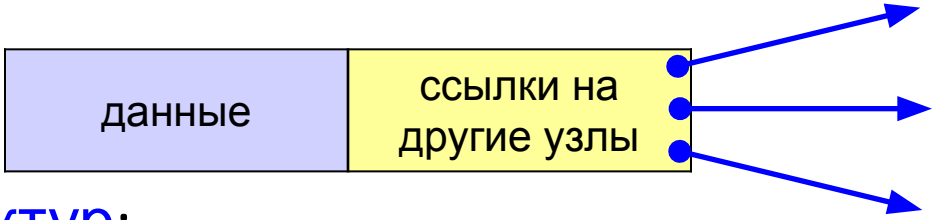
Динамические структуры данных (язык Си)

Тема 4. Списки

Динамические структуры данных

Строение: набор узлов, объединенных с помощью ссылок.

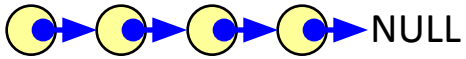
Как устроен узел:



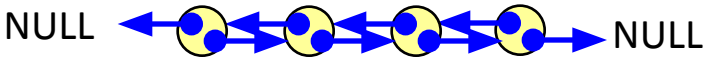
Типы структур:

списки

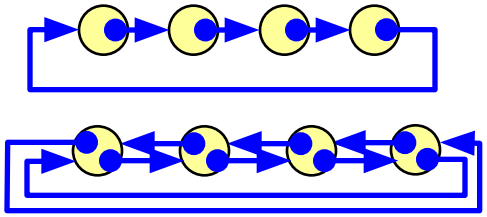
односвязный



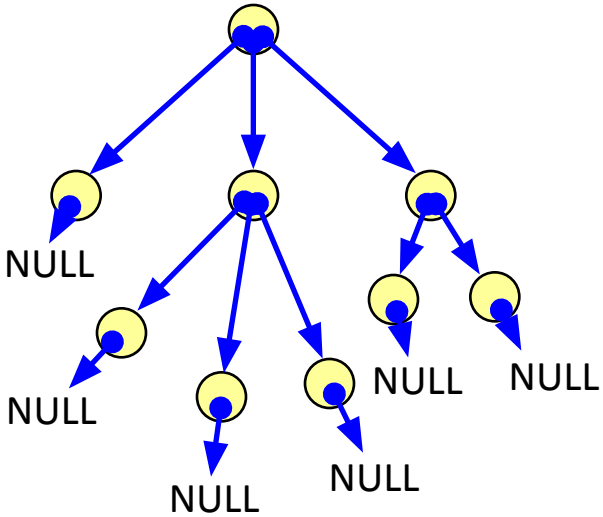
двунаправленный (двусвязный)



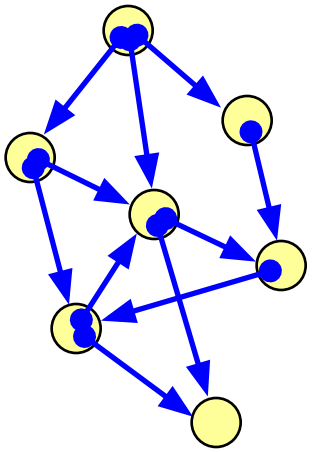
циклические списки (кольца)



деревья



графы



Когда нужны списки?

Задача (алфавитно-частотный словарь). В файле записан текст. Нужно записать в другой файл в столбик все слова, встречающиеся в тексте, в алфавитном порядке, и количество повторений для каждого слова.

Проблемы:

1) количество слов заранее неизвестно (ста~~тический массив~~);
2) количество слов определяется только в конце работы (~~динамический массив~~).

Решение – список.

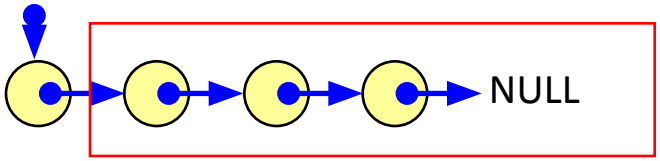
Алгоритм:

3) создать список;
4) если слова в файле закончились, то стоп.
5) прочитать слово и искать его в списке;
6) если слово найдено – увеличить счетчик повторений, иначе добавить слово в список;
7) перейти к шагу 2.

Списки: новые типы данных

Что такое список:

1) пустая структура – это список;
2) список – это начальный узел (*голова*) и связанный с ним список.



Рекурсивное определение!

Структура узла:

```
struct Node {
  char word[40];   // слово
  int  count;      // счетчик повторений
  Node *next;      // ссылка на следующий элемент
  };
```

Указатель на эту структуру:

```
typedef Node *PNode;
```

Адрес начала списка:

```
PNode Head = NULL;
```

Для доступа к списку достаточно знать адрес его головы!

Что нужно уметь делать со списком?

1. Создать новый узел.
2. Добавить узел:
   a) в начало списка;
   b) в конец списка;
   c) после заданного узла;
   d) до заданного узла.
3. Искать нужный узел в списке.
4. Удалить узел.

Создание узла

Функция `CreateNode` (*создать узел*):

вход: новое слово, прочитанное из файла;

выход: адрес нового узла, созданного в памяти.

возвращает адрес созданного узла

новое слово

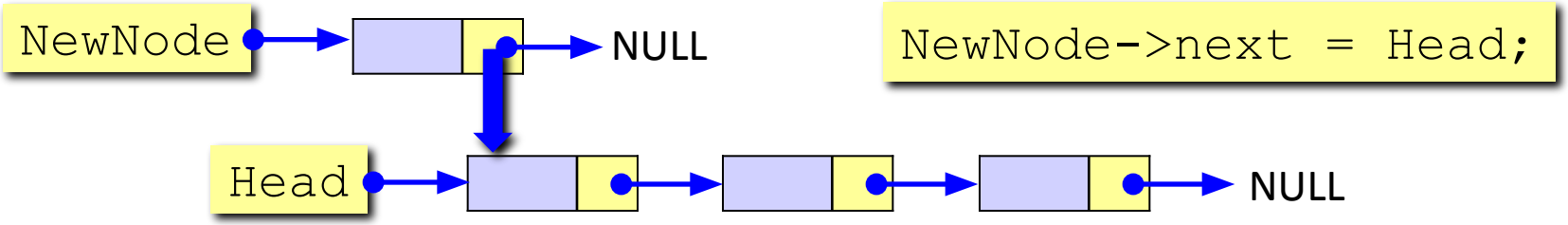
```
PNode CreateNode ( char NewWord[] )
{
  PNode NewNode = new Node;
  strcpy(NewNode->word, NewWord);
  NewNode->count = 1;
  NewNode->next = NULL;
  return NewNode;
}
```

?

