Лабораторная работа №4 (3-й семестр)

> Лучший способ в чём-то разобраться до конца — это попробовать научить этому компьютер.
>
> — Дональд Кнут

Теоретические сведения

Свя́зный спи́сок — структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

Односвязный список состоит из указателя на первый элемент списка (голову, head) и самих данных, причем каждый элемент списка содержит указатель на следующий. Последний элемент списка содержит указатель на NULL (рис. 1).

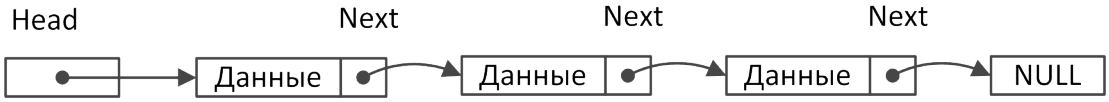

Рис. 1 — Графическое представление связанного списка

Вставка узла. Одной из типичных операций при работе со списком является вставка нового узла в определенное место связанного списка. Для этого необходимо выполнить следующие действия:

1. Выделить память под новый узел и заполнить в нем поля данных;
2. В добавляемом элементе установить указатель на следующий узел, а в предыдущем — на добавляемый (рис. 2).

Необходимо заметить, что при добавление узла в начало или конец списка алгоритм несколько изменятся, поэтому некоторые действия могут отсутствовать.

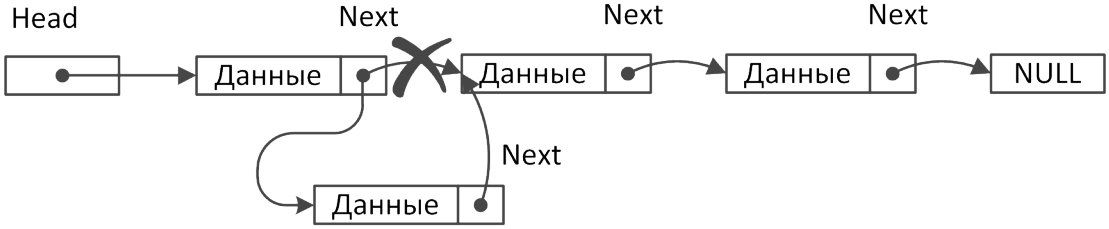

Рис. 2 — Графическое представление вставки нового элемента в список

Удаление узла. Второй типичной операцией со списками является удаление узла, находящегося в середине списка. Для этого необходимо выполнить следующие действия:

1. Записать адрес узла, следующего за удаляемым узлом, в указатель на следующий узел в узле, предшествующем удаляемому (рис. 3).
2. Освободить память занимаемую узлом, предназначенным для удаления.

При удалении узла из начала или конца списка вышеописанная последовательность действий может изменяться.

Рис. 3 — Графическое представление удаления элемента из списка

Пример. Написать программу, в которой реализован класс, описывающий связанный список для хранения целых чисел. Пользователь имеет возможность добавлять, редактировать и удалять элементы (рис. 4).

Файл: *Unit1.cpp*

```
#include "Clist.h"
#include <wx/textdlg.h>
...
CList list;
list3Frame::list3Frame(wxWindow* parent,wxWindowID id)
{
...
list.SetWxGrid(Grid1);
}
void list3Frame::OnaddButtonClick(wxCommandEvent& event)
{
    list.Add(wxAtoi(TextCtrl1->GetValue()));
}
void list3Frame::OndelButtonClick(wxCommandEvent& event)
{
    list.Del();
}
void list3Frame::OnGrid1CellLeftDClick(wxGridEvent& event)
{
    int i = event.GetRow();
    wxString value = Grid1->GetCellValue(i,0);
    wxString res = wxGetTextFromUser(_("Введите новое значе-
ние"),wxGetTextFromUserPromptStr,value);
    if (res.Length())
        list.SetValue(i,wxAtoi(res));
}
```

