**Практическая работа №16**

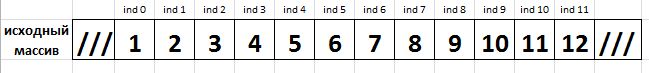
**Тема:** **Двоичный поиск в массиве (Дихотомия)**

**Цель работы**: Приобретение практических навыков в работе с одномерными массивами в языке С++. Организация двоичного (бинарного) поиска в массиве программными средствами.

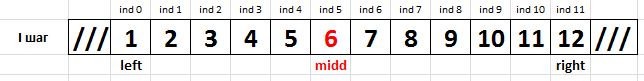
**Краткие теоретические сведения**

Двоичный (бинарный) поиск является более эффективным решением в случае, если массив заранее отсортирован.

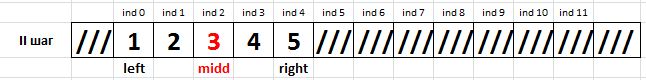
Предположим, что массив из 12-ти элементов отсортирован по возрастанию:

[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c.jpg)

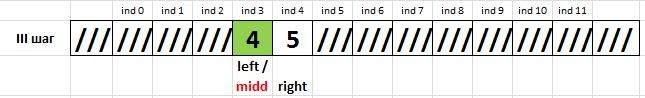
Пользователь задает искомое значение (ключ поиска). Допустим 4. На первой итерации [массив](https://purecodecpp.com/archives/1) делится на две части (ищем средний элемент – midd): (0 + 11) / 2 = 5 (0.5 отбрасываются). Сначала, проверяется значение среднего элемента массива. Если оно совпадает с ключом – алгоритм прекратит работу и программа выведет сообщение, что значение найдено. В нашем случае, ключ не совпадает со значением среднего элемента.

[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c-2.jpg)

Если ключ меньше значения среднего элемента, алгоритм не будет проводить поиск в той половине массива, которая содержит значения больше ключа (т.е. от среднего элемента до конца массива). Правая граница поиска сместится (midd – 1). Далее снова деление массива на 2.

[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c-31.jpg)

Ключ снова не равен среднему элементу. Он больше него. Теперь левая граница поиска сместится (midd + 1).

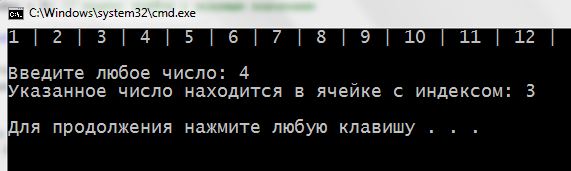
[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c-4.jpg)

На третьей итерации средний элемент – это ячейка с индексом 3: (3 + 4) / 2 = 3. Он равен ключу. Алгоритм завершает работу.

**Пример 1,2,3 (оценка 3):**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | #include <iostream>  using namespace std;    // функция с алгоритмом двоичного поиска  int Search\_Binary (int arr[], int left, int right, int key)  {  int midd = 0;  while (1)  {  midd = (left + right) / 2;  if (key < arr[midd])       // если искомое меньше значения в ячейке  right = midd - 1;      // смещаем правую границу поиска  else if (key > arr[midd])  // если искомое больше значения в ячейке  left = midd + 1;    // смещаем левую границу поиска  else                       // иначе (значения равны)  return midd;           // функция возвращает индекс ячейки    if (left > right)          // если границы сомкнулись  return -1;  }  }    int main()  {  setlocale (LC\_ALL, "rus");  const int SIZE = 12;  int array[SIZE] = {};  int key = 0;  int index = 0; // индекс ячейки с искомым значением    for (int i = 0; i < SIZE; i++) // заполняем и показываем массив  {  array[i] = i + 1;  cout << array[i] << " | ";  }  cout << "\n\nВведите любое число: ";  cin >> key;  index = Search\_Binary (array, 0, SIZE, key);    if (index >= 0)  cout << "Указанное число находится в ячейке с индексом: " << index << "\n\n";  else  cout << "В массиве нет такого числа!\n\n";  return 0;  } |

Результат:

[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c-51.jpg)

**Как создать бинарный поиск в C++**

Давайте посмотрим как работает бинарный поиск на примере. В примере ниже в строке 9 мы создали массив **arr** на 10 элементов и в строке 12 предложили пользователю с клавиатуры заполнить его ячейки.

В строке 20 мы предлагаем пользователю ввести ключ (который нужно будет найти в массиве), а дальше мы с бинарным поиском проверим массив на наличие введенного ключа пользователем. Если мы найдем ключ в массиве, то выведем индекс ячейки, в которой находится ключ.