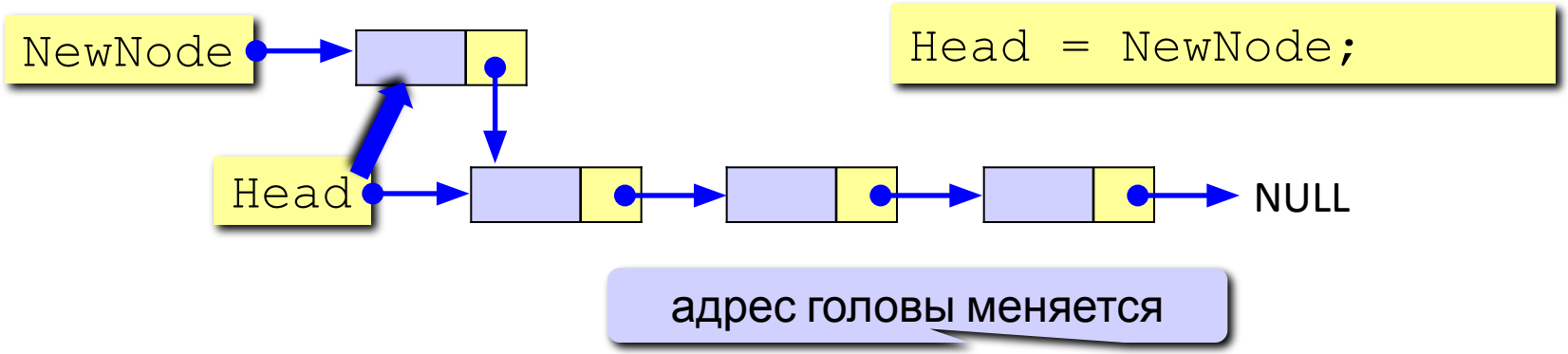
Если память выделить не удалось?

Добавление узла в начало списка

1) Установить ссылку нового узла на голову списка:



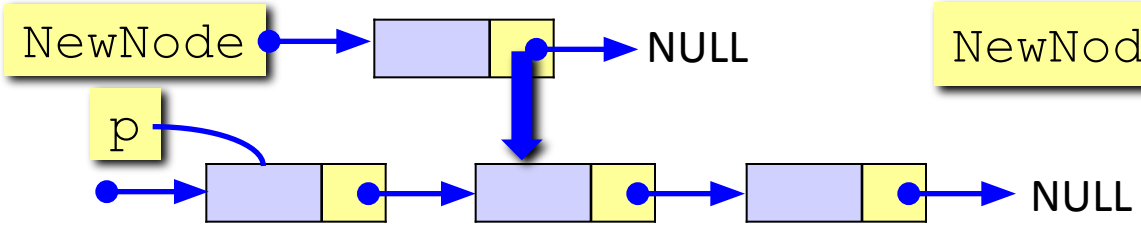
2) Установить новый узел как голову списка:




```
void AddFirst (PNode & Head, PNode NewNode)
{
  NewNode->next = Head;
  Head = NewNode;
}
```

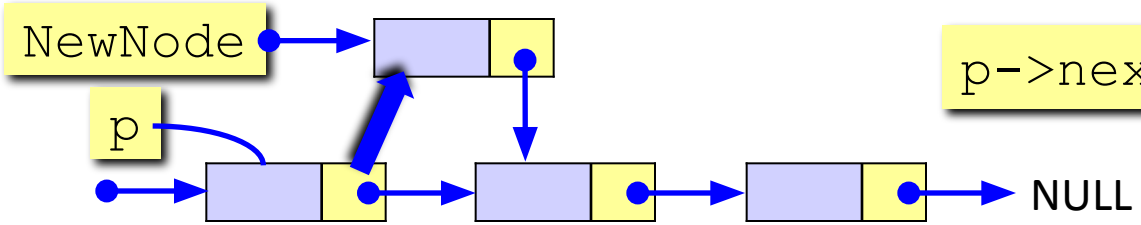
Добавление узла после заданного

1) Установить ссылку нового узла на узел, следующий за p:




```
NewNode->next = p->next;
```

2) Установить ссылку узла p на новый узел:



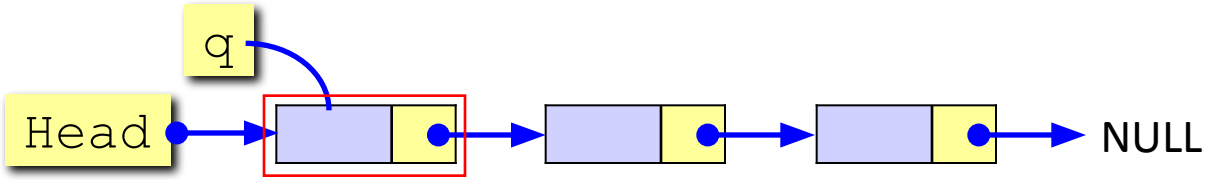

```
p->next = NewNode;
```

```
void AddAfter (PNode p, PNode NewNode)
{
   NewNode->next = p->next;
   p->next = NewNode;
}
```

Проход по списку

Задача:

сделать что-нибудь хорошее с каждым элементом списка.



Алгоритм:

1) установить вспомогательный указатель `q` на голову списка;
2) если указатель `q` равен `NULL` (дошли до конца списка), то стоп;
3) выполнить действие над узлом с адресом `q` ;
4) перейти к следующему узлу, `q->next`.

```
...
PNode q = Head;          // начали с головы
while ( q != NULL ) {    // пока не дошли до конца
  ...                    // делаем что-то хорошее с q
  q = q->next;           // переходим к следующему узлу
  }
...
```

Добавление узла в конец списка

Задача: добавить новый узел в конец списка.

