**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ**
**Київський національний університет імені Тараса Шевченка**
**Кафедра програмних систем і технологій**

# Звіт з лабораторної роботи 1.7
# тема: «Черга. Стек.»

**Варіант 2**

**Виконав: студент групи ІПЗ - 12**
**Богатько Олександр**
**Геннадійович**
**Перевірила: викладач**
**Юрчук Ірина Аркадіївна**

**Київ 2020**

# “Task 1”

## 1. Стек (у вигляді масиву)

*Задача:*

Створити структуру даних Стек з набором операцій:

занесення елемента x в Стек S (Pop(S,x));

видалення елемента зі Стеку S і присвоювання його значення змінній x (Push(S));

перевірка Стеку на порожність (Empty(S));
читання елемента без його видалення зі Стеку (StackTop(S)); перевірка переповнення Стеку (Full(S)).

Для структури даних виділяється неперервна ділянка пам’яті.

```
#include <iostream>
using namespace std;
class Stack{
private:
   int* arr;
   int Max;
   int index;
public:
   Stack(int num){
      Max = 100;
      index = 0;
      if(num<=Max)
      {
         arr = new int[num];
         Max = num;
      }
      else cout<<”Stack is overflow”;
   }
   ~Stack()
   {
      delete[] arr;
   }
//Push()
bool Push(int n)
   {
      if(index == Max){
         return “Pushed”;
      }else
      {
         arr[index] = n;
         index++;
         return “Can`t push”;
      }
   }
```

Вивід консолі:

```
1.Push element
2.Pop element
3.Empty check
4.Show top element
5.Full check

0.Exit
Enter size of stack: 3
Enter command: 1
Enter element to Push: 1
1Enter command: 1
Enter element to Push: 2
1Enter command: 1
Enter element to Push: 3
1Enter command: 5
1
Enter command: 2

3Enter command: 4
Top element is - 2
Enter command: 5
0
```

```
    //Pop()
    int Pop()
    {
      if(index<0)
        return 0;
      else
      {
        return arr[--index];
      }
    }
    //Empty()
    bool Empty()
    {
      if(index<=0)
        return "True";
      else return "False";
    }
    //StackTop()
    int StackTop(){
      return index;
    }
    //IsFull()
    bool IsFull(){
      if(index == Max){
        return true;
      }else

        return false;
      }
    };
int main(){
    int s, ch, n;
    cout<<"1.Push element\n2.Pop element\n3.Empty check\n4.Show top element\n5.Full check\n\n0.Exit\n";
    cout<<"Enter size of stack: ";
    cin>>s;
    Stack st(s);
    while(true){
    cout<<"Enter command: ";
    cin>>ch;
    switch(ch){
      case 1:
        cout<<"Enter element to Push: ";
        cin>>n;
        cout<<st.Push(n);
        break;
      case 2:cout<<"\n"<<st.Pop();
        break;
      case 3:
        cout<<"\n"<<st.Empty();
        break;
      case 4:
        cout<<"Top element is – "<<st.StackTop()<<"\n";
        break;
      case 5:
        cout<<st.IsFull()<<"\n";
        break;
    }
    }
```

## “Task 2”

### 2. Стек (за допомогою вказівників/показників)

*Задача:*

Створити структуру даних Стек з набором операцій:

занесення елемента x в Стек S (Pop(S,x));

видалення елемента зі Стеку S і присвоювання його значення змінній x (Push(S));

перевірка Стеку на порожність (Empty(S));
читання елемента без його видалення зі Стеку (StackTop(S)); перевірка переповнення Стеку (Full(S)).

