**Практическая работа № 15**

**Тема: Реализация рекурсивных вариантов линейного поиска в массиве.**

**Цель: Написание и редактирование программ с использованием пузырьковой сортировки.**

**Краткие теоретические сведения**

**Как работает алгоритм пузырьковой сортировки**

Принцип работы пузырьковой сортировки можно описать в три пункта:

1. Прохождение по всему массиву;
2. Сравнивание между собой пар *соседних* ячеек;
3. Если при сравнении оказывается, что значение ячейки i больше, чем значение ячейки i + 1, то мы меняем значения этих ячеек местами;

Ниже вы можете увидеть, как работает пузырьковая сортировка в действии.

**Как создать пузырьковую сортировку**

Вот что нам придется делать для создания пузырьковой сортировки:

* Создать два цикла for, чтобы проходить по всем элементам массива N раз (N это размер массива).
* Сравнивать ячейки массива, с помощью [оператора ветвления](http://codelessons.ru/cplusplus/lessons/operator-if-else-in-cpp.html) if.
* Менять местами значения ячеек.

**Базовые задачи №1,№2(оценка 3)**

**Задача № 1**

Заполнить массив и отсортировать его, используя пузырьковую сортировку

|  |  |
| --- | --- |
| 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 | #include <iostream>    using namespace std;    int main() {    setlocale(LC\_ALL, "rus");      int digitals[10]; // объявили массив на 10 ячеек      cout << "Введите 10 чисел для заполнения массива: " << endl;      for (int i = 0; i < 10; i++) {      cin >> digitals[i]; // "читаем" элементы в массив    }    **for (int i = 0; i < 10; i++) {**  **for (int j = 0; j < 9; j++) {**  **if (digitals[j] > digitals[j + 1]) {**  **int b = digitals[j]; // создали дополнительную переменную**  **digitals[j] = digitals[j + 1]; // меняем местами**  **digitals[j + 1] = b; // значения элементов**  **}**  **}**  **}**      cout << "Массив в отсортированном виде: ";      for (int i = 0; i < 10; i++) {      cout << digitals[i] << " "; // выводим элементы массива    }    system("pause");    return 0;  } |

**Задача № 2 (упрощенный вариант пузырьковой сортировки)**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | for (int i = 0; i < 10; i++)  {      bool flag = true;      for (int j = 0; j < 10 - (i + 1); j++) {         if (digitals[j] > digitals[j + 1]) {          flag = false;          swap (digitals[j], digitals[j + 1]);        }      }      if (flag) {м        break;      }    } |

**Задания для самостоятельной работы (№3,№4)(оценка 4)**

**Задание №3**

1. Заполните массив из 15 элементов .
2. Отсортируйте массив, используя алгоритм пузырьковой сортировки.
3. Выведите весь массив на экран.

**Задание №4**

Дан массив целых чисел. Количество запросить с клавиатуры. Найти:

сумму элементов массива, больших данного числа А (А вводить с клавиатуры).

**Сортировка элементов массива**

При работе с массивами данных не редко возникает задача их **сортировки по возрастанию или убыванию, т.е. упорядочивания**. Это значит, что элементы нужно расположить строго по порядку.

·         по возрастанию — каждый следующий элемент больше предыдущего — А[1] < А[2] <... < A[N];

·         по не убыванию — каждый следующий элемент не меньше предыдущего, то есть больше или равен А[1 ] ≤ А[2] ≤... ≤A[N];

·         по убыванию — каждый следующий элемент меньше предыдущего А[1 ] > А[2] > ... > A[N];

·   по не возрастанию — каждый следующий элемент не больше предыдущего, то есть меньше или равен А[1] ≥ А[2] ≥ ... ≥ A[N].

Алгоритмы сортировки отличаются друг от друга степенью эффективности, под которой понимается количество сравнений и количество обменов, произведенных в процессе сортировки.

Достаточно простой для понимания является сортировка **методом пузырька**, который также называют **методом простого обмена**.

Как известно воздух легче воды, поэтому пузырьки воздуха всплывают. В сортировке методом пузырька по возрастанию более легкие (с меньшим значением) элементы постепенно "всплывают" в начало массива, а более тяжелые друг за другом опускаются на дно (в конец массива).

**Метод пузырька (простой обмен):**

-При первом проходе по массиву элементы попарно сравниваются между собой: первый со вторым, затем второй с третьим, следом третий с четвертым и т.д. Если предшествующий элемент оказывается больше последующего, то их меняют местами, постепенно самое большое число оказывается последним. Остальная часть массива остается не отсортированной, хотя некоторое перемещение элементов с меньшим значением в начало массива наблюдается.

-При втором проходе незачем сравнивать последний элемент с предпоследним. Последний элемент уже стоит на своем месте. Значит, число сравнений будет на одно меньше.

-На третьем проходе уже не надо сравнивать предпоследний и третий элемент с конца. Поэтому число сравнений будет на два меньше, чем при первом проходе.

-В конце концов, при проходе по массиву, когда остаются только два элемента, которые надо сравнить, выполняется только одно сравнение.

* к меньшему.

**Задания для самостоятельного решения (Оценка 4)**

6.    Составить программу для сортировки простым выбором:

Каким образом выполняется сортировка простым выбором?  
7. Определить есть ли в массиве равные элементы. Вывести их на экран.