Алгоритм:

1) найти последний узел `q`, такой что `q->next` равен `NULL`;
2) добавить узел после узла с адресом `q` (процедура `AddAfter`).

Особый случай: добавление в пустой список.

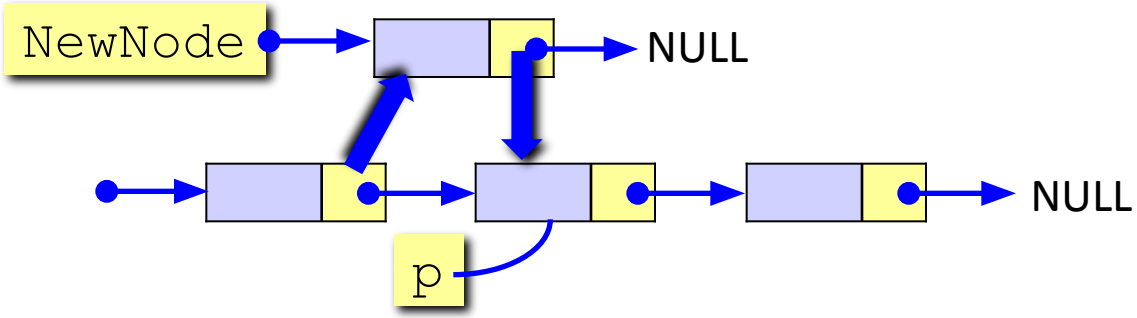
```
void AddLast ( PNode &Head, PNode NewNode )
{
PNode q = Head;
if ( Head == NULL ) {
   AddFirst ( Head, NewNode );
   return;
   }
while ( q->next ) q = q->next;
AddAfter ( q, NewNode );
}
```

особый случай – добавление в пустой список

ищем последний узел

добавить узел после узла `q`

Добавление узла перед заданным



Проблема:

нужно знать адрес предыдущего узла, а идти назад нельзя!

Решение: найти предыдущий узел q (проход с начала списка).

```
void AddBefore ( PNode & Head, PNode p, PNode NewNode )
{
PNode q = Head;
if ( Head == p ) {
   AddFirst ( Head, NewNode );
   return;
   }
while ( q && q->next != p ) q = q->next;
if ( q ) AddAfter(q, NewNode);
}
```

особый случай – добавление в начало списка

ищем узел, следующий за которым – узел p

добавить узел после узла q

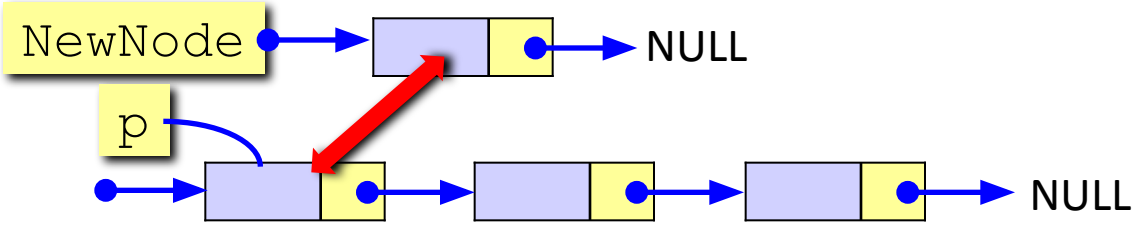
? Что плохо?

Добавление узла перед заданным (II)

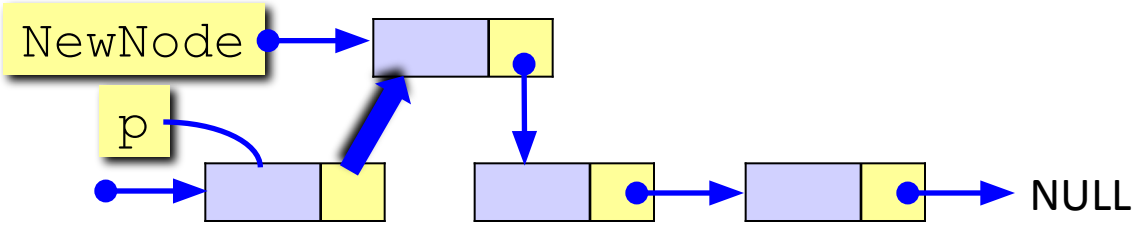
Задача: вставить узел перед заданным без поиска предыдущего.

Алгоритм:

1) поменять местами данные нового узла и узла `p`;



2) установить ссылку узла `p` на `NewNode`.




```
void AddBefore2 ( PNode p, PNode NewNode )
{
  Node temp;
  temp = *p; *p = *NewNode;
  *NewNode = temp;
  p->next = NewNode;
}
```

! Так нельзя, если `p = NULL` или адреса узлов где-то еще запоминаются!

Поиск слова в списке

Задача:

найти в списке заданное слово или определить, что его нет.

Функция `Find`:

вход: слово (символьная строка);

выход: адрес узла, содержащего это слово или `NULL`.

Алгоритм: проход по списку.

результат – адрес узла

ищем это слово

```
PNode Find ( PNode Head, char NewWord[] )
{
  PNode q = Head;
  while ( q && strcmp ( q->word, NewWord) )
    q = q->next;
  return q;
}
```

пока не дошли до конца списка и слово не равно заданному

Куда вставить новое слово?

Задача:

найти узел, перед которым нужно вставить, заданное слово, так чтобы в списке сохранился алфавитный порядок слов.

