Лекция 4.
Файлы записей.
Структуры данных

Локальные и глобальные переменные (объекты)

Время жизни переменной/объекта может быть глобальным и локальным.

Область видимости переменной (объекта или функции) определяет набор функций или модулей, внутри которых допустимо использование имени этой переменной. Область видимости переменной начинается в точке объявления переменной.

Глобальными называют переменные/объекты, объявление которых дано вне функции. Они доступны (видимы) во всем файле, в котором они объявлены. В течение всего времени выполнения программы с глобальной переменной ассоциирована некоторая ячейка памяти.

Локальными называют переменные, объявление которых дано внутри блока или функции. Эти переменные доступны только внутри того блока, в котором они объявлены. Переменным с локальным временем жизни выделяется новая ячейка памяти каждый раз при вызове функции, в которой они объявлены. Когда выполнение функции завершается, память, выделенная под локальную переменную, освобождается, и переменная теряет своё значение.

```
#include <stdio.h>

void func()
{
   int k = 1;                                  // локальная переменная
   printf(" \n k = %d ", k);
   k = k + 1;
}

int main()
{
   for (int i = 0; i <= 5; i++)   //область видимости i- цикл
   func();
   return 0;
}
```

```
#include <stdio.h>

int x, y, z;                    // глобальные переменные

void sum( )
{
   z = x + y;   }

int main()
{
        printf("x= ");
        scanf("%d", &x);
        printf("y= ");
        scanf("%d", &y);
        sum();
        printf("z= %d", z);
        return 0;
}
```

```
#include <stdio.h>
int sum(int x, int y)   {
   int z;
   z = x + y;
   return z;    }

int main()  {
    int x, y, z;
          printf("x= ");
          scanf("%d", &x);
          printf("y= ");
          scanf("%d", &y);
          z=sum(x,y);        //передача параметров по значению
          printf("z= %d", z);
          return 0;
}
```

```
int sum(int x, int y) {
    return x+y;
 }
```

```
#include <stdio.h>
void sum(int x, int y, int* z)
{
   *z = x + y;
 }
int main()
{
    int x, y, z;
          printf("x= ");
          scanf("%d", &x);
          printf("y= ");
          scanf("%d", &y);
          sum(x, y, &z);        //передача параметров по адресу
          printf("z= %d", z);
          return 0;
}
```

```
void sum(int* x, int* y, int* z)
{
 *z = *x + *y;
 }

 …
 sum(&x,&y,&z);
```

Массивы в параметрах функции

void print(int numbers[]);
void print(int[]);

void print(int *numbers);
void print(int*);

```
#include <stdio.h>

void print(int[]);

int main()
{
        int nums[]= {1, 2, 3, 4, 5};
        print(nums);
}

void print(int numbers[])                    //  void print(int *numbers)
{
         printf( "First number:  " ,  numbers[0] );
}
```

Структуры данных. Линейный список

Линейный список - это упорядоченная последовательность переменного числа элементов.

X[1], X[2], … , X[n], n≥0

1. Если n > 0, то X[1] – первый элемент;
2. Если 1< k <n , то X[k-1], X[k], X[k+1];
3. X[n] - последний элемент.

Основные операции с линейным списком:

1. получить доступ к k-му элементу;
2. включить k-й элемент;
3. исключить k-й элемент;
4. найти необходимый элемент.