Файл: *CList.h*

```
#ifndef CLIST_H
#define CLIST_H
#include <wx/grid.h>
struct Element   // Элемент данных
{
     int data;       // Данные
     Element * Next; // Адрес следующего элемента списка
};
class CList
{
     Element * Head;     // Указатель на голову списка
     int Count;          // Количество элементов списка
     wxGrid *wxg;   // StringGrid для отображения
public:
     CList();  // Конструктор
     ~CList(); // Деструктор
     void Add(int data);// Добавление элемента в список
     void Del();// Удаление элемента из списка
     void DelAll(); // Удаление всего списка
     void SetWxGrid(wxGrid* _wxg);
     void PrintToWxG();// Распечатка содержимого списка
     void SetValue(int index, int data); // Задание нового зач-ния
```

```
    int GetCount(); // Получение количества элементов в списке
};
#endif // CLIST_H
```

Файл: *CList.cpp*

```
#include "CList.h"
CList::CList()
{
    Head = NULL;   // Изначально список пуст
    Count = 0;
}
CList::~CList()
{
    wxg = NULL;
    DelAll();      // Вызов функции удаления
}
int CList::GetCount()
{
    return Count; // Возвращаем количество элементов
}
void CList::Add(int data)
{
    Element * temp = new Element; // создание нового элемента
    temp->data = data; // заполнение данными
    temp->Next = Head; // следующий элемент - головной
    Head = temp; // новый элемент становится головным элементом
    Count++; // Увиличиваем кол-во э-тов
    PrintToWxG();
}
void CList::Del()
{
    if (Head == NULL)
        return;
    Element * temp = Head;   // запоминаем адрес головного эл-та
    Head = Head->Next;  // перебрасываем голову на следующий эл-т
    delete temp;        // удаляем бывший головной элемент
    Count--;            // уменьшаем кол-во э-тов на 1
    PrintToWxG();       // печатаем список
}
void CList::DelAll()
{
    while(Head != NULL) // Пока еще есть элементы
        Del();          // Удаляем элементы по одному
}
void CList::SetWxGrid(wxGrid* _wxg)
{
    wxg = _wxg;
}
void CList::PrintToWxG()
{
    if (wxg == NULL)
        return;
    Element * 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++;
     }
}
void CList::SetValue(int index, int data)
{
     if (index > GetCount())
          return;
     Element * temp = Head;
     int i = 0;
     while (temp != 0 && i < index)
     {
          i++;
          temp = temp->Next;
     }
     temp->data = data;
     PrintToWxG();
}
```

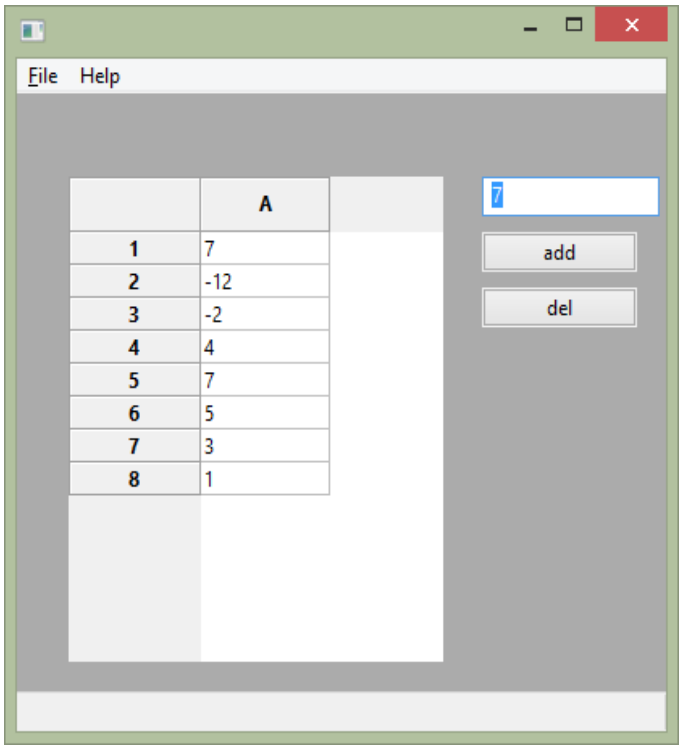

Рис. 4 — Пример формы приложения