Лекция №2

Кросс-платформенное программирование

Структуры данных

Структуры данных

Структура данных (англ. data structure) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

Структуры данных

Структуры данных применяются преимущественно для решения трех задач:

- Хранение реальных данных.
- Инструментарий программиста.
- Моделирование.

Типы данных

Атомарные

bool, char, float/double, int, enum

Составные

массив, структура, объединение

Абстрактные

список, стек, очередь, дерево, граф

Полный список типов данных
http://en.wikipedia.org/wiki/List_of_data_structures
http://ru.wikipedia.org/wiki/Список_структур_данных

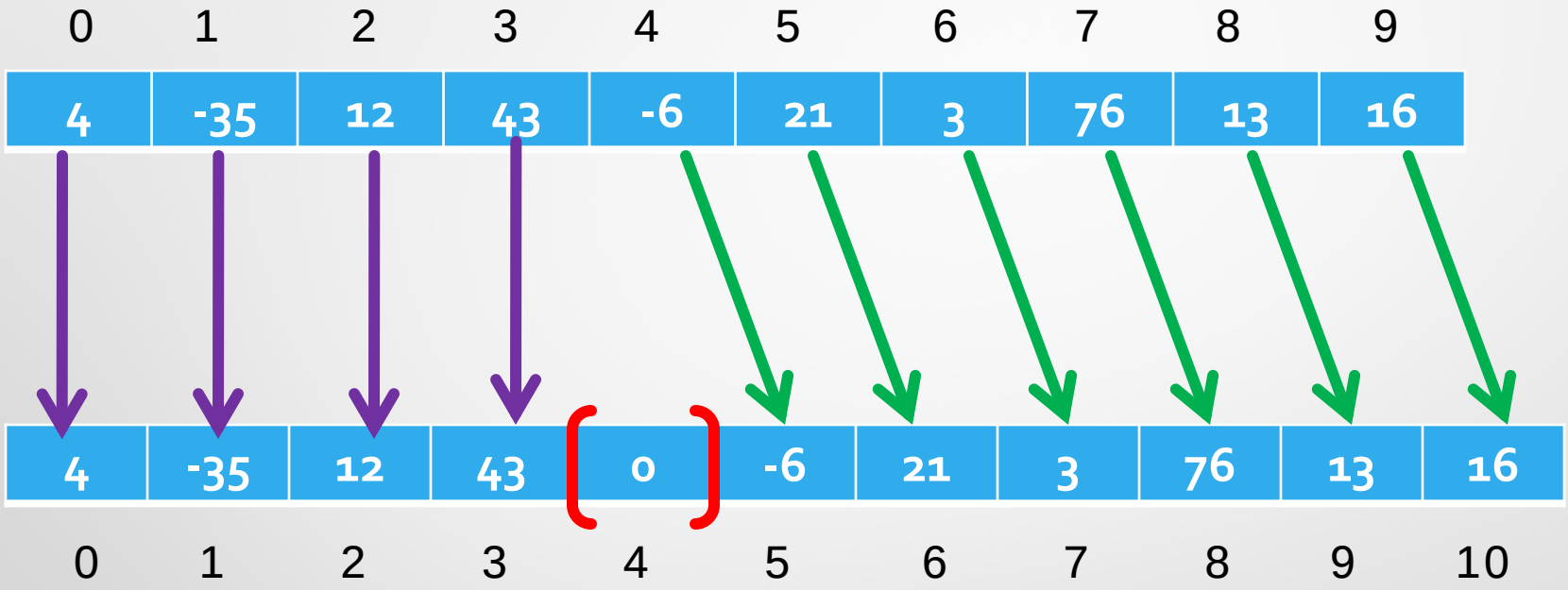
Массив

- **Массив** — структура данных, которая хранит в памяти однотипные элементы непосредственно друг за другом, доступ к которым осуществляется по индексу (индексам). Массив является структурой с произвольным доступом.

Вставка элемента в массив

Дан массив A[10]

Исходный массив
Вставить после 4 элемента 0



Какова сложность такой операции?