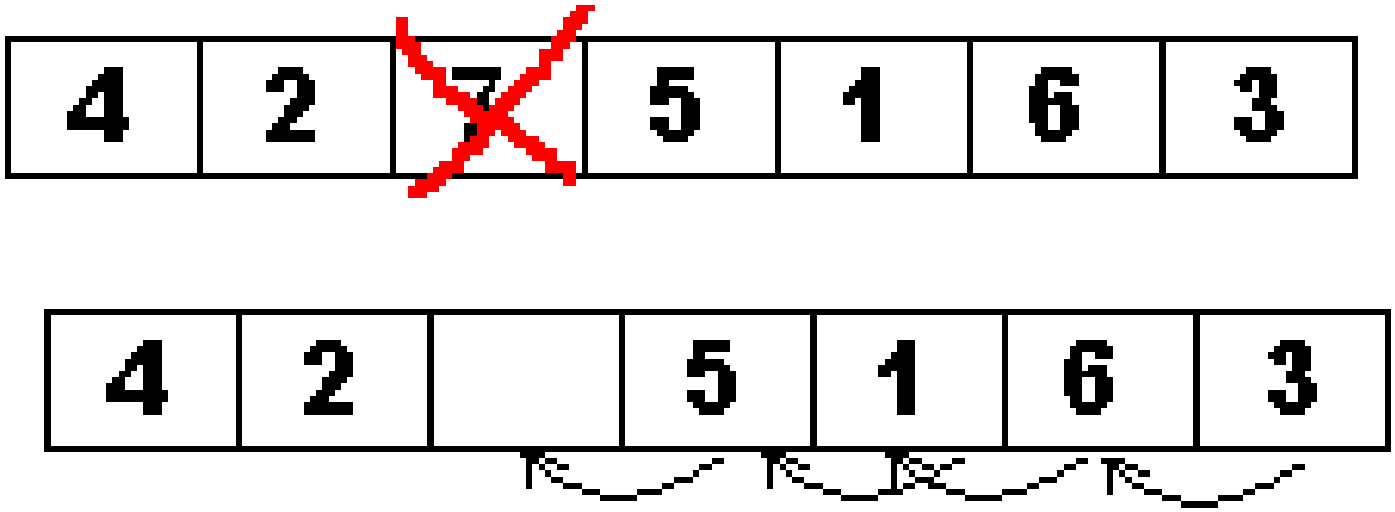
Способы представления линейных списков

Последовательное - элементы списка расположены в последовательных ячейках памяти.

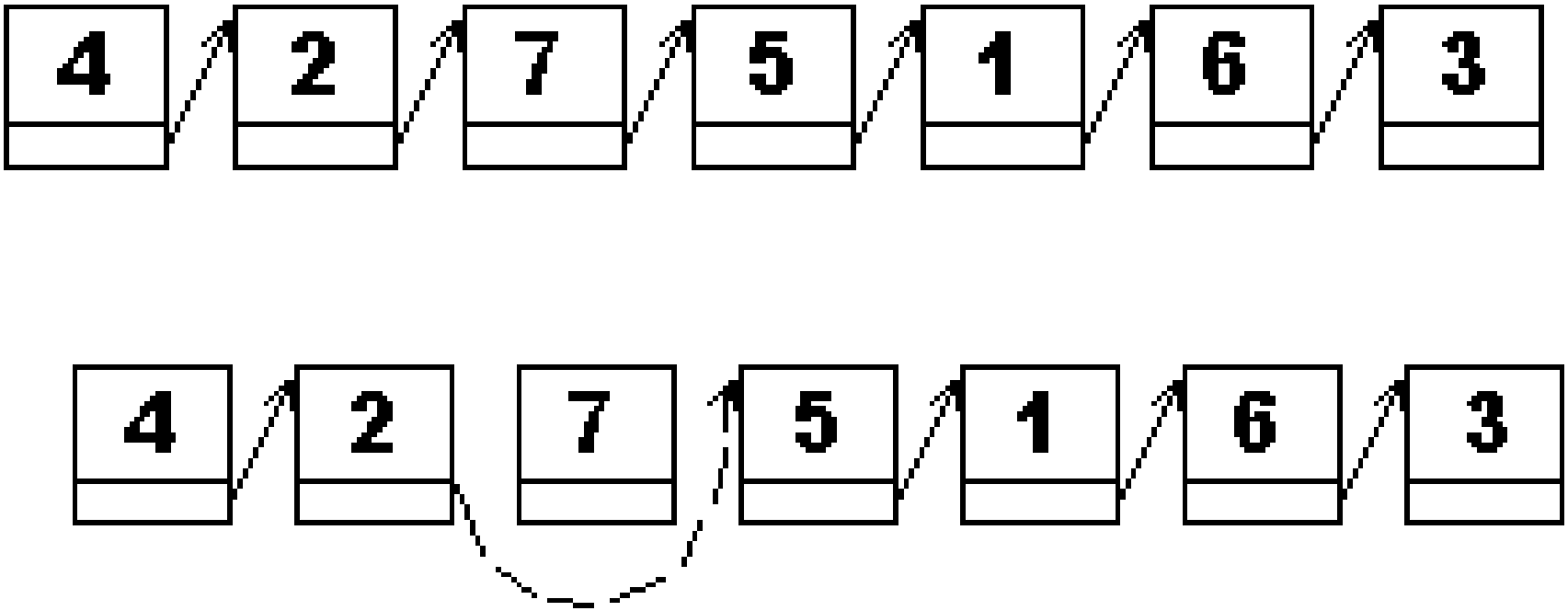
Связное (связанное) представление – элементы списка расположены в произвольных ячейках памяти. Требуется дополнительная память для хранения связей.