Структура даних створюється з використанням вказівників/показників.

```cpp
#include <iostream>
using namespace std;

template<typename T>
class Node
{
public:
    Node* pNext;
    T data;
    Node(T data = T(), Node* pNext = nullptr)
    {
        this->data = data;
        this->pNext = pNext;
    }
};
template<typename T>
class List
{
public:
    List();
    ~List();
    void pop_front();
    void push_back(T data);
    void clear();
    int GetSize() { return Size; }
    T& operator[](const int index);
    void push_front(T data);
    void insert(T data, int index);
    void removeAt(int index);
    void pop_back();
private:

    int Size;
    Node<T>* head;
};
template<typename T>
List<T>::List()
{
    Size = 0;
    head = nullptr;
}
template<typename T>
```

```cpp
List<T>::~List()
{
    clear();
}
template<typename T>
void List<T>::pop_front()
{
    Node<T>* temp = head;

    head = head->pNext;

    delete temp;

    Size--;

}
template<typename T>
void List<T>::push_back(T data)
{
    if (head == nullptr)
    {
        head = new Node<T>(data);
    }
    else
    {
        Node<T>* current = this->head;

        while (current->pNext != nullptr)
        {
            current = current->pNext;
        }
        current->pNext = new Node<T>(data);

    }

    Size++;
}
template<typename T>
void List<T>::clear()
{
    while (Size)
    {
        pop_front();
    }
}
template<typename T>
T& List<T>::operator[](const int index)
{
    int counter = 0;
    Node<T>* current = this->head;
    while (current != nullptr)
    {
        if (counter == index)
        {
            return current->data;
        }
        current = current->pNext;
        counter++;
    }
}
template<typename T>
void List<T>::push_front(T data)
{
    head = new Node<T>(data, head);
    Size++;
}
template<typename T>
void List<T>::insert(T data, int index)
{
    if (index == 0)
    {
        push_front(data);
    }
    else
    {
        Node<T>* previous = this->head;
        for (int i = 0; i < index - 1; i++)
        {
```

```
            previous = previous->pNext;
        }
        Node<T>* newNode = new Node<T>(data, previous->pNext);
        previous->pNext = newNode;
        Size++;
    }
}
template<typename T>
void List<T>::removeAt(int index)
{
    if (index == 0)
    {
        pop_front();
    }
    else
    {
        Node<T>* previous = this->head;
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext;
        }


        Node<T>* toDelete = previous->pNext;

        previous->pNext = toDelete->pNext;

        delete toDelete;

        Size--;
    }

}

template<typename T>
void List<T>::pop_back()
{
    removeAt(Size - 1);
}
int main()
{
    int Size,operation,element, opperation2;
    List<int> lst;

        cout << "1.Add Element\n";
        cout << "2.Show Stack\n";
        cout << "3.Delete Element\n";
        cout <<"Enter size:";
        cin>>Size;
    while (true)
    {
        cout<<"Enter command: ";
        cin >> operation;
        switch(operation){
        case 1:
            if (lst.GetSize() == Size)
            {
                cout << "Stack is full" << endl;
            }
            else
            {
                cout << "Enter element:";
                cin >> element;
                lst.push_front(element);
            }
        break;
        case 2:
            if (lst.GetSize() == 0)
            {
                cout << "Stack empty" << endl;
            }
            else
            {
                for (int i = 0; i < lst.GetSize(); i++)
                {
                    cout << lst[i] << " ";
                }
            }
```

```
            cout << endl;
        break;
        case 3:
            if (lst.GetSize() == 0)
            {
                cout << "Stack empty" << endl;
            }
            else
                lst.pop_back();
        break;}
    }
    return 0;
}
```

Вивід консолі:

```
"/Users/aleksandrbogatko/Desktop/Alexar
1.Add Element
2.Show Stack
3.Delete Element
Enter size:3
Enter command: 1
Enter element:9
Enter command: 1
Enter element:6
Enter command: 1
Enter element:3
Enter command: 2
3 6 9
Enter command: 1
Stack is full
Enter command: 3
Enter command: 2
3 6
Enter command:
```

## “Task 3”

### Черга (у вигляді масиву)

*Задача:*

Створити структуру даних Черга з набором операцій:

занесення елемента x в Чергу Q (Insert(Q,x));

видалення елемента з Черги Q і присвоювання його значення змінній x (Remove(Q));

перевірка Черги Q на порожність (Empty(S)) ;

перевірка переповнення Черги Q (Full(Q)).
Для структури даних виділяється неперервна ділянка пам’яті.

