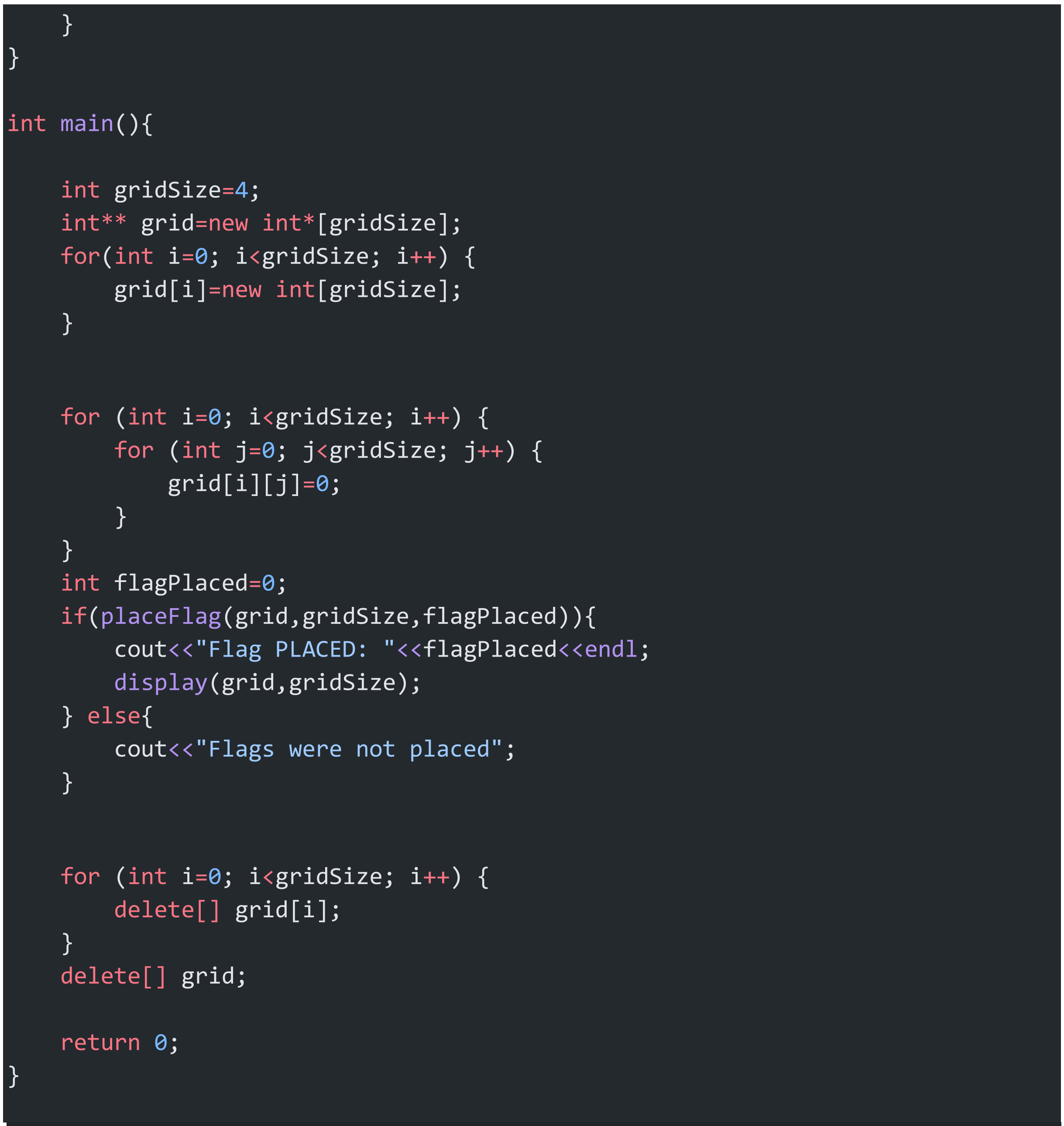
```
    int** maze=new int*[mazeSize];
    int** path=new int*[mazeSize];
    for(int i=0; i<mazeSize; i++) {
        maze[i]=new int[mazeSize];
        path[i]=new int[mazeSize];
    }

    int initialMaze[5][5]={{1, 0, 1, 0, 1},
                           {1, 1, 1, 1, 1},
                           {0, 1, 0, 1, 1},
                           {1, 0, 0, 1, 1},
                           {1, 1, 1, 0, 1}};

    for (int i=0; i<mazeSize; i++) {
        for (int j=0; j<mazeSize; j++) {
            maze[i][j]=initialMaze[i][j];
            path[i][j]=0;
        }
    }


    if(findPath(maze,path,mazeSize)){
        cout<<"THE PATH THE LION NAVIGATED TO REACH THE MEAT:"<<endl;
        display(path,mazeSize);
    } else {
        cout<<"NO PATH FOUND";
    }


    for (int i=0; i<mazeSize; i++) {
        delete[] maze[i];
        delete[] path[i];
    }
    delete[] maze;
    delete[] path;

    return 0;
}
```



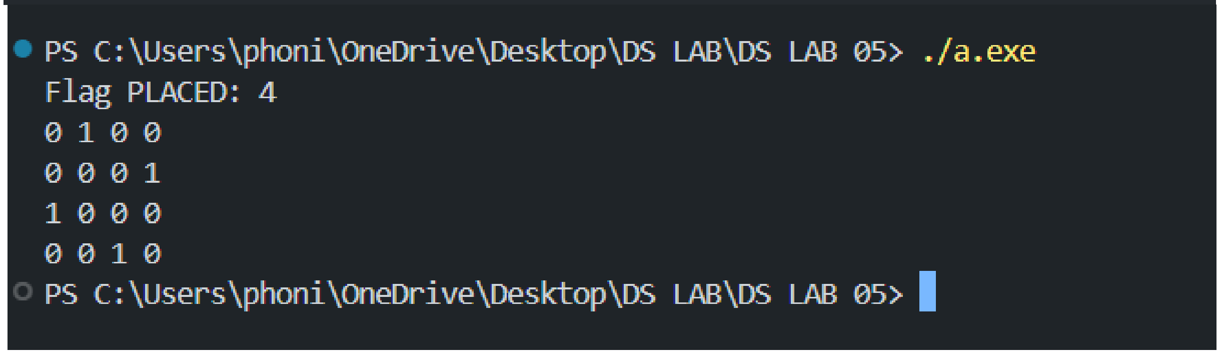
```
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05> g++ TASK05.cpp
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05> ./a.exe
THE PATH THE LION NAVIGATED TO REACH THE MEAT:
1 0 0 0 0
1 1 1 1 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05>
```

TASK 06:

```
# include <iostream>

using namespace std;

bool isSafe(int** grid, int i, int j, int gridSize){
    if(i<0 || j<0 || i>=gridSize || j>=gridSize || grid[i][j]==1) return
false;

    for(int k=0 ; k<gridSize ; k++){
        if(i+k<gridSize && j+k<gridSize && grid[i+k][j+k]==1) return false;
        if(i+k<gridSize && j-k>-1 && grid[i+k][j-k]==1) return false;
