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

Звіт з лабораторної роботи 1.8
тема: «Стек»

Варіант 2

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

Київ 2020

“Task 1”

1. Лінійний список (у вигляді масивів)

Задача:

Створюється структура даних лінійний однозв'язний список з необхідним набором операцій:

- створити порожній список;
- додати елемент до списку;
- додати елемент після заданого;
- видалити елемент зі списку;
- знайти елемент у списку;
- сортування елементів списку;
- додати елемент в відсортований список; - перегляд всіх елементів списку;
- перегляд всіх елементів списку по циклу.

```
1. Add element
2. Insert element
3. Delete element
4. Delete end element
5. Find element
6. Sort list
7. Insert element in sorted list
8. Show list
0. Exit
Enter size: 4

Choose action: 1
Enter element:9
Element pushed
Choose action: 1
Enter element:6
Element pushed
Choose action: 1
Enter element:8
Element pushed
Choose action: 2
Enter element: 1
Enter position: 3

Choose action: 8
9   6   1   8

Choose action: 6
Result:
1   6   8   9

Choose action: 3
Enter element:1

Choose action: 8
6   8   9

Choose action: 0
Choose command from 1 to 8.Program ended with exit code: 0
```

Під структуру даних виділяється неперервна ділянка пам'яті.

Вказівка:

При роботі зі створеною структурою необхідно аналізувати ситуацію, коли число елементів в списку перевищить допустиму кількість. У цій ситуації необхідно виділяти додатково необхідну кількість пам'яті, якщо це можливо.

```
#include <iostream>
using namespace std;
struct List
{
private:
    int *list = nullptr;
    int size, num;
public:
    List(int s)
    {
```

```
    list = new int(s);
    size = s;
    num = 0;
}
void Push(int el)
{
    if(num < size)
    {
        list[num] = el;
        num++;
        cout<<"Element pushed";
    }else
    {
        cout<<"No space in list";
    }
}
void Insert(int el,int dx)
{
    if(num <= size - 1){
        for(int i = num;i>(dx-1);i--)
        {
            list[i] = list[i-1];

        }
            list[dx-1] = el;
            num++;
        }
    }
void Print()
{
    for(int i = 0; i<num; i++)
    {
        cout<<list[i]<<"\t";

    }
    cout<<endl;
}
void Pop()
{
    if(num>0)
    {
        list[num] = 0;
        num--;
    }
}
void Delete(int el)
{
    if(num>0)
    {
        int pos = 0, ok = 0;
        for(int i=0; i<num; i++)
        {
            if(el == list[i])
            {
                pos = i;
                ok++;
            }
        }
        if(ok == 0)
        {
            cout<<"This element not in list";
        }
        for(int i = pos;i<num;i++)
        {
            list[i] = list[i+1];
        }
        list[num] = 0;
        num--;
    }
```

```
    }
    void Find(int el)
    {
        int ok = 0;
        for (int i = 0; i<num;i++)
        {
            if(el == list[i])
            {
                cout<<el<<"on the"<<i+1<<"position"<<endl;
                ok++;
            }
        }
        if(ok == 0)
        {
            cout<<"This element not in list";
        }
    }
    void Sort()
    {
        int k = 1, buf;
        while(k != 0)
        {
            k = 0;
            for(int i = 1;i < num; i++)
            {
                if(list[i] < list[i-1])
                {
                    buf = list[i-1];
                    list[i-1] = list[i];
                    list[i] = buf;
                    k++;
                }
            }
        }
    }
    void PushSort(int el)
    {
        Sort();
        Push(el);
        Sort();
    }
};
int main() {
    int op = 1, el = 0, dx = 0, size;
    string name;
    cout<<"1. Add element\n2. Insert element\n3. Delete element\n4. Delete end
element\n"
    "5. Find element\n6. Sort list\n7. Insert element in sorted list\n8. Show list\n0.
Exit\n";
    cout<<"Enter size: ";
    cin>>size;
    List lst = List(size);
    while(op!=0)
    {
        cout<<"\nChoose action: ";
        cin>>op;
        switch(op)
        {
            case 1: cout<<"Enter element:";
                cin>>el;
                lst.Push(el);
                break;
            case 2:cout<<"Enter element: ";
                cin>>el;
                cout<<"Enter position: ";
                cin>>dx;
                lst.Insert(el, dx);
                break;
            case 3:cout<<"Enter element:";
```

```
                cin>>el;
                lst.Delete(el);
                break;
            case 4: lst.Pop();
                break;
            case 5:cout<<"Enter element: ";
                lst.Find(el);
                break;
            case 6:lst.Sort();
                cout<<"Result: \n";
                lst.Print();
                break;
            case 7:cout<<"Enter element:";
                cin>>el;
                lst.PushSort(el);
                cout<<"Result:\n";
                lst.Print();
                break;
            case 8: lst.Print();
                break;
            default: cout<<"Choose command from 1 to 8.";break;
        }
    }
    return 0;
}
```

“Task 2”

2. Лінійний список (за допомогою вказівників/показників)

Задача:

Створюється структура даних лінійний однозв'язний список з необхідним набором операцій:

- створити порожній список; - додати елемент до списку;

- додати елемент до списку після заданого елементу; - видалити елемент зі списку;
- знайти елемент у списку;
- сортування елементів списку;

- додати елемент в відсортований список;

- перегляд всіх елементів списку.
Структура даних створюється з використанням вказівників/показників.