Вивід консолі:

```
1.Push
2.Pop
3.IsEmpty
4.IsFull
Enter size: 3

Choose command:1
Enter element to push: 9

Choose command:1
Enter element to push: 6

Choose command:1
Enter element to push: 3

Choose command:4
1

Choose command:2
9
Choose command:4
0

Choose command:2
6
Choose command:2
3
Choose command:3
1
Choose command:
```

```
#include <iostream>
#include <new>
using namespace std;

template <typename T>
class Queue
{
private:
    T* p;
    int count;
public:
    Queue()
    {
        count = 0;
    }
    Queue(const Queue& obj)
    {
        count = obj.count;
        try {
            p = new T[count];
            for (int i = 0; i < count; i++)
                p[i] = obj.p[i];
        }
        catch (bad_alloc e)
        {
            cout << e.what() << endl;
            count = 0;
        }
    }
    void Push(T item)
    {
        T* p2;
        p2 = p;
        try {
            p = new T[count + 1];
            for (int i = 0; i < count; i++)
                p[i] = p2[i];
            p[count] = item;
            count++;
            if (count > 1)
                delete[] p2;
        }
        catch (bad_alloc e)
        {
            cout << e.what() << endl;
            p = p2;
        }
    }
    T Pop()
    {
        if (count == 0)
            return 0;
        T item;
        item = p[0];
        try {
```

```
            T* p2;
            p2 = new T[count - 1];
            count--;
            for (int i = 0; i < count; i++)
                p2[i] = p[i + 1];
            if (count > 0)
                delete[] p;
            p = p2;
            return item;
        }
        catch (bad_alloc e)
        {
            cout << e.what() << endl;
            return 0;
        }
    }
    Queue& operator=(const Queue& obj)
    {
        T* p2;
        try {
            p2 = new T[obj.count];
            if (count > 0)
                delete[] p;
            p = p2;
            count = obj.count;
            for (int i = 0; i < count; i++)
                p[i] = obj.p[i];
        }
        catch (bad_alloc e)
        {
            cout << e.what() << endl;
        }
        return *this;
    }
    ~Queue()
    {
        if (count > 0)
            delete[] p;
    }
    bool IsEmpty()
    {
        return count == 0;
    }
    bool IsFull(int x)
    {   if (x == count)
        return true;
    else return false;
    }
};
int main()
{ int s, n, ch;
    Queue<int> q;
    cout<<"1.Push\n2.Pop\n3.IsEmpty\n4.IsFull\n";
    cout<<"Enter size: ";
    cin>>s;
    while(true){
    cout<<"\nChoose command:";
    cin>>ch;
    switch (ch){
        case 1: cout<<"Enter element to push: ";
            cin>>n;
            q.Push(n);
            break;
        case 2:cout<<q.Pop();
            break;
        case 3:cout<<q.IsEmpty();
            break;
        case 4:cout<<q.IsFull(s)<<"\n";
            break;
        }
    }
}
```

# “Task 4”

### Черга (за допомогою вказівників/показників)

*Задача:*

Створити структуру даних Черга з набором операцій:
- занесення елемента x в Чергу Q (Insert(Q,x));
- видалення елемента з Черги Q і присвоювання його значення

змінній x (Remove(Q));
- перевірка Черги Q на порожність (Empty(S)) ; - перевірка переповнення Черги Q (Full(Q)).