- Доступ к k-му элементу быстрее при последовательном представлении.
- При связанном представлении легко включать и исключать элементы, при последовательном для этого необходимо сдвигать информацию.
- При связанном представлении легко объединять списки и разбивать их на части.

Линейный список в виде одномерного массива



Линейный список в динамической памяти

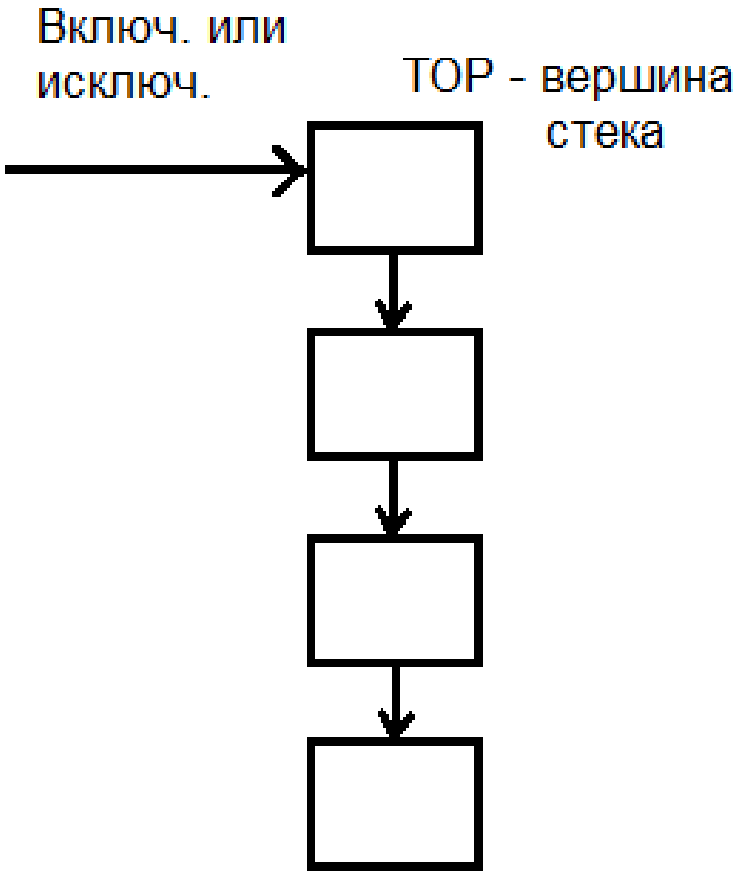


На практике часто применяются некоторые частные случаи линейных списков:

- **Стек** (**LIFO** (Last In First Out) список) - это линейный список, в котором все включения и исключения (и обычно всякий доступ) элементов происходят на одном конце списка.
- **Очередь** (**FIFO** (First In First Out) список) - это линейный список, в котором все включения элементов происходят на одном конце списка, а все исключения (и обычно всякий доступ) - на другом.
- **Дек** (double ended queue) - это линейный список, в котором все включения и исключения элементов производятся на обоих концах списка.