Вказівка:

При роботі зі створеною структурою необхідно в разі видалення елемента звільняти виділену під його зберігання пам'ять.

```
#include <iostream>
using namespace std;
template <typename T>
struct List
{
private:
   template <typename V>
   struct Node
   {
   public:
      V data;
      Node *next;
      Node(V data = V(), Node *next = nullptr)
      {
         this->data = data;
         this->next = next;
      };
   };
   int size;
   Node<T> *start;
public:
   List()
   {
      size = 0;
      start = nullptr;
   };
   void push_back(T data)
   {
      if (start == nullptr)
      {
         start = new Node<T>(data);
      }
      else
      {
         Node<T> *curr = this->start;
         while (curr->next != nullptr)
         {
            curr = curr->next;
         }
         curr->next = new Node<T>(data);
      }
      size++;
   }
   int getSize() {return size;}
   T& operator[](const int index)
   {
      int count = 0;
      Node<T> *curr = this->start;
      while (curr != nullptr)
      {
         if (count == index)
         {
            return curr->data;
         }
         curr = curr->next;
         count++;
      }
       void pop_front()
```

```
1. Add element
2. Insert element
3. Delete element
4. Delete element from the end of list
5. Find element
6. Sort list
7. Insert element in sorted list
8 Show elements
0. Exit
Enter size of list: 4
Choose operation: 1
Enter element: 9
Element has been pushed
Choose operation: 1
Enter element: 6
Element has been pushed
Choose operation: 1
Enter element: 7
Element has been pushed
Choose operation: 2
Enter element: 8
Enter the position to insert: 3
Choose operation: 8
9   6   8   7
Choose operation: 3
Enter element: 9
Choose operation: 8
6   8   7
Choose operation: 6
Result:
6   7   8
Choose operation: 7
Enter element: 2
Element has been pushed
Result:
2   6   7   8

Choose operation: 0

Process finished with exit code 0
```

```
{
   Node<T> *temp = start;
   start = start->next;
   delete temp;
   size--;
}
void clear()
{
   while(size)
   {
      pop_front();
   }
}
void push(T data)
{
   start = new Node<T> (data, start);
   size++;
}
void pop()
{
   remove(size - 1);
}
void insert(T data, int index)
{
   if (index == 0)
   {
      push(data);
   }
   else
   {
      Node<T> *prev = this->start;
      for (int i  = 0; i < index - 1; i++)
      {
         prev = prev->next;
      }
      Node<T> *newN = new Node<T>(data, prev->next);
      prev->next = newN;
   }
   size++;
}
void remove(int index)
{
   if (index == 0)
   {
      pop_front();
   } else
   {
      Node<T> *prev = this->start;
      for (int i = 0; i < index - 1; i++)
      {
         prev = prev->next;
      }
      Node<T> *toDel = prev->next;
      prev->next = toDel->next;
      delete toDel;
      size--;
   }
}
void sort()
{
   for(int i = 0; i < size; i++)
   {
      for(int j = 0; j < i; j++)
```

```
            {
                if((*this)[j] > (*this)[j + 1])
                {
                    T buf = (*this)[j];
                    (*this)[j] = (*this)[j + 1];
                    (*this)[j + 1] = buf;
                }
            }
        }
    }
        void push_sort(T data)
    {
        sort();
        push_back(data);
        sort();
    }
    void find(T data)
    {
        int ok = 0;
        for (int i = 0; i < size; i++)
        {
            if ((*this)[i] == data)
            {
                cout << (*this)[i] << "on the" << i << "position";
                ok++;
            }
        }
        if (ok == 0)
        {
            cout << "Element not in list";
        }
    }
    void print()
    {
        for (int i = 0; i < size; i++)
        {
            cout << (*this)[i] << '\t';
        }
        cout << endl;
    }
    ~List()
    {
        clear();
    }
    };
int main() {
    int op = 1, el, dx;
    string name;
    cout << "1. Add element\n2. Insert element\n3. Delete element\n"
            "4. Delete element from the end of list\n5. Find element\n"
            "6. Sort list\n7. Insert element in sorted list\n8 Show elements\n0. Exit\n";
    List <int> f;
    while (op != 0)
    {
        cout << "Choose command: ";
        cin >> op;
        switch (op)
        {
            case 1: cout << "Enter element: ";
                cin >> el;
                f.push_back(el);
                break;
            case 2: cout << "Enter element: ";
```

```
            cin >> el;
            cout << "Enter the index to insert: ";
            cin >> dx;
            f.insert(el, dx - 1);
            break;
        case 3: cout << "Enter index of element: ";
            cin >> dx;
            f.remove(dx - 1);
            break;
        case 4: f.pop();
            break;
        case 5: cout << "Enter element: ";
            cin >> el;
            f.find(el);
            break;
        case 6: f.sort();
            cout << "Result: " << endl;
            f.print();
            break;
        case 7: cout << "Enter element: ";
            cin >> el;
            f.push_sort(el);
            cout << "Result: " << endl;
            f.print();
            break;
        case 8: f.print();
        default: break;
      }
   }
   return 0;
}
};
```