Функция `FindPlace`:

вход: слово (символьная строка);

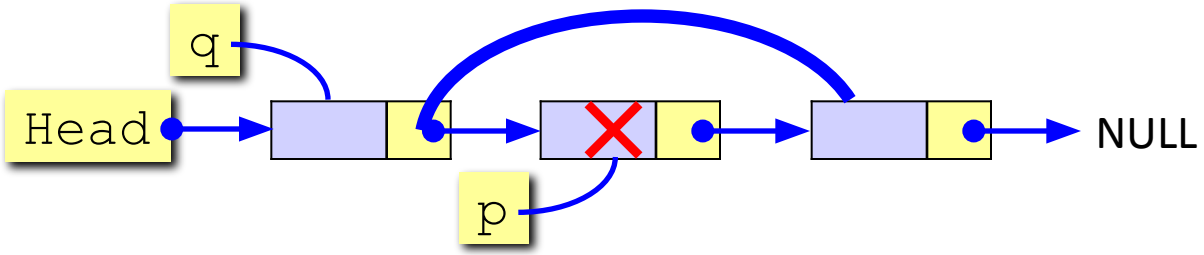
выход: адрес узла, перед которым нужно вставить это слово или `NULL`, если слово нужно вставить в конец списка.

```
PNode FindPlace ( PNode Head, char NewWord[] )
{
  PNode q = Head;
  while ( q  &&  strcmp(NewWord, q->word) > 0 )
    q = q->next;
  return q;
}
```

слово `NewWord` стоит по алфавиту *до* `q->word`

Удаление узла

Проблема: нужно знать адрес предыдущего узла q.




```
void DeleteNode ( PNode &Head, PNode p )
{
PNode q = Head;
if ( Head == p )
    Head = p->next;
else {
  while ( q  &&  q->next != p )
    q = q->next;
    if ( q == NULL ) return;
    q->next = p->next;
    }
delete p;
}
```

особый случай: удаляем первый узел

ищем предыдущий узел, такой что q->next == p

освобождение памяти

Алфавитно-частотный словарь

Алгоритм:

1) открыть файл на чтение;

```
FILE *in;
in = fopen("input.dat", "r");
```

read, чтение

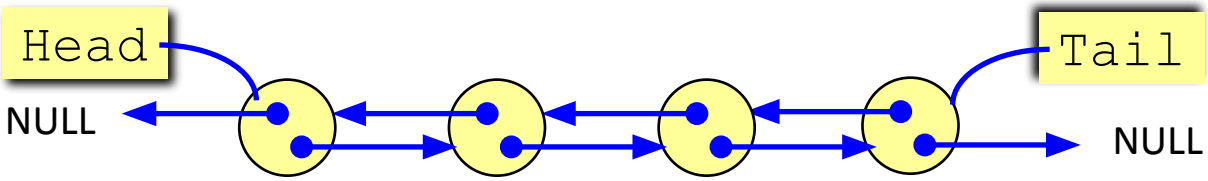
2) прочитать слово:

```
char word[80];
...
n = fscanf(in, "%s", word);
```

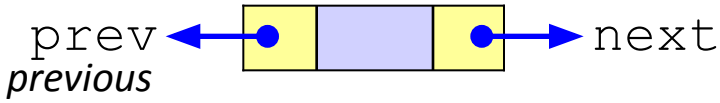
вводится только одно слово (до пробела)!

3) если файл закончился (`n!=1`), то перейти к шагу 7;
4) если слово найдено, увеличить счетчик (поле `count`);
5) если слова нет в списке, то
   - создать новый узел, заполнить поля (`CreateNode`);
   - найти узел, перед которым нужно вставить слово (`FindPlace`);
   - добавить узел (`AddBefore`);
6) перейти к шагу 2;
7) вывести список слов, используя проход по списку.

Двусвязные списки



Структура узла:




```
struct Node {
  char word[40];   // слово
  int  count;      // счетчик повторений
  Node *next;      // ссылка на следующий элемент
  Node *prev;      // ссылка на предыдущий элемент
  };
```

Указатель на эту структуру:

```
typedef Node *PNode;
```

Адреса «головы» и «хвоста»:

```
PNode Head = NULL;
PNode Tail = NULL;
```

можно двигаться в обе стороны

нужно правильно работать с двумя указателями вместо одного

Динамические структуры данных (язык Си)

Тема 5. Стеки, очереди, деки

Стек

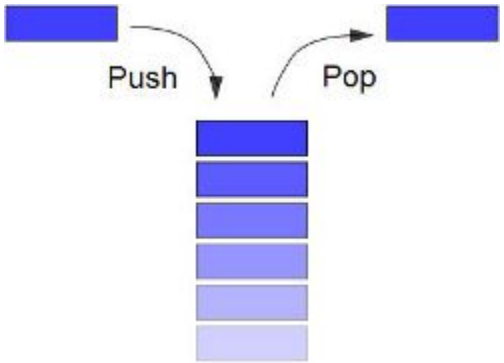
Стек – это линейная структура данных, в которой добавление и удаление элементов возможно только с одного конца (вершины стека). *Stack* = кипа, куча, стопка (англ.)

LIFO = *Last In – First Out*

«Кто последним вошел, тот первым вышел».

Операции со стеком:

1) добавить элемент на вершину (*Push* = втолкнуть)*;*
2) снять элемент с вершины *(Pop* = вылететь со звуком*).*



Пример задачи

Задача: вводится символьная строка, в которой записано выражение со скобками трех типов: `[ ]`, `{ }` и `( )`. Определить, верно ли расставлены скобки (не обращая внимания на остальные символы). Примеры:

`[ ( ) ] { }` `] [` `[ ( { ) ] }`

Упрощенная задача: то же самое, но с одним видом скобок.

Решение: счетчик вложенности скобок. Последовательность правильная, если в конце счетчик равен нулю и при проходе не разу не становился отрицательным.