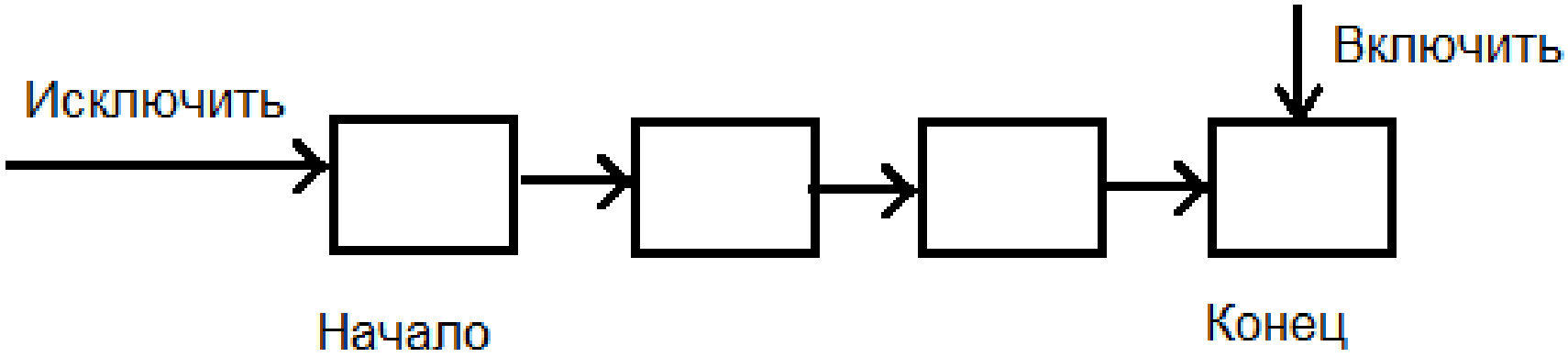
Некоторые специальные списки

Стек LIFO



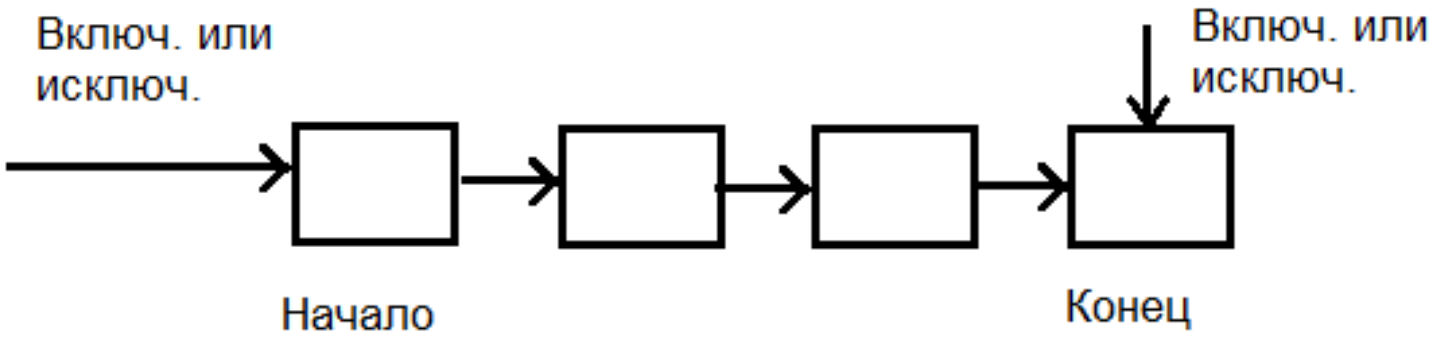
Некоторые специальные списки

Очередь FIFO



Некоторые специальные списки

Дек (deque)



Основные функции для работы со структурами данных

- **push** — добавить элемент в структуру данных;
- **pop** — удалить элемент из структуры данных;
- **peek** — получить значение элемента без его удаления.
- **isEmpty** — проверить на наличие элементов.
- **size** — возвращает количество элементов в структуре данных.

