Тема 8

Структуры данных

Структуры данных – что это?

Структура данных – способ организации хранения данных и доступа к ним, предназначенный для выполнения определённого набора операций над этими данными.

Для каждой структуры вводится понятие удобных и неудобных операций.

Удобные операции – те, которые выполняются при использовании быстрее, чем при использовании других структур.

Неудобные операции – те, которые либо не могут быть выполнены, либо те, которые выполняются медленнее по сравнению с другими структурами.

Простейшие структуры данных

- массив
- линейный однонаправленный список
- стек
- очередь

Массив

Удобные операции:

- доступ к элементу массива по индексу;
- просмотр элементов по возрастанию индексов;
- поиск элемента по значению в упорядоченном массиве

Неудобные операции:

- вставка элементов в произвольное место массива и удаление из произвольного места;
- поиск элемента по значению в неупорядоченном массиве

Дихотомия

Дихотомия («деление пополам») – алгоритм поиска элемента по значению в упорядоченном по возрастанию массиве

Описание алгоритма:

- если массив пуст, элемент не найден;
- сравниваем искомое значение со средним элементом массива;
- если они равны, элемент найден;
- если средний элемент меньше искомого значения, продолжаем поиск в нижнем подмассиве, иначе – в верхнем.

Реализация дихотомии

```
bool Find(int what, int* mass, int first, int last) {
  if (first>last)
   return false;
  int medium = (first+last)/2;
  if (mass[medium]==what)
   return true;
  if (mass[medium]<what)
      return Find(what, mass, medium+1, last);
  else
   return Find(what, mass, first, medium-1);
}
…
int mas[] = {2, 4, 7, 18, 40, 45, 48, 52, 76, 101};
cout << (Find(101, mas, 0, 9) ? "Yes" : "No") << endl;
```

Списки

Список – совокупность элементов, каждый из которых, кроме последнего, содержит информацию (ссылку) о следующем элементе. Отдельно должна храниться информация о первом элементе списка.

Удобные операции над списками:

- вставка в заранее определённое место списка;
- удаление из заранее определённого места списка.

Неудобные операции над списками:

- поиск элемента по значению;
- доступ к элементу по его номеру.