(	(	)	)	(	)
1	2	1	0	1	0

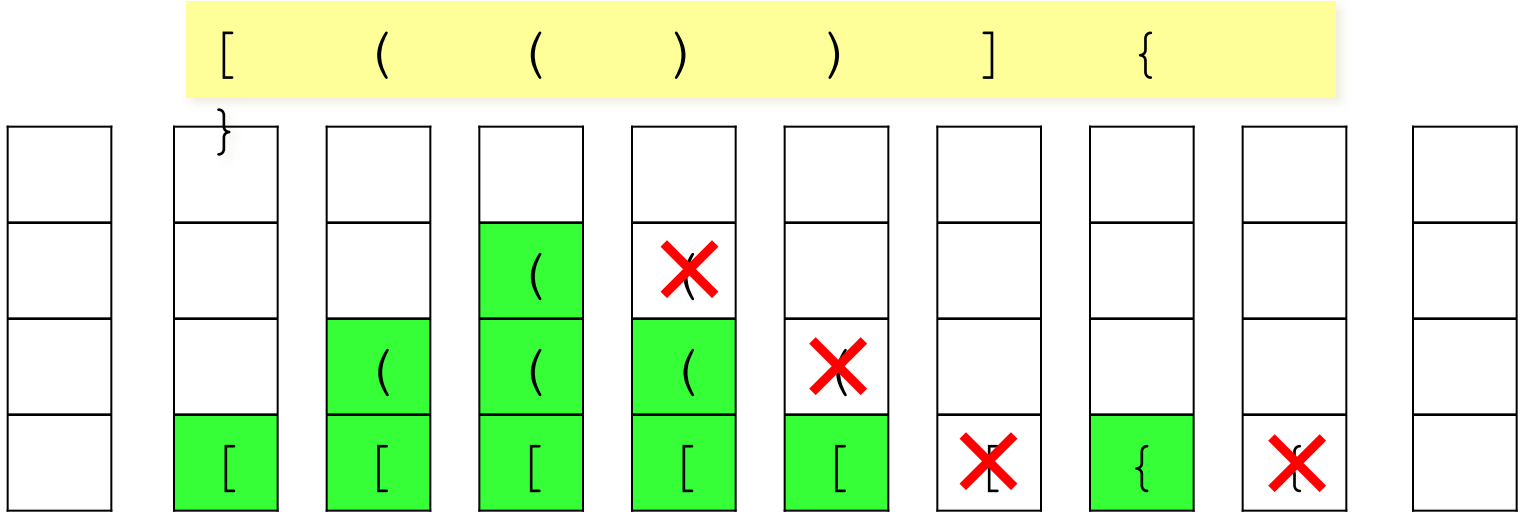
(	(	)	)	)	(
1	2	1	0	-1	0

(	(	)	)	(
1	2	1	0	1

? Можно ли решить исходную задачу так же, но с тремя счетчиками?

	[	(	{	)	]	}
(:	0		1		0	
[:	0	1				0
{ :	0			1		0

Решение задачи со скобками



Алгоритм:

1) в начале стек пуст;
2) в цикле просматриваем все символы строки по порядку;
3) если очередной символ – открывающая скобка, заносим ее на вершину стека;
4) если символ – закрывающая скобка, проверяем вершину стека: там должна быть соответствующая открывающая скобка (если это не так, то ошибка);
5) если в конце стек не пуст, выражение неправильное.

Реализация стека (массив)

Структура-стек:

```
const MAXSIZE = 100;
struct Stack {
   char data[MAXSIZE]; // стек на 100 символов
   int  size;          // число элементов
   };
```

Добавление элемента:

```
int Push ( Stack &S, char x )
{
if ( S.size == MAXSIZE ) return 0;
S.data[S.size] = x;
S.size ++;
return 1;
}
```

ошибка: переполнение стека

добавить элемент

нет ошибки

Реализация стека (массив)

Снятие элемента с вершины:

```
char Pop ( Stack &S )
{
if ( S.size == 0 ) return char(255);
S.size --;
return S.data[S.size];
}
```

ошибка:
стек пуст

Пусто й или нет?

```
int isEmpty ( Stack &S )
{
if ( S.size == 0 )
     return 1;
else return 0;
}
```

```
int isEmpty ( Stack &S )
{
return (S.size == 0);
}
```

Программа

```
void main()
{
  char br1[3] = { '(', '[', '{' };
  char br2[3] = { ')', ']', '}' };
  char s[80], upper;
  int i, k, error = 0;
  Stack S;
  S.size = 0;
  printf("Введите выражение со скобками > ");
  gets ( s );
  ...  // здесь будет основной цикл обработки
  if ( ! error && (S.size == 0) )
       printf("\nВыражение правильное\n");
  else printf("\nВыражение неправильное\n");
}
```

открывающие скобки

закрывающие скобки

то, что сняли со стека

признак ошибки

Обработка строки (основной цикл)

```
for ( i = 0; i < strlen(s); i++ )
  {
  for ( k = 0; k < 3; k++ )
    {
    if ( s[i] == br1[k] )  // если открывающая скобка
      {
      Push ( S, s[i] ); // втолкнуть в стек
      break;
      }
    if ( s[i] == br2[k] ) // если закрывающая скобка
      {
      upper = Pop ( S );  // снять верхний элемент
      if ( upper != br1[k] ) error = 1;
      break;
      }
    }
  if ( error ) break;
  }
```

цикл по всем символам строки s

цикл по всем видам скобок

ошибка: стек пуст или не та скобка

была ошибка: дальше нет смысла проверять

Реализация стека (список)

Структура узла:

```
struct Node {
   char data;
      Node *next;
      };
typedef Node *PNode;
```

Добавление элемента:

```
void Push (PNode &Head, char x)
{
   PNode NewNode = new Node;
   NewNode->data = x;
   NewNode->next = Head;
   Head = NewNode;
}
```

Реализация стека (список)

Снятие элемента с вершины:

```
char Pop (PNode &Head) {
  char x;
  PNode q = Head;
  if ( Head == NULL ) return char(255);
  x = Head->data;
  Head = Head->next;
  delete q;
  return x;
}
```

стек пуст

Изменения в основной программе:

```
Stack S;
S.size = 0;
...
if ( ! error && (S.size == 0) )
     printf("\nВыражение правильное\n");
else printf("\nВыражение неправильное \n");
```

PNode S = NULL;

(S == NULL)

Вычисление арифметических выражений

Как вычислять автоматически:

`(a+b) /(c+d-1)`

Инфиксная запись
(знак операции между операндами)

необходимы скобки!

Префиксная запись (знак операции до операндов)

`/ a + b c + d - 1`

польская нотация,
Jan Łukasiewicz (1920)

скобки не нужны, можно однозначно вычислить!