Реализация стека в виде массива

```
1.  include<stdio.h>
2.   int Stack[100], TOP=0 ;
3.   void push(int);
4.   void pop(*int);
```

Реализация стека в виде массива
Функция включения элемента в стек

```
1.  void push (int y)
2.    {
3.       if( TOP>=100 )
4.       {   printf(“Переполнение стека!”);
5.           return;  }
6.       Stack[TOP] = y;
7.       TOP++;
8.    }
```

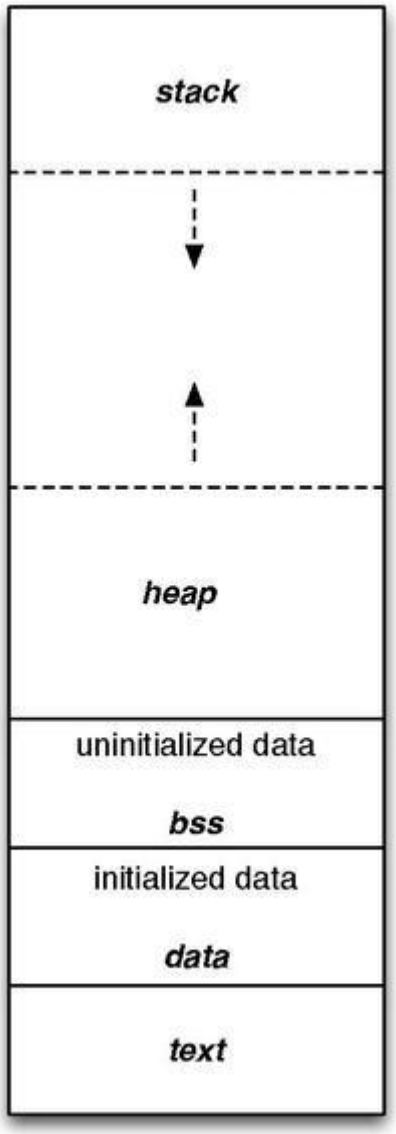
Реализация стека в виде массива. Функция исключения элемента из стека

```
1.  void pop (int*  y)
2.   {
3.      if( TOP==0 )
4.       {   printf(“Стек пуст!”);
5.           return;  }
6.     *y = Stack[TOP];
7.      TOP--;
8.  }
```

Реализация стека в виде массива

```
1.  int main(  )
2.   {
3.       int y, i;
4.       for (i=0; i<10; i++)
5.       {
6.          scanf(“%d”, &y);
7.           push(y);
8.       }
9.       pop(&y);
10.      printf(“%d”, y);
11.   }
```

Адресное пространство процесса



СТЕК (локальные переменные)	Старшие адреса
ДИНАМИЧЕСКАЯ ПАМЯТЬ (куча)	
РАЗДЕЛ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ И КОНСТАНТ	
КОД ПРОГРАММЫ	Младшие адреса

Динамическое выделение памяти

```
#include <stdlib.h>
void *malloc(size_t size);  // выделить память размера size*тип
void free(void *ptr);       //освободить память по указателю
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
void *reallocarray(void *ptr, size_t nmemb, size_t size);
```

Вызов **calloc(nmemb, size)** аналогичен вызову **malloc(nmemb * size)**, но при использовании calloc:

- выделенная память обнуляется,
- переполнение nmemb * size приводит к ошибке, а не к выделению неправильного количества памяти.