Структура линейного однонаправленного списка

Элемент списка

Ссылка на первый элемент списка

Пустой список

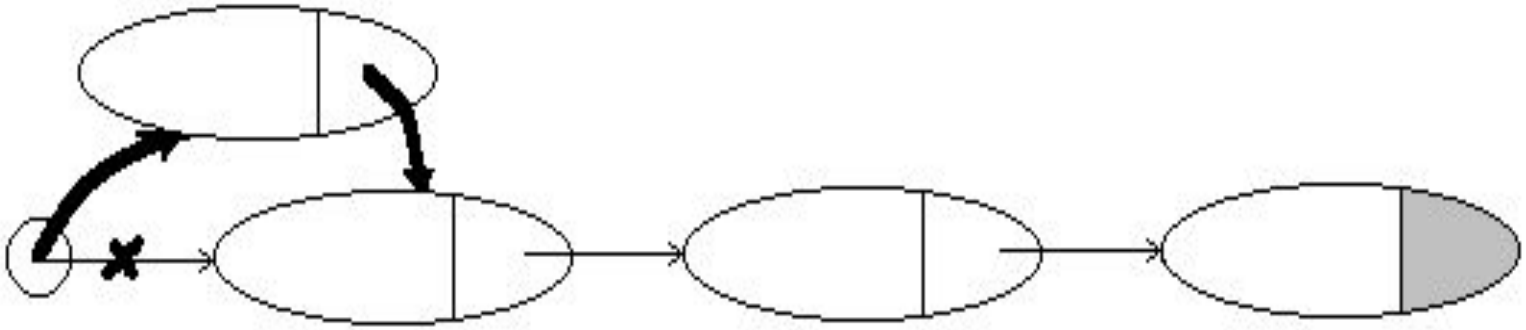
Список, состоящий из трех элементов

Реализация списка с использованием динамической памяти

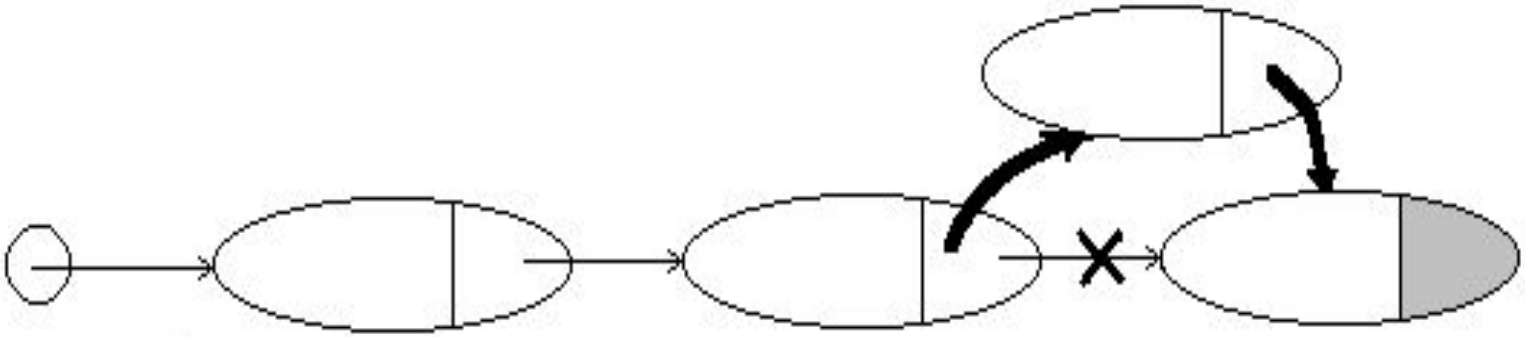
```
struct ListItem {
  int Info;
  ListItem *Next;
};

ListItem *First;
```

Схема добавления в список нового элемента



Вставка нового элемента в начало списка



Вставка нового элемента в середину списка

Реализация вставки в список нового элемента

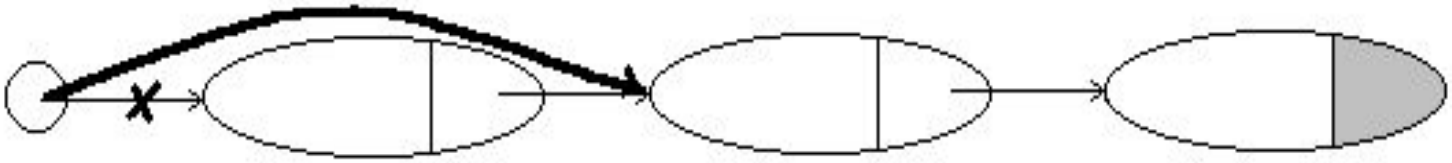
В начало списка

```
ListItem *P = new ListItem;
// заполнение поля P->Info
P->Next = First;
First = P;
```

После элемента, адрес которого находится в указателе Q

```
ListItem *P = new ListItem;
// заполнение поля P->Info
P->Next = Q->Next;
Q->Next = P;
```

Схема удаления элемента из списка



Удаление первого элемента списка

Удаление элемента из середины списка

Удаление последнего элемента списка

Реализация удаления элемента из списка

Из начала списка

```
if (First == NULL)
 // обработать ситуацию ”List is already empty!”
ListItem *P = First;
First = First->Next;
delete P;
```

После элемента, адрес которого находится в указателе Q

```
ListItem *P = Q->Next;
if (P == NULL)
   // обработать ситуацию ”Nothing to delete!”
Q->Next = P->Next;
delete P;
```

Просмотр всех элементов списка

```
ListItem *P = First;
while (P != NULL) {
  // выполнить действия над элементом P->Info
  P = P->Next;
}
```

Организация списка с использованием массива

first	5

0		
1		
2	Петров	7
3		
4		
5	Алексеев	8
6		
7	Сидоров	-1
8	Борисов	2

Проблема "сборки мусора"

Суть проблемы: как организовать повторное использование памяти, освободившейся после удаления элемента списка

- При удалении элемента из списка , организованного с использованием динамической памяти, память сразу же становится доступной для повторного использования
- При удалении элемента из списка, организованного в виде массива, эту проблему приходится решать самостоятельно!