Постфиксная запись (знак операции после операндов)

`a + b c + d - 1 /`

обратная польская нотация,
F. L. BauerF. L. Bauer and E. W. Dijkstra

Запишите в постфиксной форме

```
(32*6-5)*(2*3+4)/(3+7*2)
```

```
(2*4+3*5)*(2*3+18/3*2)*(12-3)
```

```
(4-2*3)*(3-12/3/4)*(24-3*12)
```

Вычисление выражений

Постфиксная форма:

X = a b + c d + 1 – /

					d		1		
		b		c	c	c+d	c+d	c+d-1	
	a	a	a+b	a+b	a+b	a+b	a+b	a+b	X

Алгоритм:

1) взять очередной элемент;
2) если это не знак операции, добавить его в стек;
3) если это знак операции, то
   - взять из стека два операнда;
   - выполнить операцию и записать результат в стек;
4) перейти к шагу 1.

Системный стек (*Windows – 1 Мб*)

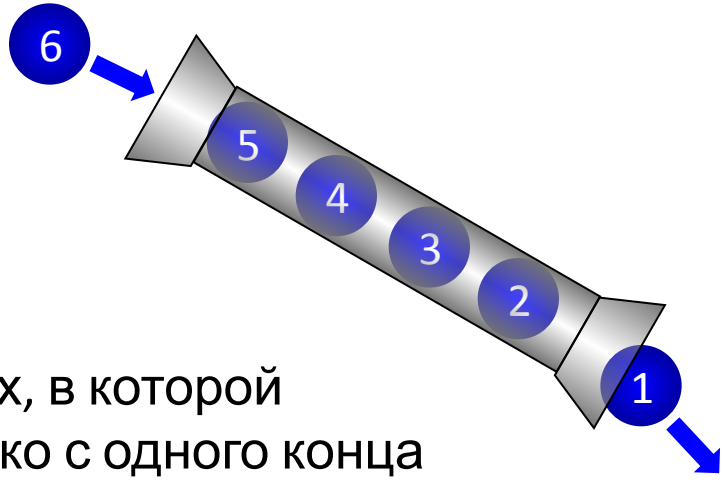
Используется для

1) размещения локальных переменных*;*
2) хранения адресов возврата (по которым переходит программа после выполнения функции или процедуры);
3) передачи параметров в функции и процедуры;
4) временного хранения данных (в программах на языке *Ассмеблер*).

Переполнение стека *(stack overflow):*

1) слишком много локальных переменных (выход – использовать динамические массивы)*;*
2) очень много рекурсивных вызовов функций и процедур (выход – переделать алгоритм так, чтобы уменьшить глубину рекурсии или отказаться от нее вообще).

Очередь



Очередь – это линейная структура данных, в которой добавление элементов возможно только с одного конца (конца очереди), а удаление элементов – только с другого конца (начала очереди).

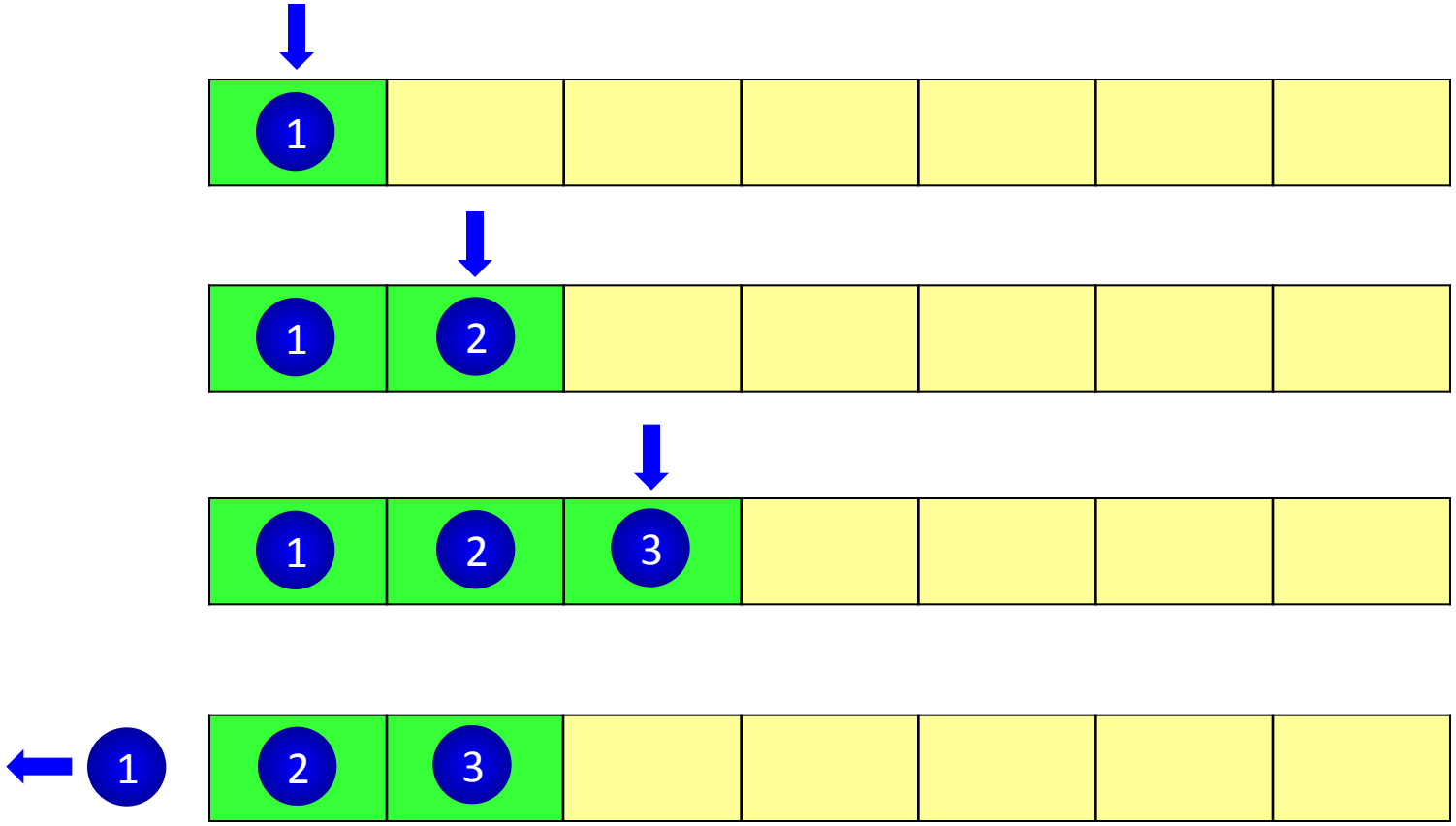
FIFO = *First In – First Out*

«Кто первым вошел, тот первым вышел».