**Пример 2**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | #include <iostream>  #include <algorithm>    using namespace std;    int main() {    setlocale(LC\_ALL, "rus");      int arr[10]; // создали массив на 10 элементов    int key; // создали переменную в которой будет находиться ключ      cout << "Введите 10 чисел для заполнения массива: " << endl;    for (int i = 0; i < 10; i++) {      cin >> arr[i]; // считываем элементы массива    }      sort (arr, arr + 10); // сортируем с помощью функции sort (быстрая сортировка)      cout << endl << "Введите ключ: ";      cin >> key; // считываем ключ      bool flag = false;    int l = 0; // левая граница    int r = 9; // правая граница    int mid;      while ((l <= r) && (flag != true)) {      mid = (l + r) / 2; // считываем срединный индекс отрезка [l,r]        if (arr[mid] == key) flag = true; //проверяем ключ со серединным элементом      if (arr[mid] > key) r = mid - 1; // проверяем, какую часть нужно отбросить      else l = mid + 1;    }      if (flag) cout << "Индекс элемента " << key << " в массиве равен: " << mid;    else cout << "Извините, но такого элемента в массиве нет";      system("pause");    return 0;  } |

Бинарный поиск находится в строках 29 — 35. В строке 27 мы создали переменную mid, в которой будет **храниться индекс среднего элемента** (из отрезка [l, r]). В строке 27 считываем средний элемент отрезка [l, r] в переменную mid, по формуле: (l + r) / 2 (в которой l — левая граница, r — правая граница). В строке 29 проверяем условие arr[mid] == key:

* Если результат условия равен true, то булевой переменной flag присваиваем значение true (это значит, что мы нашли ключ).

В строке 30 мы проверяем условие arr[mid] > key:

* Если значение arr[mid] больше ключа, то переменной r присваиваем значение mid. Потому что проверять *верхнюю часть* не имеет смысла, так как ключ может находится только в ячейках ниже индекса mid (это если массив отсортирован по возрастанию).
* Если значение arr[mid] меньше ключа, то переменной l присваиваем значение mid. Потому что проверять *нижнюю часть* не имеет смысла, так как ключ может находится только в ячейках выше индекса mid (это если массив отсортирован по возрастанию).

В строке 37 мы проверяем flag:

* Если flag равен true, значит мы нашли ключ, а значит выводим «Индекс элемента key в массиве равен mid«.
* Если flag равен false, то выводим «Извините, но такого элемента в массиве нет».

Давайте выполним этот код:

**Результат:**

**bin\_search.cpp**

**Напишите 10 чисел для заполнения массива:**

**17 20 26 31 44 54 55 65 77 93**

**Введите ключ : 54**

**Индекс элемента 54 в массиве равен: 5**

**Process returned 0 (0x0) execution time : 0.005 s**

**Press any key to continue.**

**А если такого ключа нет в массиве :**

**bin\_search.cpp**

**Напишите 10 чисел для заполнения массива:**

**17 20 26 31 44 54 55 65 77 93**

**Введите ключ : 18**

**Извините, но такого элемента в массиве нет**

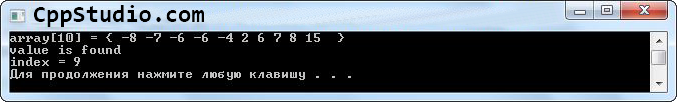
**Process returned 0 (0x0) execution time : 0.005 s**

**Press any key to continue.**

**Пример 3**

Разработаем программу, в которой объявим одномерный массив, и организуем двоичный поиск. Объявленный массив нужно инициализировать некоторыми значениями, причём так, чтобы эти значения были упорядочены.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | **// binary\_search.cpp: определяет точку входа для консольного приложения.**    **//#include "stdafx.h"**  **#include <iostream>**  **using namespace std;**    **int main(int argc, char\* argv[])**  **{**  **const int size\_array = 10;**  **int array\_[size\_array] = {-8, -7, -6, -6, -4, 2, 6, 7, 8, 15 }; // объявление одномерного массива**  **cout << "array[" << size\_array << "] = { ";**  **for (int counter = 0; counter < size\_array; counter++)**  **{**  **cout << array\_[counter] << " "; // печать элементов одномерного массива array1**  **}**  **cout << " }";**  **int average\_index = 0, // переменная для хранения индекса среднего элемента массива**  **first\_index   = 0, // индекс первого элемента в массиве**  **last\_index    = size\_array -1, // индекс последнего элемента в массиве**  **//--------------------------------------------------------**  **search\_value  = 15; // искомое (ключевое) значение**  **//--------------------------------------------------------**  **if (last\_index == -1) cout << "\narray is empty" << endl; // массив пуст**    **while (first\_index < last\_index)**  **{**  **average\_index = first\_index + (last\_index - first\_index) / 2; // меняем индекс среднего значения**  **search\_value <= array\_[average\_index] ? last\_index = average\_index : first\_index = average\_index + 1;    // найден ключевой элемент или нет**  **}**  **if ( array\_[last\_index] == search\_value)**  **cout << "\nvalue is found" << "\nindex = " << last\_index << endl;**  **else**  **cout << "\nvalue is not found" << endl;**  **system("pause");**  **return 0;**  **}** |



