Лабораторная работа №5 (2-й семестр)

> Программировать — значит понимать.
>
> Кристин Нюгард

Теоретические сведения

Двусвязный список представляет собой расширение идеи односвязного списка и состоит из элементов данных, каждый из которых содержит ссылки как на следующий, так и на предыдущий элементы. На рис. 1 показана организация ссылок в двусвязном списке.

Рис. 1 — Графическое представление двусвязного списка

Наличие двух ссылок вместо одной предоставляет несколько преимуществ. Наиболее важное из них состоит в том, что перемещение по списку возможно в обоих направлениях. Это упрощает работу со списком, в частности, вставку и удаление. Помимо этого, пользователь может просматривать список в любом направлении. Еще одно преимущество имеет значение только при некоторых сбоях. Поскольку весь список можно пройти не только по прямым, но и по обратным ссылкам, то в случае, если какая-то из ссылок станет неверной, целостность списка можно восстановить по другой ссылке.

Пример. Написать программу, в которой реализован класс, описывающий двусвязанный список для хранения целых чисел. Пользователь имеет возможность добавлять, редактировать и удалять элементы (рис. 2).

Файл: *Unit1.cpp*

```
#include <wx/textdlg.h>
#include "List.h"
List list;
likedlistFrame::likedlistFrame(wxWindow* parent,wxWindowID id)
{
…
list.SetWxGrid(Grid1);
}

void likedlistFrame::OnaddButtonClick(wxCommandEvent& event)
{
    list.AddHead(wxAtoi(TextCtrl1->GetValue()));
}
void likedlistFrame::OndelButtonClick(wxCommandEvent& event)
{
     list.Del(0);
}
void likedlistFrame::OnGrid1CellLeftDClick(wxGridEvent& event)
{   int i = event.GetRow();
    wxString value = Grid1->GetCellValue(i,0);
       wxString res = wxGetTextFromUser(_("Введите новое значение"),wxGetTextFromUserPromptStr,value);
    if (res.Length())
```

```
        list.SetValue(i,wxAtoi(res));
}
```

Файл: *List.h*

```
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED
#include <wx/grid.h>
struct Elem
{
     int data; // данные
     Elem * next, * prev;
};
class List
{
     Elem * Head, * Tail;// Голова, хвост
     int Count;          // Количество элементов
     wxGrid *wxg;   // StringGrid для отображения
public:
     List();   // Конструктор
     ~List();  // Деструктор
     int GetCount();     // Получить количество э-тов
     void DelAll();      // Удалить весь список
     void Del(int pos = 0);   // Удаление элемента
     void AddHead(int n);     // Добавление в начало списка
     void SetWxGrid(wxGrid* _wxg);
     void PrintToWxG(); // Распечатка содержимого списка
     void SetValue(int index, int data);
};
#endif // LIST_H_INCLUDED
```

Файл: *List.cpp*

```
#include "list.h"
List::List()
{
     // Изначально список пуст
     Head = Tail = NULL;
     Count = 0;
}
List::~List()
{
     wxg = NULL;
     DelAll();      // Удаляем все элементы
}
void List::AddHead(int n)
{
     Elem * temp = new Elem; // новый элемент
     temp->prev = NULL;  // Предыдущего нет
     temp->data = n;     // Заполняем данные
     temp->next = Head;  // Следующий - бывшая голова
     if (Head != NULL)   // Если элеиенты есть?
          Head->prev = temp;
     // Если элеиент первый, то он одновременно и голова и хвост
     if (Count == 0)
          Head = Tail = temp;
```

```
     else
          Head = temp;     // иначе новый элемент - голова
     Count++;
     PrintToWxG();
}
void List::Del(int pos)
{
    pos++;
     if(pos < 1 || pos > Count)
          return;
     int i = 1;      // Счетчик
     Elem * Del = Head;
     while(i < pos)
     {
          Del = Del->next; // Доходим до удаляемого э-та,
          i++;
     }
     // Доходим до элемента,
     // который предшествует удаляемому
     Elem * PrevDel = Del->prev;
     // Доходим до элемента, который следует за удаляемым
     Elem * AfterDel = Del->next;
     if (PrevDel != 0 && Count != 1) // Если удаляем не голову
          PrevDel->next = AfterDel;
      if(AfterDel != 0 && Count != 1) // Если удаляем не хвост
          AfterDel->prev = PrevDel;
     if(pos == 1)    // Удаляются крайние?
          Head = AfterDel;
     if(pos == Count)
          Tail = PrevDel;
     delete Del;     // Освобождение памяти
     Count--;
     PrintToWxG();
}
void List::DelAll()
{
     // Пока остаются элементы, удаляем по одному с головы
     while(Count != 0)
          Del(1);
}
int List::GetCount()
{
     return Count;
}
void List::SetValue(int index, int data)
{
     if (index > GetCount())
          return;
     Elem * temp = Head;
     int i = 0;
     while (temp != 0 && i < index)
     {
          i++;
```

```
            temp = temp->next;
        }
        temp->data = data;
        PrintToWxG();
}
void List::SetWxGrid(wxGrid* _wxg)
{
        wxg = _wxg;
}
void List::PrintToWxG()
{
        if (wxg == NULL)
                return;
        Elem * temp = Head;       // запоминаем адрес головного эл-та
        wxg->ClearGrid();
        if (wxg->GetRows())
                wxg->DeleteRows(0,wxg->GetRows());
        int i = 0;
        while(temp != NULL)       // Пока еще есть элементы
        {
                wxg->InsertRows(i);
                wxg->SetCellValue(i,0,wxString()<<temp->data);
                temp = temp->next;  // Переходим на следующий элемент
                i++;
        }
}
```

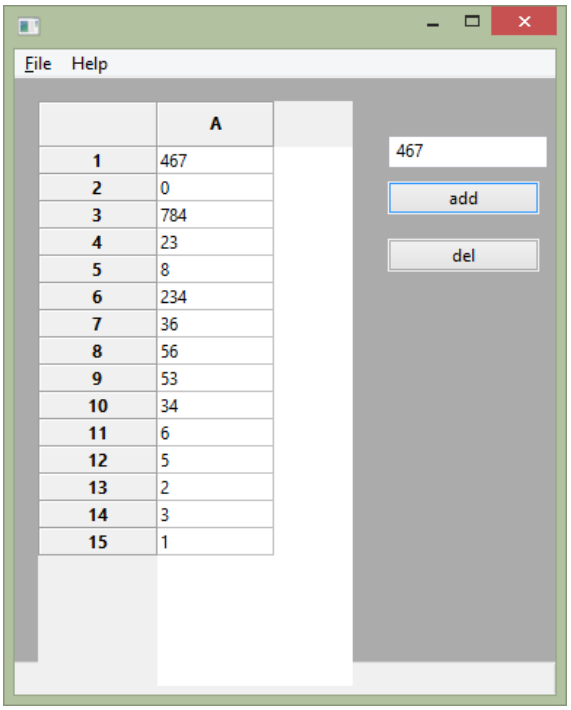

Рис. 2 — Пример формы приложения