Операции с очередью:

1) добавить элемент в конец очереди (*PushTail* = втолкнуть в конец)*;*
2) удалить элемент с начала очереди *(Pop).*

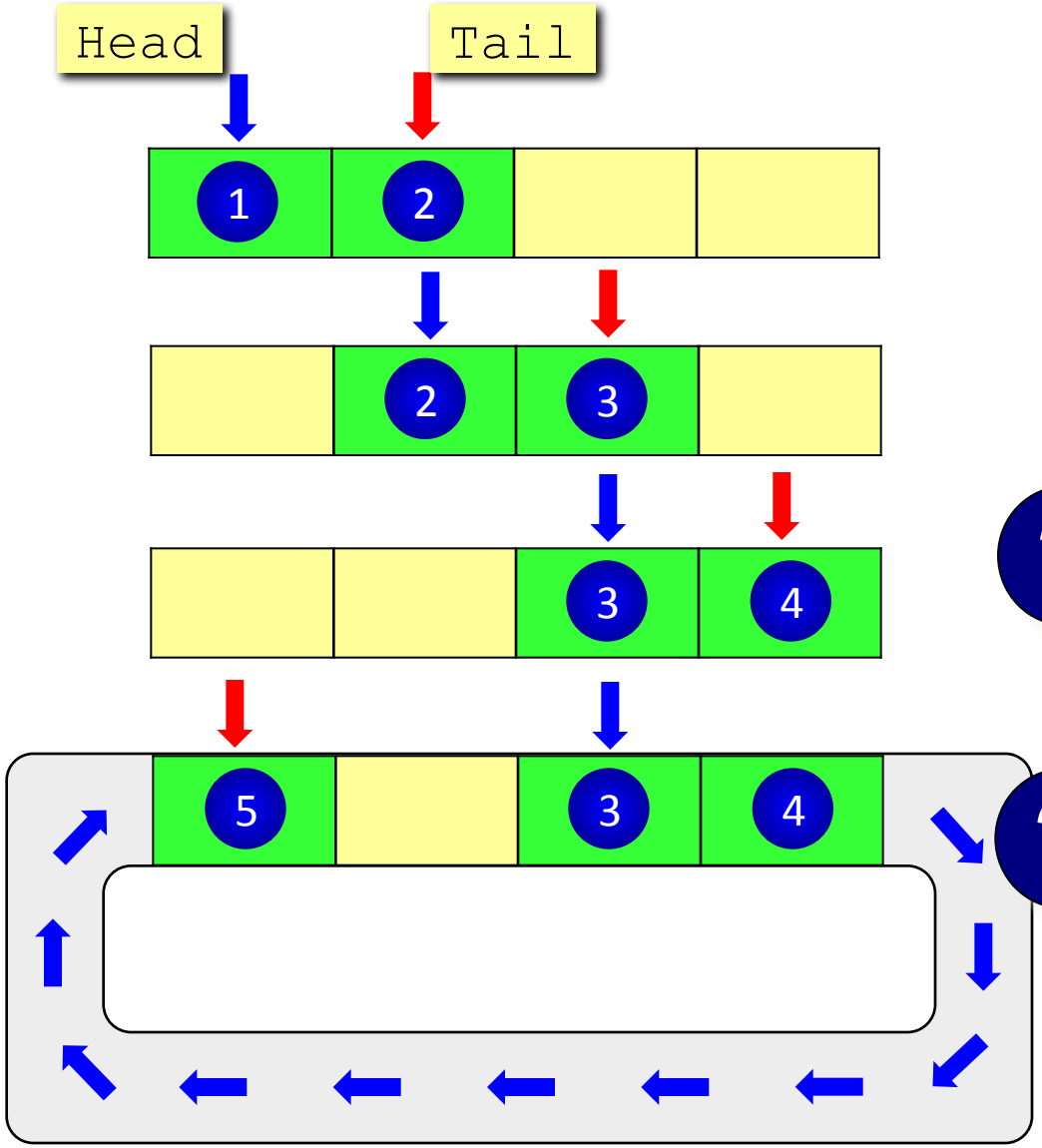
Реализация очереди (массив)

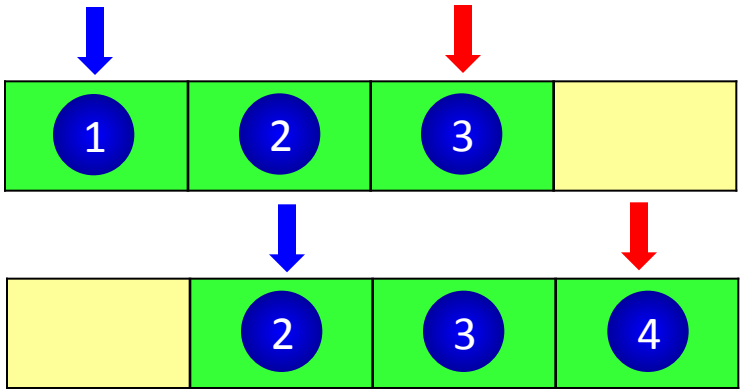


самый простой способ

1) нужно заранее выделить массив;
2) при выборке из очереди нужно сдвигать все элементы.

Реализация очереди (кольцевой массив)



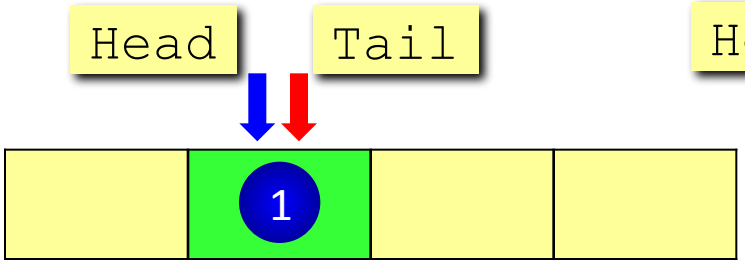


Сколько элементов можно хранить в такой очереди?