**Пример 4 (1,2,3,4 – оценка 4)**

**Бинарный поиск можно оптимизировать, как и все другие алгоритмы. Для его оптимизации можно уменьшить условие цикла ПОКА(while):**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | while (l < r) {      mid = (l + r) / 2; // считываем срединный индекс отрезка [l,r]        if (arr[mid] > key) r = mid; // проверяем, какую часть нужно отбросить с поиска      else l = mid + 1;  }    r--; // уменьшаем на один    if (arr[r] == key) cout << "Индекс элемента " << key << " в массиве равен: " << r;  else cout << "Извините, но такого элемента в массиве нет"; |

**ПРИМЕР 5,6 на оценку 5**

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

  setlocale(LC\_ALL, "rus");

  int arr[10]; // создали массив на 10 элементов

  int key; // создали переменную в которой будет находиться ключ

  cout << "Введите 10 чисел для заполнения массива: " << endl;

  for (int i = 0; i < 10; i++) {

    cin >> arr[i]; // считываем элементы массива

  }

  sort (arr, arr + 10); // сортируем с помощью функции sort (быстрая сортировка)

  cout << endl << "Введите ключ: ";

  cin >> key; // считываем ключ

  bool flag = false;

  int l = 0; // левая граница

  int r = 9; // правая граница

  int mid;

  while ((l <= r) && (flag != true)) {

    mid = (l + r) / 2; // считываем срединный индекс отрезка [l,r]

    if (arr[mid] == key) flag = true; //проверяем ключ со серединным элементом

    if (arr[mid] > key) r = mid - 1; // проверяем, какую часть нужно отбросить

    else l = mid + 1;

  }

  if (flag) cout << "Индекс элемента " << key << " в массиве равен: " << mid;

  else cout << "Извините, но такого элемента в массиве нет";

  system("pause");

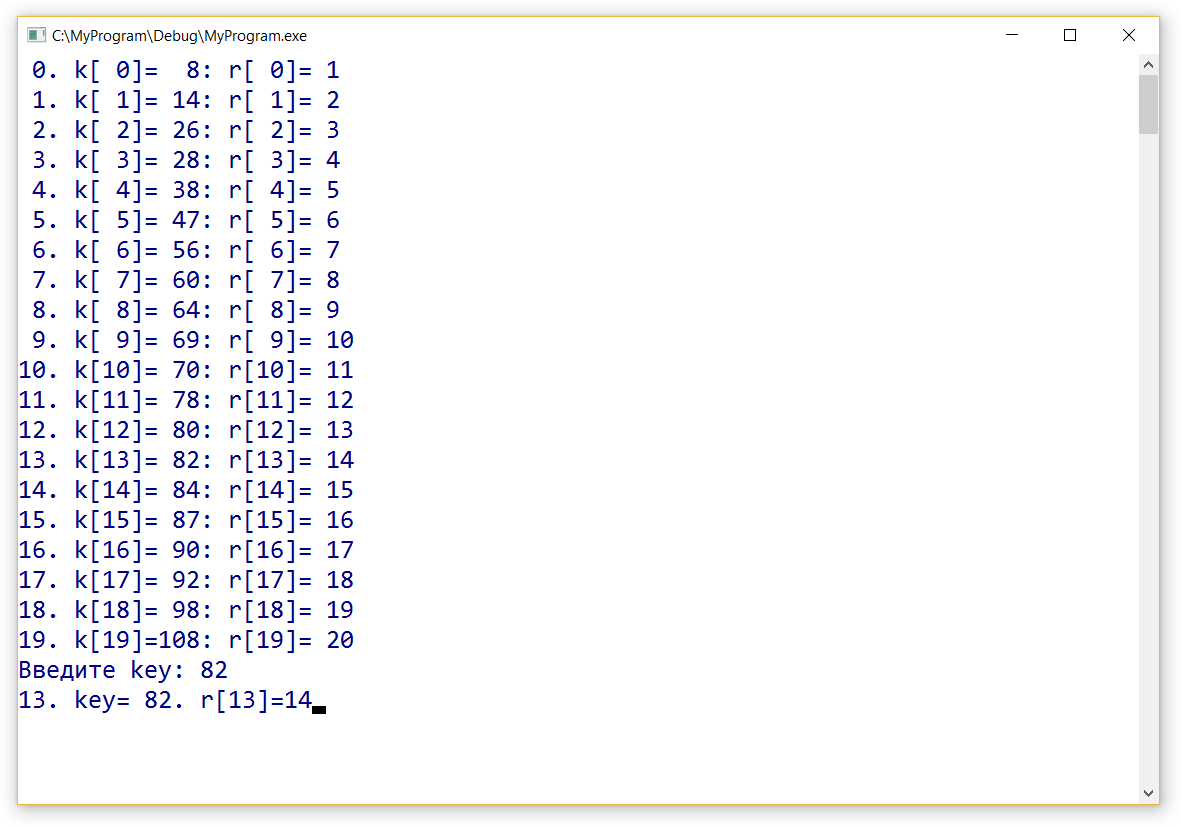
  return 0;