Сборка мусора при организации списка в виде массива

0		6
1		3
2	Петров	7
3		0
4		-1
5	Алексеев	8
6		4
7	Сидоров	-1
8	Борисов	2

Индекс начального элемента: 5

Индекс начального свободного элемента: 1

Инициализация списка в виде массива

0		1
1		2
2		3
3		4
4		5
5		6
6		7
7		8
8		-1

Индекс начального элемента: -1

Индекс начального свободного элемента: 0

Работа со списком в виде массива

0	Кузнецов	-1
1		2
2		3
3		4
4		5
5		6
6		7
7		8
8		-1

Добавляем «Кузнецов»

Индекс начального элемента: 0

Индекс начального свободного элемента: 1

Работа со списком в виде массива (часть 2)

0	Кузнецов	-1
1	Иванов	0
2		3
3		4
4		5
5		6
6		7
7		8
8		-1

Добавляем «Иванов»

Индекс начального элемента: 1

Индекс начального свободного элемента: 2

Работа со списком в виде массива (часть 3)

0	Кузнецов	-1
1	Иванов	2
2	Ковалёв	0
3		4
4		5
5		6
6		7
7		8
8		-1

Добавляем «Ковалёв»

Индекс начального элемента: 1

Индекс начального свободного элемента: 3

Работа со списком в виде массива (часть 4)

0	Кузнецов	-1
1	Иванов	3
2	Ковалёв	0
3		4
4		5
5		6
6		7
7		8
8		-1

Удаляем «Иванов»

Индекс начального элемента: 2

Индекс начального свободного элемента: 1

Стеки

Стек – структура данных, предназначенная для выполнения следующих операций:

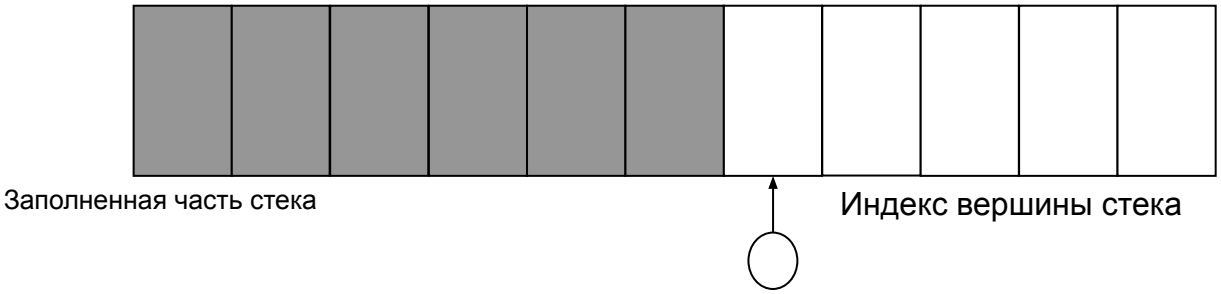
основные

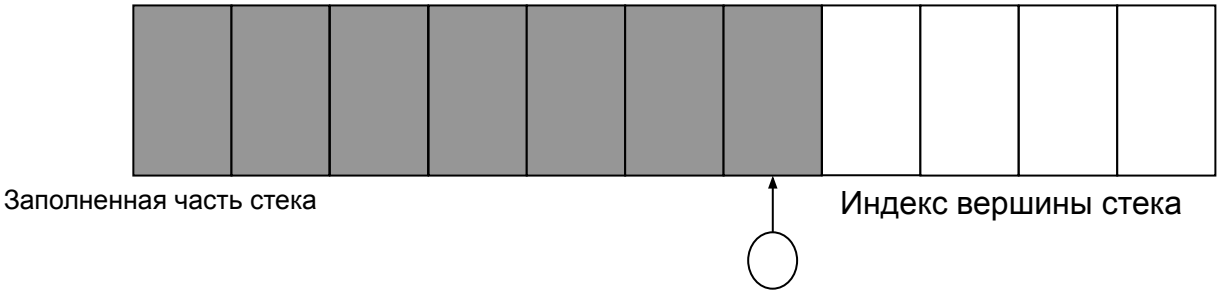
- занесение нового элемента на вершину стека (push);
- удаление элемента с вершины стека (pop)

дополнительные

- просмотр и, возможно, изменение элементов, находящихся в стеке, без изменения его структуры

Реализация стека с использованием массивов





Реализация стека на массивах

```
int * Stack, Top = 0;
…
Stack = new int [N];
// push (заносится значение k)
if (Top>=N)
  // обработать ситуацию ”Stack is full!”;
Stack[Top++] = k;
// pop (значение извлекается в k)
if (Top==0)
  throw ”Stack is empty!”;
k = Stack[--Top];
```

Очереди

Очередь – структура данных, предназначенная для выполнения следующих операций:

основные

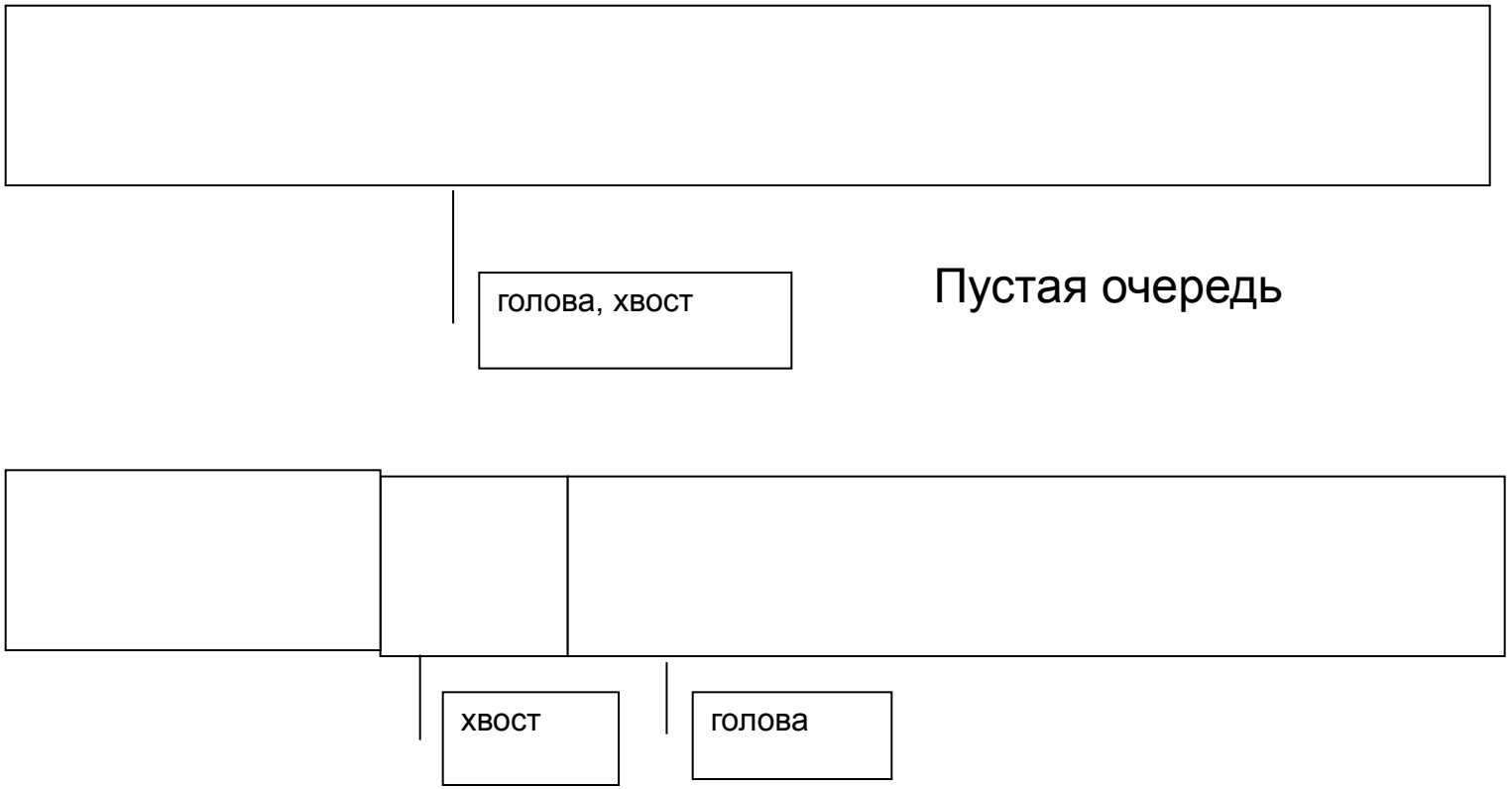
- занесение нового элемента в конец очереди (push);
- удаление элемента из начала очереди (pop)

дополнительные

- просмотр и, возможно, изменение элементов, находящихся в очереди, без изменения ее структуры

Реализация очереди на массивах

Организация циклической очереди



Программная реализация циклической очереди

```
int *Queue, Front = 0, Rear = 0;
…
Queue = new int [N+1];


// push (заносится значение k)
if ((Rear+1==Front) || ((Rear==N)&&(Front==0)))
  throw ”Queue is full!”;
Queue[Rear++] = k;
if (Rear>N) Rear=0;


// pop (значение извлекается в k)
if (Rear==Front)
  throw ”Queue is empty!”;
k = Queue[Front++];
if (Front>N) Front=0;
```

Реализация очереди с использованием списков