«Task 3»

3. Структура даних нелінійний список

Задача:

Створити мультисписок. Пов'язувати вузли можна за допомогою двох полів-покажчиків в двох незалежних списках, по одному на кожне поле. Наприклад, одне поле пов'язує вузли в порядку їх створення. Друге поле, можливо, в відсортованому порядку. Це дозволяє вибрати порядок перегляду елементів.

Вказівка:

За допомогою генератора випадкових чисел створити мультисписок зі 100 елементів. Впорядкувати його за зростанням. Мати можливість переглядати елементи в порядку зростання і порядку генерації.

```
include <iostream>
#include <iomanip>
using namespace std;
template<typename V>
struct Node
{
```

```
public:
  Node(V data = V(), Node<V>* next = nullptr) : Data(data), Next(next) {};
  Node<V>* Next;
  V Data;
};
template <typename T>
class List
{
private:
  Node<T>* start;
  Node<T>* last;
  int count{};
public:
  List()
  {
    count = 0;
    start = nullptr;
    last = nullptr;
  };
  Node<T>* operator[](int index) __const
  {
    Node<T>* element = start;
    for(int i = 0; i < index; i++)
      element = element -> Next;
    return element;
  };
  List(const List<T>& list) : List()
  {
    for(int i = 0; i < list.Count(); i++)
      push_back((list[i]) -> Data);
  };

  bool push_back(T element)
  {
    Node<T>* newElement = new Node<T>(element, nullptr);
    if(count == 0)
    {
      last = newElement;
      start = last;
    }
    else
    {
      last->Next = newElement;
      last = newElement;
    }
    ++count;
    return true;
  };
  void sort()
  {
    for(int i = 0; i < count; i++)
      for(int j = 0; j < count - 1; j++)
        if( ((*this)[j]) -> Data > ((*this)[j + 1]) -> Data )
        {
          T element = ((*this)[j]) -> Data;
          ((*this)[j]) -> Data = ((*this)[j + 1]) -> Data;
          ((*this)[j + 1]) -> Data = element;
        }
  };
  int Count() __const
  {
    return count;
  };
```

```
Original list:
  30  91  15  72  61  41  10  37  98  41  94  80  26  96  10  88  59   5  84  14
  26  13  83  54  87  41  46  37  53  19  60  51  54  78  94  76  91  56  60  94
  62  80   7  37  66  90   7  12  58  26  26  40  14  36  45  14  99  45  87  20
  46  18  97  75  12  28   3  71  96  11   3  29  12  52  31  83  92  22  68  65
  53  73  76  94  91  76  91  79  61  43  18  63  74   3  29  50  36  59  69  95

Sorted list:
   3   3   3   5   7   7  10  10  11  12  12  12  13  14  14  14  15  18  18  19
  20  22  26  26  26  26  28  29  29  30  31  36  36  37  37  37  40  41  41  41
  43  45  45  46  46  50  51  52  53  53  54  54  56  58  59  59  60  60  61  61
  62  63  65  66  68  69  71  72  73  74  75  76  76  76  78  79  80  80  83  83
  84  87  87  88  90  91  91  91  91  92  94  94  94  94  95  96  96  97  98  99

Process finished with exit code 0
```

```
  void Print()
  {
     for(int i = 0; i < count; i++)
     {
        cout << setw(4) << ((*this)[i]) -> Data;
        if (i % 20 == 19)
        {
           cout << endl;
        }
     }
  };
};
int main()
{
  srand(0);
  List<int> list{};
  for(int i = 0; i < 100; i++)
     list.push_back(rand() % 100);
  List<int> copy(list);
  list.sort();
  cout << "Original list: " << endl;
  copy.Print();
  cout << endl << "Sorted list: " << endl;
  list.Print();
  return 0;
}
```