Структура даних створюється з використанням вказівників/показників.

```
#include <iostream>
using namespace std;

template<typename T>
class Node
{
public:
    Node* pNext;
    T data;

    Node(T data = T(), Node* pNext = nullptr)
    {
        this->data = data;
        this->pNext = pNext;
    }
};
template<typename T>
class List
{
public:
    List();
    ~List();
    void pop_front();

    void push_back(T data);

    void clear();

    int GetSize() { return Size; }

    T& operator[](const int index);

    void push_front(T data);

    void insert(T data, int index);

    void removeAt(int index);

    void pop_back();
private:
    Node<T> *head;
    int Size;
};
template<typename T>
List<T>::List()
```

```
{
    Size = 0;
    head = nullptr;
}


template<typename T>
List<T>::~List()
{
    clear();
}


template<typename T>
void List<T>::pop_front()
{
    Node<T>* temp = head;

    head = head->pNext;

    delete temp;

    Size--;

}

template<typename T>
void List<T>::push_back(T data)
{
    if (head == nullptr)
    {
        head = new Node<T>(data);
    }
    else
    {
        Node<T>* current = this->head;

        while (current->pNext != nullptr)
        {
            current = current->pNext;
        }
        current->pNext = new Node<T>(data);

    }

    Size++;
}

template<typename T>
void List<T>::clear()
{
    while (Size)
    {
        pop_front();
    }
}


template<typename T>
T& List<T>::operator[](const int index)
{
    int counter = 0;
```

```
    Node<T>* current = this->head;

    while (current != nullptr)
    {
        if (counter == index)
        {
            return current->data;
        }
        current = current->pNext;
        counter++;
    }
}

template<typename T>
void List<T>::push_front(T data)
{
    head = new Node<T>(data, head);
    Size++;
}

template<typename T>
void List<T>::insert(T data, int index)
{

    if (index == 0)
    {
        push_front(data);
    }
    else
    {
        Node<T>* previous = this->head;

        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext;
        }

        Node<T>* newNode = new Node<T>(data, previous->pNext);

        previous->pNext = newNode;

        Size++;
    }
}
template<typename T>
void List<T>::removeAt(int index)
{
    if (index == 0)
    {
        pop_front();
    }
    else
    {
        Node<T>* previous = this->head;
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext;
        }
        Node<T>* toDelete = previous->pNext;
        previous->pNext = toDelete->pNext;
        delete toDelete;
```

```
        Size--;
    }
}
template<typename T>
void List<T>::pop_back()
{
    removeAt(Size - 1);
}
int main()
{
    int SizeLimit = 10;
    int peration,element, opperation2;
    List<int> lst;
        cout << "1.Add Element\n2.Show Queue\n3,Delete Element\n";
        cout<<"Enter size: ";
        cin>>SizeLimit;
        while (true)
        {cout<<"Enter command: ";
        cin >> peration;
        switch(peration){
            case 1:
                if (lst.GetSize() == SizeLimit)
                {
                    cout << "Stack is full!\n";
                }
                else {
                    cout << "Enter element:";
                    cin >> element;
                    lst.push_front(element);
                }
                break;
            case 2:

                if (lst.GetSize() == 0)
                {
                    cout << "Stack is empty!\n";
                }
                else
                {
                    for (int i = 0; i < lst.GetSize(); i++)
                    {
                        cout << lst[i] << " ";
                    }
                }
                cout << endl;
                break;
            case 3:

                if (lst.GetSize() == 0)
                {
                    cout << "Stack us empty!\n";
                }
                else
                    lst.pop_back();
                break;
        }
    }
```

Вивід консолі:

```
1.Add Element
2.Show Queue
3,Delete Element
Enter size: 3
Enter command: 1
Enter element:5
Enter command: 1
Enter element:3
Enter command: 1
Enter element:2
Enter command: 1
Stack is full!
Enter command: 2
2 3 5
Enter command: 3
Enter command: 2
2 3
Enter command:
```

## Висновок

В цій лаброраторній роботі навчилися працювати зі стеком,та чергою.
Навчилися створювати чергу та стек на мові програмування С++, за допомогою динамічних масивів та вказівників.