Временная сложность операций с массивом

Отлично	Хорошо	Удовлетворительно	Плохо	Ужасно

Структура данных	Временная сложность							
	Средняя				Худшая			
	Доступ	Поиск	Вставка	Удален.	Доступ	Поиск	Вставка	Удален.
Массив (Array)	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)

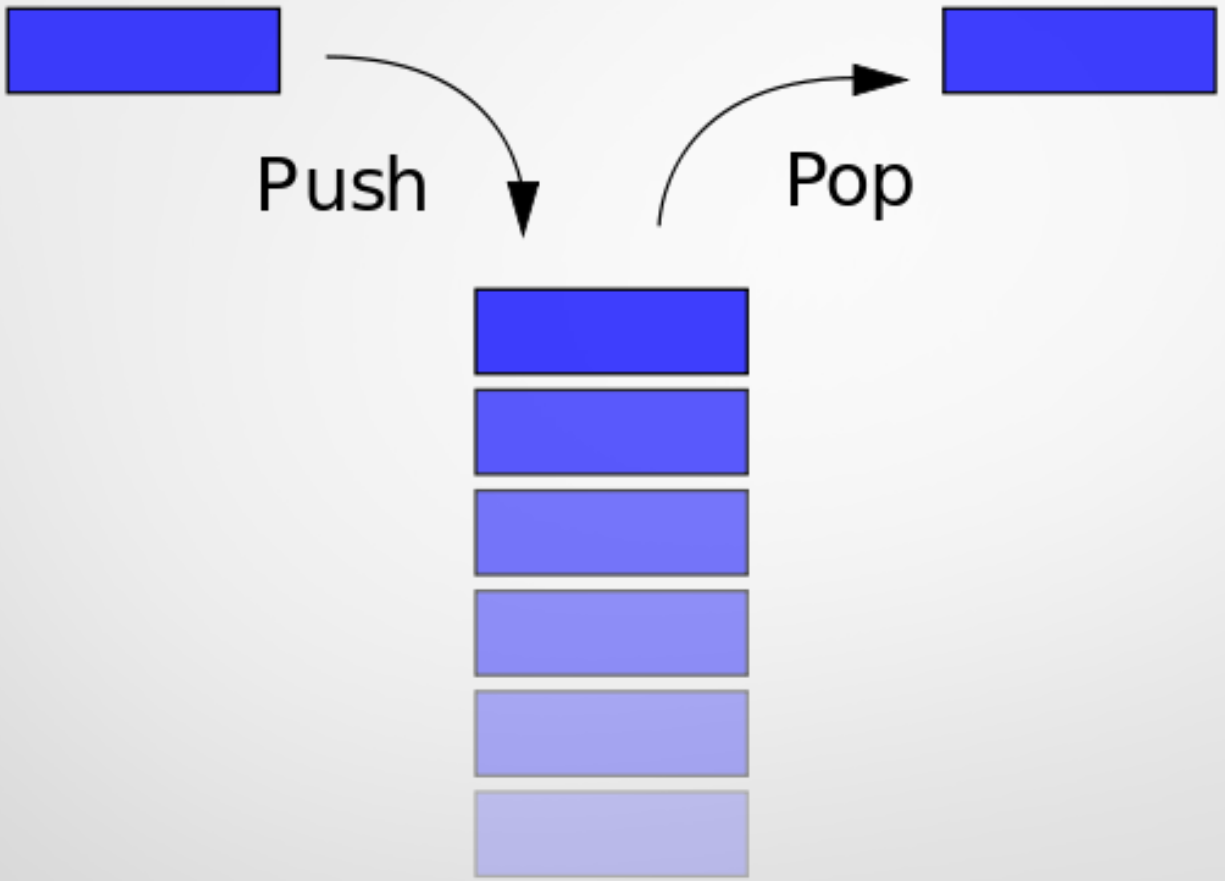
Стек

Стек (англ. stack — стопка) — структура данных, в которой доступ к элементам организован по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

Стек

Добавление элемента, называемое также проталкиванием (push), возможно только в вершину стека (добавленный элемент становится первым сверху). Удаление элемента, называемое также выталкиванием (pop), тоже возможно только из вершины стека, при этом второй сверху элемент становится верхним

Стек



Принцип работы стека

Стек

Стеки широко применяются в вычислительной технике. Например, для отслеживания точек возврата из подпрограмм используется стек вызовов, который является неотъемлемой частью архитектуры большинства современных процессоров. Языки программирования высокого уровня также используют стек вызовов для передачи параметров при вызове процедур.

