Списки. Лабораторная работа №2

Односвязный линейный список

- Каждый элемент списка имеет указатель на следующий элемент
- Последний элемент списка указывает на NULL
- В односвязном списке можно передвигаться только в сторону конца списка
- Узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно

Односвязный линейный список

```
struct Student{
   char Name[20];
   char NameLast[30];
   int Age;
   char School[30];

   void Input(Student &student);        //Функция ввода данных в структуру
   Student *Next;                       //Адрес на следующий элемент
};

class List{
   Student *Head;                  //Указатель на начало списка
public:
   List():Head(NULL){};            //Конструктор по умолчанию (Head=NULL)
   ~List();                        //Прототип деструктора
   void Add(Student &student);     //Прототип функции добавления элемента в
     список
   void Show();                    //Прототип функции вывода списка на экран
};
```

```
void Student::Input(Student &student){
   cout << endl;
   cout << "Имя:  ";
   cin.getline(Name,20);
   cout << "Фамилия:  ";
   cin.getline(NameLast,30);
   cout << "Полных лет ";
   cin >> Age;
   cin.ignore();                    //Игнорируем символ «Enter»
   cout << "Где учится ";
   cin.getline(School, 30);
}

 List::~List() {//Деструктор класса List
while (Head != NULL)              //Пока по адресу есть хоть что-то
      {
         Student *temp = Head->Next;     //Сразу запоминаем указатель на адрес
     следующего элемента структуры
         delete Head;                 //Освобождаем память по месту начала списка
         Head = temp;                  //Меняем адрес начала списка
      }
}
```

```
void List::Add(Student &student){
    Student *temp = new Student;          //Выделение памяти
    под новую структуру
    temp->Next = Head;                    //Указываем, что адрес
    следующего элемента это начало списка

     strcpy(temp->Name, student.Name);
     strcpy(temp->NameLast, student.NameLast);
     temp->Age = student.Age;
     strcpy(temp->School, student.School);
    Head = temp;                          //Смена адреса начала
    списка
}
```

```
void List::Show(){
 Student *temp = Head;              //Объявляем указатель
    и изначально он указывает на начало
    while (temp != NULL)            //Пока по адресу на
    начало хоть что-то есть
    {
    //Выводим все элементы структуры
        cout << temp->Name << "\t\t";
        cout << temp->NameLast <<"\t\t";
        cout << temp->Age << "\t\t";
        cout << temp->School << endl;
        temp = temp->Next;          //Указываем на
    следующий адрес из списка
    }
    cout << endl;
}
```

```
int main (){
  Student student;                    //Объявили переменную, тип которой Студент
  int N;                              //Объявили переменную - число студентов
  List lst;                           //Объявили переменную типа Список. Она выступает
      как контейнер данных

    cout << "N = ";
    cin >> N;                         //Ввели число студентов
    cin.ignore();                      //Игнорируем клавишу Enter

    for (int i=0; i<N; i++)
    {
     student.Input(student);           //Передаем в функцию заполнения
       переменную студент
     lst.Add(student);                 //Добавляем заполненную структуру в список
    }

    cout << endl;
    lst.Show();                       //Показываем список на экране
  cin.ignore().get();
  return 0;
}
```

Задание №1.

Написать функцию Remove() – метод класса List, которая удаляет последний узел.

Вывести 5 элементов списка, после чего удалить последние два и снова вывести.