Как различить состояния «очередь пуста» и «очередь полна»?

Реализация очереди (кольцевой массив)

В очереди 1 элемент:

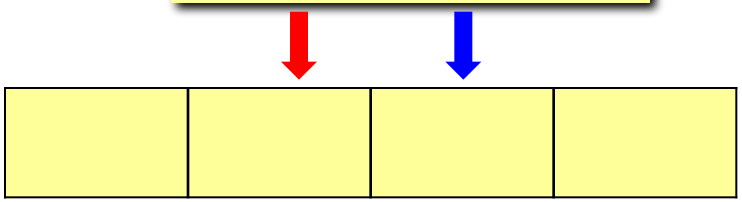


`Head == Tail`

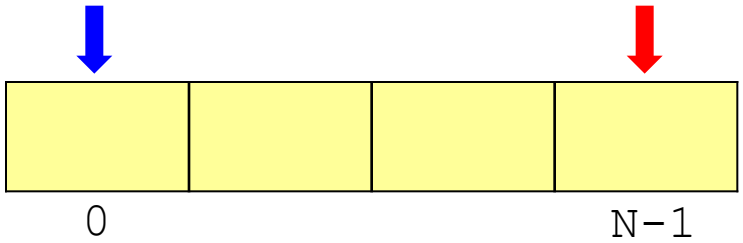
размер массива

Очередь пуста:

`Head == Tail + 1`

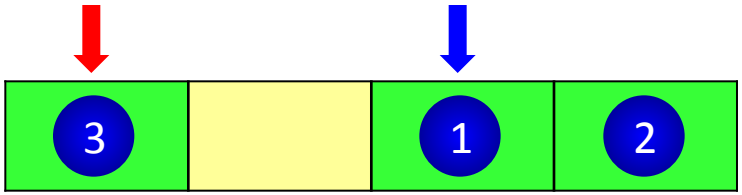


`Head == (Tail + 1) % N`

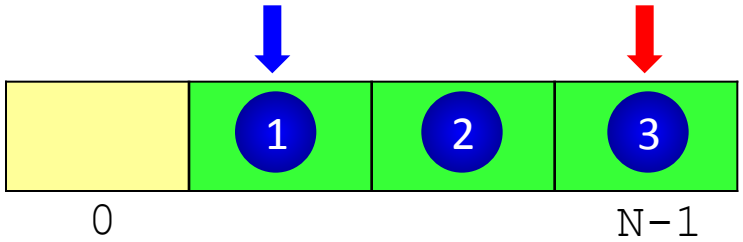


Очередь полна:

`Head == Tail + 2`



`Head == (Tail + 2) % N`



Реализация очереди (кольцевой массив)

Структура данных:

```
const MAXSIZE = 100;
struct Queue {
   int data[MAXSIZE];
   int head, tail;
   };
```

замкнуть в кольцо

Добавление в очередь:

```
int PushTail ( Queue &Q, int x )
{
   if ( Q.head == (Q.tail+2) % MAXSIZE )
      return 0;
   Q.tail = (Q.tail + 1) % MAXSIZE;
   Q.data[Q.tail] = x;
   return 1;
}
```

удачно добавили

очередь полна, не добавить

Реализация очереди (кольцевой массив)

Выборка из очереди:

```
int Pop ( Queue &Q )
{
   int temp;
   if ( Q.head == (Q.tail + 1) % MAXSIZE )
     return 32767;
   temp = Q.data[Q.head];
   Q.head = (Q.head + 1) % MAXSIZE;
   return temp;
}
```

очередь пуста

взять первый элемент

удалить его из очереди

Реализация очереди (списки)

Структура узла:

```
struct Node {
   int  data;
   Node *next;
   };
typedef Node *PNode;
```

Тип данных «очередь»:

```
struct Queue {
   PNode Head, Tail;
   };
```

Реализация очереди (списки)

Добавление элемента:

```
void PushTail ( Queue &Q, int x )
{
    PNode NewNode;
    NewNode = new Node;
    NewNode->data = x;
    NewNode->next = NULL;
    if ( Q.Tail )
         Q.Tail->next = NewNode;
    Q.Tail = NewNode;
    if ( Q.Head == NULL )
         Q.Head = Q.Tail;
}
```

создаем новый узел

если в списке уже что-то было, добавляем в конец

если в списке ничего не было, …

Реализация очереди (списки)

Выборка элемента:

```
int Pop ( Queue &Q )
{
   PNode top = Q.Head;
   int x;
   if ( top == NULL )
      return 32767;
   x = top->data;
   Q.Head = top->next;
   if ( Q.Head == NULL )
        Q.Tail = NULL;
   delete top;
   return x;
}
```

если список пуст, …

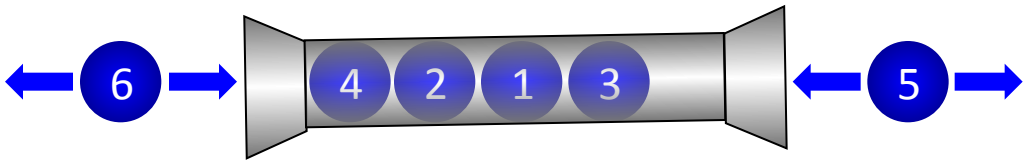
запомнили первый элемент

если в списке ничего не осталось, …

освободить память

Дек

Дек (*deque = double ended queue*, очередь с двумя концами) – это линейная структура данных, в которой добавление и удаление элементов возможно с обоих концов.



Операции с деком:

1) добавление элемента в начало *(Push);*
2) удаление элемента с начала *(Pop)*;
3) добавление элемента в конец *(PushTail)*;
4) удаление элемента с конца *(PopTail)*.

Реализация:

1) кольцевой массив*;*
2) двусвязный список.