Стек

```
const int stacksize = 10;
const int EMPTY = -1;
class Stack
{
    int arr[stacksize]; // Массив для хранения данных
    int top; // Вершина стека
public:
    Stack(); // Конструктор
    void push(int c); // Добавление элемента
    int pop(); // Выталкивание элемента
    void clear(); // Очистка стека
    bool isEmpty(); // Проверка на наличие элементов в стеке
    bool isFull(); // Проверка на заполнение всего стека
    int getCount(); // Количество элементов в стеке
};
```

Временная сложность операций со стеком

Отлично	Хорошо	Удовлетворительно	Плохо	Ужасно

Структура данных	Временная сложность							
	Средняя				Худшая			
	Доступ	Поиск	Вставка	Удален.	Доступ	Поиск	Вставка	Удален.
Массив (Array)	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
Стек (Stack)	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)

Поиск парных скобок

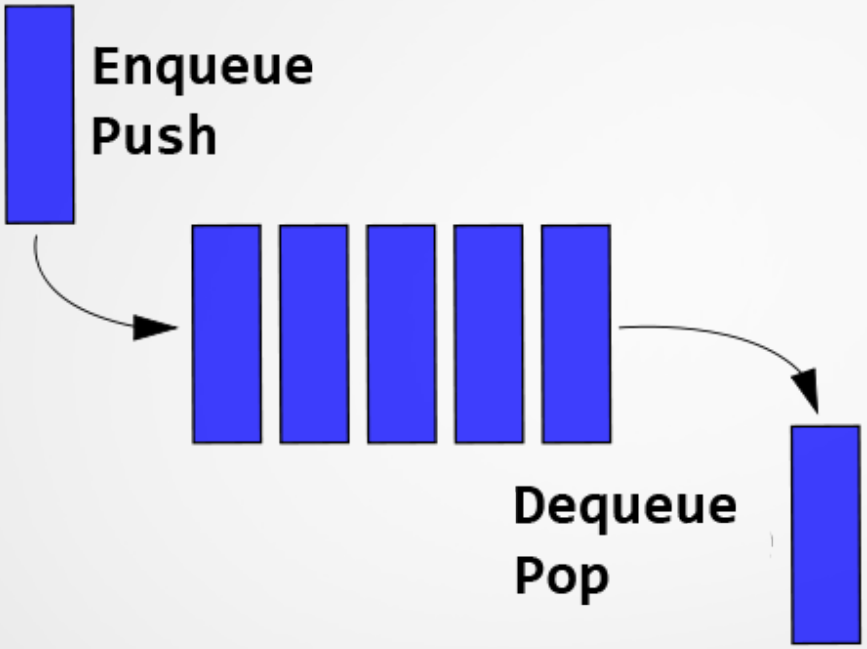
- Стеки также часто используются при разборе некоторых видов текстовых строк.

```
c[d] // Правильно
a{b[c]d}e // Правильно
a{b(c]d}e // Неправильно; ] не соответствует (
a[b{c}d]e}// Неправильно; у завершающей скобки } нет пары
a{b(c) // Неправильно; у открывающей скобки { нет пары
```

Структуры данных

О́чередь — структура данных с типом доступа к элементам «первый пришёл — первый вышел» (FIFO, First In — First Out). Добавление элемента возможно лишь в конец очереди, а извлечение — только из начало очереди, при этом выбранный элемент из очереди удаляется. В различных библиотеках методы добавления и извлечения элементов в очередь могут называться по-разному. Часто для добавления используют название методов push или enqueue, а для извлечения — pop или dequeue (рис. 1).