}

**Пример №6**

Реализация бинарного поиска

#define \_CRT\_SECURE\_NO\_WARNINGS // для корректной работы scanf()  
#include <stdio.h>  
#include <stdlib.h> // для использования функций system()  
int main()  
{  
  int k[20]; // массив ключей основной таблицы  
  int r[20]; // массив записей основной таблицы  
  int key, i;  
  //system("chcp 1251"); // перевод русского языка в консоли  
//  system("cls");     // очистка окна консоли  
  // Инициализация ключевых полей упорядоченными значениями  
  k[0] = 8;  k[1] = 14;  
  k[2] = 26;  k[3] = 28;  
  k[4] = 38;  k[5] = 47;  
  k[6] = 56;  k[7] = 60;  
  k[8] = 64;  k[9] = 69;  
  k[10] = 70; k[11] = 78;  
  k[12] = 80; k[13] = 82;  
  k[14] = 84; k[15] = 87;  
  k[16] = 90; k[17] = 92;  
  k[18] = 98; k[19] = 108;  
  // Ввод записей  
  for (i = 0; i < 20; i++)   
  {  
    printf("%2d. k[%2d]=%3d: r[%2d]= ", i, i, k[i], i);  
    scanf("%d", &r[i]);  
  }  
  printf("Введите key: "); // вводим искомое ключевое поле  
  scanf("%d", &key);  
  int left = 0; // задаем левую и правую границы поиска  
  int right = 19;  
  int search = -1; // найденный индекс элемента равен -1 (элемент не найден)  
  while (left <= right) // пока левая граница не "перескочит" правую  
  {  
    int mid = (left + right) / 2; // ищем середину отрезка  
    if (key == k[mid]) {  // если ключевое поле равно искомому  
      search = mid;     // мы нашли требуемый элемент,  
      break;            // выходим из цикла  
    }  
    if (key < k[mid])     // если искомое ключевое поле меньше найденной середины  
      right = mid - 1;  // смещаем правую границу, продолжим поиск в левой части  
    else                  // иначе  
      left = mid + 1;   // смещаем левую границу, продолжим поиск в правой части  
  }  
  if (search == -1)     // если индекс элемента по-прежнему -1, элемент не найден  
    printf("Элемент не найден!\n");  
  else          // иначе выводим элемент, его ключ и значение  
    printf("%d. key= %d. r[%d]=%d", search, k[search], search, r[search]);  
  getchar(); getchar();  
  return 0;  
}

Результат выполнения  


**Контрольные вопросы для всех**

* 1. Что такое массивы?
  2. Зачем нужны массивы?
  3. Что такое элемент массива?
  4. Что такое индекс массива?
  5. Как можно обратиться к ячейке массива?
  6. Какого типа могут быть элементы массива?

Вопросы 1-6 На оценку 3

* 1. Какого типа может быть индекс массива?
  2. Как можно описать массив из десяти элементов целого типа?
  3. Как можно осуществить ввод и вывод элементов массива с клавиатуры? Вопросы 1-9 на оценку 4
  4. Как можно осуществить ввод массива через датчик случайных чисел?
  5. Что такое «метод дихотомии»? Для чего он используется?
  6. Как оценить эффективность того или иного метода сортировки?
  7. Как добавляются новые данные в отсортированный массив?

Вопросы 1-13 на оценку 5