        if(j+k<gridSize && i-k>-1 && grid[i-k][j+k]==1) return false;
        if(j-k>-1 && i-k>-1 && grid[i-k][j-k]==1) return false;
        if(grid[i][k]==1) return false;
        if(grid[k][j]==1) return false;
    }

    return true;
}


bool placeFlag(int** grid, int gridSize, int &flagPlaced, int row=0, int
column=0) {

    if (row>=gridSize) return true;
    if (column>=gridSize) return false;

    if (isSafe(grid, row, column, gridSize)) {
        grid[row][column]=1;
        if (placeFlag(grid, gridSize, flagPlaced, row+1, 0)){
            flagPlaced++;
            return true;
        }
        grid[row][column]=0;
    }

    return placeFlag(grid, gridSize, flagPlaced, row, column+1);
}

void display(int** matrix,int size){
    for(int i=0 ; i<size ; i++){
        for(int j=0 ; j<size ; j++){
            cout<<matrix[i][j]<<" ";
        }
        cout<<endl;
```



```cpp
    }
}

int main(){

    int gridSize=4;
    int** grid=new int*[gridSize];
    for(int i=0; i<gridSize; i++) {
        grid[i]=new int[gridSize];
    }


    for (int i=0; i<gridSize; i++) {
        for (int j=0; j<gridSize; j++) {
            grid[i][j]=0;
        }
    }
    int flagPlaced=0;
    if(placeFlag(grid,gridSize,flagPlaced)){
        cout<<"Flag PLACED: "<<flagPlaced<<endl;
        display(grid,gridSize);
    } else{
        cout<<"Flags were not placed";
    }


    for (int i=0; i<gridSize; i++) {
        delete[] grid[i];
    }
    delete[] grid;

    return 0;
}
```



```
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05> ./a.exe
Flag PLACED: 4
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
PS C:\Users\phoni\OneDrive\Desktop\DS LAB\DS LAB 05>
```