Структуры данных



Общий принцип работы очереди

```
class Queue
{
    int *q; // Очередь
    int qEnd; // Текущий размер очереди
    int maxQLength; // Максимальный размер очереди
public:
    Queue(int MaxLength);// Конструктор
    ~Queue(); // Деструктор
    bool push(int elem); // Добавление элемента в очередь
    bool pop(int & elem);// Извлечение элемента из очереди
    bool isEmpty(); // Очередь пуста?
    bool isFull(); // Очередь заполнена?
    void setSG(TStringGrid* _sg);
    void printToSG(); // Вывод очереди
};
```

Временная сложность операций с очередью

Отлично	Хорошо	Удовлетворительно	Плохо	Ужасно

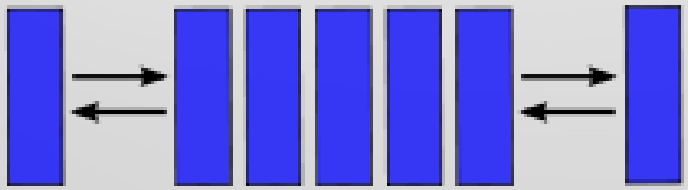
Структура данных	Временная сложность							
	Средняя				Худшая			
	Доступ	Поиск	Вставка	Удален.	Доступ	Поиск	Вставка	Удален.
Массив (Array)	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
Стек (Stack)	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Очередь (Queue)	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)

Структуры данных

Дек (deque) представляет собой двустороннюю очередь. И вставка, и удаление элементов могут производиться с обоих концов. Соответствующие методы могут называться insertLeft() / insertRight() и removeLeft() / removeRight() .

Если ограничиться только методами insertLeft() и removeLeft() (или их эквивалентами для правого конца), дек работает как стек. Если же ограничиться методами insertLeft() и removeRight() (или противоположной парой), он работает как очередь.

По своей гибкости деки превосходят и стеки, и очереди; иногда они используются в библиотеках классов-контейнеров для реализации обеих разновидностей.



Приоритетная очередь

- Приоритетная очередь (очередь с приоритетами) является более специализированной структурой данных, чем стек или очередь, однако и он неожиданно часто оказывается полезным.
- У приоритетной очереди, как и у обычной, имеется начало и конец, а элементы извлекаются от начала. Но у приоритетной очереди элементы упорядочиваются по специальному полю - ключу, так что элемент с наименьшим (в некоторых реализациях — наибольшим) значением ключа всегда находится в начале. Новые элементы вставляются в позициях, сохраняющих порядок сортировки.

Связанные списки

Массивы являются удобной формой хранения данных в том случае, когда объем данных в процессе обработки заранее известен и не подвержен изменениям, а также требуется высокая эффективность (производительность) доступа к данным.

Иногда структура данных в связи с моделируемой предметной областью такова, что заранее нельзя предсказать их объем, а также часто возникает необходимость в операциях удаления и добавления элементов.

Связанные списки

- **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

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.

```
struct Element // Элемент данных
{   int data; // Данные
    Element * Next; // Адрес следующего элемента списка
};
```

Класс для работы со списком

```
class CList
{
    Element * Head; // Указатель на голову списка
    int Count; // Количество элементов списка
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 PrintToWxG();// Распечатка содержимого списка
    void SetValue(int index, int data); // Задание нового зач-ния
    int GetCount(); // Получение количества элементов в списке
};
```

Односвязный список

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:

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

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

Добавление элемента

```
void List::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
}
```

Односвязный список

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

- Записать адрес узла, следующего за удаляемым узлом, в указатель на следующий узел в узле, предшествующем удаляемому.
- Освободить память занимаемую узлом, предназначенным для удаления.

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

Вывод списка на экран

```
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++;
    }
}
```

Временная сложность операций со связанным списком

Отлично	Хорошо	Удовлетворительно	Плохо	Ужасно

Структура данных	Временная сложность							
	Средняя				Худшая			
	Доступ	Поиск	Вставка	Удален.	Доступ	Поиск	Вставка	Удален.
Массив (Array)	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
Стек (Stack)	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Очередь (Queue)	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Связанный список (Linked list)	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)