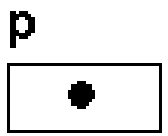
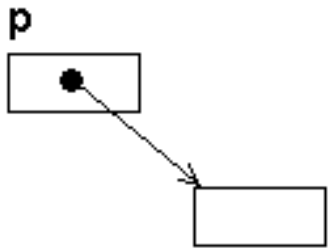
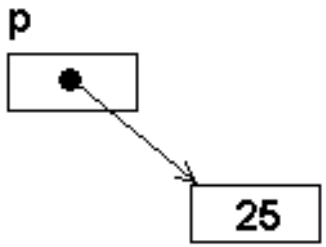
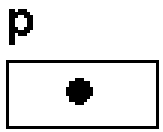
realloc(ptr, size) изменяет размер ранее выделенного блока памяти ptr, выполняя копирование и освобождение старого блока, если дополнить его не получается. Возвращаемый указатель может отличаться или не отличаться от ptr!

```
int *x = (int*) malloc(10);
int *y = (int*) calloc(10, sizeof(int));
```

```
1. char *c;
2. // Выделить память 10 байт
3. c=(char *)malloc(10);
4. // или
5. c=(char *)malloc(sizeof(char)*10);
```

По мере работы программы выделяется все больше памяти, которая никогда не освобождается. Указатели на выделенную память заменяются другими значениями, что делает освобождение невозможным.

Использование функций malloc() и free()

int *p;	
p=(int*)malloc(sizeof(int));	
*p = 25; printf(“%d”, *p);	
free(p);	

Создание массива в динамической памяти

```
1.  int *p, i;
2.  p = (int *) malloc(100 * sizeof(int));
3.  if (p==NULL)  {
4.        printf("Недостаточно памяти\n");
5.        return 0;     }
6.  for (i = 0; i < 100; i++)
7.        *(p+i) = i;
8.  for (i = 0; i < 100; i++)
9.        printf("%d ", p[i] );
10.  free(p);
```

Элементы языка C++ . Поточный ввод-вывод в C++

В C++ используется библиотека ввода-вывода **iostream.**

#include <iostream> // поток ввода/вывода

Библиотека **iostream** определяет три стандартных потока:

- **cin** стандартный входной поток (**stdin** в C)
- **cout** стандартный выходной поток (**stdout** в C)
- **cerr** стандартный поток вывода сообщений об ошибках (**stderr** в C)

Строка **using namespace std;** объявляет импорт всего пространства имен **std**. Это пространство имен содержит все имена из стандартной библиотеки языка C++, такие как поток вывода **cin** или класс **string**. В противном случае каждый вызов функций **cin**, **cout** будет дополняться пространством имен **std** и оператором разрешения контекста **::** .

```
#include <iostream>

int main() {

   std::string  name;

   std::cout << "Enter your name: ";

   std::cin >> name;

//С помощью одной команды cout можно вывести несколько
//значений

   std::cout << "Hello, " << name << "! \n";

   return 0;  }
```

```
#include <iostream>

int main()
{
   int age;
   double weight;
   std::cout << "Input age: ";
   std::cin >> age;
   std::cout << "Input weight: ";
   std::cin >> weight;
   std::cout << "Age: " << age << "\t Weight: " << weight << std::endl;
}
```

Перегруженные операторы сдвига влево/вправо

<< - **двигать** информацию в поток

>> - **двигать** информацию из потока

```
#include <iostream>

 // using namespace std; можно использовать глобально

int main()
{
   using namespace std;
   int age;
   double weight;
   cout << "Input age: ";
   cin >> age;
   cout << "Input weight: ";
   cin >> weight;
   cout << "Age: " << age << "\t Weight: " << weight << endl;
}
```

:: - оператор раскрытия области видимости. Если перед ним (слева) не указана какая-либо область видимости, то подразумевается глобальная область (глобальное пространство имен).

```
#include <iostream>

void foo() {
    std::cout << "function: foo()" << std::endl;   }

int global_a = 5;

int main() {
     int global_a = 10;
     :: foo();
      std::cout << ::global_a << " " << global_a << std::endl;
      return 0;
}
```

function: foo()
5 10