Двусвязный линейный список

```
struct Node {                    //Структура, являющаяся звеном списка
    int x;                       //Значение x будет передаваться в список
    Node *Next, *Prev;           //Указатели на адреса следующего и
     предыдущего элементов списка
 };


class List {                     //Создаем тип данных Список
Node *Head, *Tail;               //Указатели на адреса начала списка и его
     конца
 public:
    List():Head(NULL),Tail(NULL){};    //Инициализируем адреса как пустые
    ~List();                     //Прототип деструктора
    void Show();                 //Прототип функции отображения списка на
     экране
    void Add(int x);             //Прототип функции добавления элементов в
     список
 };
```

```
List::~List() {
while (Head)                  //Пока по адресу на начало списка что-то есть
    {
       Tail = Head->Next;          //Резервная копия адреса следующего звена списка
       delete Head;                //Очистка памяти от первого звена
       Head = Tail;                //Смена адреса начала на адрес следующего элемента
    }
 }


void List::Add(int x) {
Node *temp = new Node;             //Выделение памяти под новый элемент структуры
  temp->Next = NULL;               //Указываем, что изначально по следующему адресу пусто
  temp->x = x;                     //Записываем значение в структуру

  if (Head != NULL)                //Если список не пуст
  {
     temp->Prev = Tail;            //Указываем адрес на предыдущий элемент в соотв. поле
     Tail->Next = temp;            //Указываем адрес следующего за хвостом элемента
     Tail = temp;                  //Меняем адрес хвоста
}
  else //Если список пустой
  {
     temp->Prev = NULL;            //Предыдущий элемент указывает в пустоту
     Head = Tail = temp;           //Голова = Хвост = тот элемент, что сейчас добавили
  }
 }
```

```
void List::Show() {
Node *temp=Tail;                //Временный указатель на адрес последнего
     элемента
    while (temp != NULL)            //Пока не встретится пустое значение
    {
       cout << temp->x << " ";      //Выводить значение на экран
       temp = temp->Prev;           //Указываем, что нужен адрес предыдущего
     элемента
    }
    cout << "\n";
}


int main (){
 system("CLS");
 List lst;
 lst.Add(100);
 lst.Add(200);
 lst.Add(900);
 lst.Add(888);
 lst.Show(); //Отображаем список на экране
  system("PAUSE");
    return 0;
}
```

Задание №2.

Написать функцию Remove() – метод класса List, которая удаляет последний узел.

Вывести 5 элементов списка, после чего удалить последние два и снова вывести.

void Merge(int *A, int first, int last)

```
void Merge(int *A, int first, int last){
int middle, start, final, j;
int *mas=new int[100];
middle=(first+last)/2; //вычисление среднего элемента
start=first; //начало левой части
final=middle+1; //начало правой части
for(j=first; j<=last; j++) //выполнять от начала до конца
if ((start<=middle) && ((final>last) || (A[start]<A[final]))){
mas[j]=A[start];
start++;}
else{
mas[j]=A[final];
final++;}

//возвращение результата в список
for (j=first; j<=last; j++) A[j]=mas[j];
delete[]mas;
}
```

void MergeSort(int *A, int first, int last)

```
void MergeSort(int *A, int first, int last){
{
if (first<last)
{
MergeSort(A, first, (first+last)/2); //сортировка левой части
MergeSort(A, (first+last)/2+1, last); //сортировка правой части
Merge(A, first, last); //слияние двух частей
}
}
```

Лабораторная работа №2

Реализовать класс Интернет-магазин (Rate). В классе предусмотрены следующие параметры: Товар (Position), Стоимость товара (Price, тип double/float), Дата прибытия (ArrivingDate, формат dd.mm.yyyy), Количество товара (Count, тип integer). На основании односвязного списка необходимо реализовать:

А) Добавление в начало списка;

Б) Добавление в конец списка;

В) Удаление любого элемента списка, номер удаляемого элемента вводится с клавиатуры. Если удаляемого номера нет, необходимо выбрасывать ошибку, что такого номера нет.

Г) Вывод введенного списка на экран.

Дополнительно, реализовать следующие функции:

Д) Расчет общей стоимости товара: Необходимо для названия товара, введенного с клавиатуры, найти общую стоимость поставки с учетом наценки по следующей формуле: Price*Count*18%. Если товара нет, то выбрасываем ошибку (try/catch).

Е) Отсортировать записи в порядке увеличения стоимости товара (сортировка слиянием).

Требования к лабе:

1. Поле «Position» (Brief) не должно быть короче 5 символов;
2. «Стоимость товара» (Price) и «Количество товара» (Count) не могут быть отрицательными числами или нулём.
3. Каждый пункт в лабе (А-Д) – отдельный пункт меню.

Срок сдачи – 10.10.2019.