8. Дан массив целых чисел. Количество запросить с клавиатуры. Найти:  
максимальный элемент массива и его номер, при условии, что все элементы различные;  
 9. Дан массив из 10 элементов. Первые 4 упорядочить по возрастанию, последние 4 по убыванию.

РЕКУРСИЯ

**Что такое рекурсия в С++**

**Рекурсия в С++ иногда называемая *циклическим определением*, представляет собой процесс определения чего-либо на собственной основе. В программирование это процесс вызова функцией самой себя. Функция которая вызывает саму себя называют *рекурсивной*.**

Рассмотрим процесс рекурсии при вычислении факториала числа 3. Он равен 6 = 1х2х3

Сначала **нерекурсивная** версия программы:

**int fact (int n);**

**int main ( )**

**{**

**cout<<"fact 5 = " << fact(5);**

**return 0;**

**}**

**int fact (int x){**

**int answer=1;**

**for (int i=1; i<=x; i++) answer=answer\*i;**

**return answer;**

}

Тут используется цикл, в котором перемножаются последовательно числа от 1 и заканчивая числом, заданным в качестве параметра.

**Нерекурсивная** версия выполняется быстрее, чем рекурсивная и занимает меньше памяти.

А теперь тоже самое но **с рекурсией:**

**int factr (int n);**

**int main ( )**

**{**

**cout<<"fact 5 with recursion = " << factr(5);**

**return 0;**

**}**

**int factr (int n){**

**int answer;**

**if (n==1) return 1;**

**answer = factr (n-1)\*n;**

**cout << answer << '\n'; // 2 6 24 120 fact 4 with recursion = 120**

**return answer;**

**}**

**В C++** [**любая функция**](https://purecodecpp.com/archives/1078) **кроме main() может вызывать саму себя. То есть в теле функции может быть размещен вызов этой же функции**. Это называется рекурсией.

Когда программа выполняет код рекурсивной функции – она будет выполнять его бесконечно, если не предусмотреть условие выхода из рекурсии. Об этом надо помнить, чтобы избежать зацикливания вызовов. **Поэтому в определении рекурсивной функции обязательно надо указывать условие выхода. Например, можно разместить её вызов**

**внутри блока if**

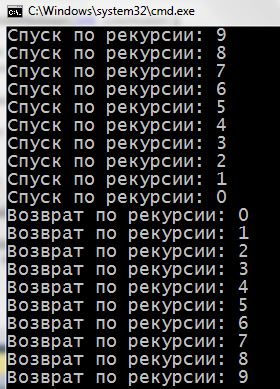
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **#include <iostream>**  **using namespace std;**    **void recursionGoDown(int someNumber)**  **{**  **cout << "Спуск по рекурсии: " << someNumber << endl;**  **if (someNumber > 0)**  **{**  **recursionGoDown(someNumber - 1); // рекурсивный вызов**  **}**  **cout << "Возврат по рекурсии: " << someNumber << endl;**  **}**    **int main()**  **{**  **setlocale(LC\_ALL, "rus");**    **recursionGoDown(9);**  return 0;  } |

В этой программе функция **recursionGoDown()** будет вызывать саму себя, пока в нее передается число больше нуля. Каждый раз, функция получает число, от которого в сигнатуре отнимается единица. Как только условие станет ложным (число станет меньше нуля) – вызовы остановятся и начнется выход из рекурсии.

Если мы передаем в такую функцию число 9, вызовов будет 10. Программа как бы углубляется на 10 уровней, выполняя рекурсию. Чтобы ей выйти из рекурсии, надо пройти эти десять уровней обратно. Так строка кода

|  |  |
| --- | --- |
| 1 | cout << "Возврат по рекурсии: " << someNumber << endl; |

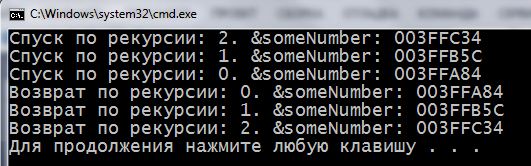
будет выполняться тоже 10 раз, но значение **someNumber** будет меняться в обратном порядке. Сначала оно будет соответствовать тому, что передалось в функцию на 10 уровне рекурсии, потом тому что передалось на 9 уровне и т.д.

[](https://purecodecpp.com/wp-content/uploads/2014/10/rekursiya-c-2.jpg)

Хочу обратить внимание на важный момент. При каждом рекурсивном вызове создается новая переменная **someNumber** с новым значением. Эти переменные будут занимать какое-то место в оперативной памяти. Если вызовов 10, как в нашем примере, то и в памяти будет храниться 10 переменных с разными значениями. Можно дописать в программу вывод на экран адресов, по которым хранятся эти значения и убедиться в этом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <iostream>  using namespace std;    void recursionGoDown(int someNumber)  {  cout << "Спуск по рекурсии: " << someNumber << ". &someNumber: "<< &someNumber << endl;  if (someNumber > 0)  {  recursionGoDown(someNumber - 1); // рекурсивный вызов  }  cout << "Возврат по рекурсии: " << someNumber << ". &someNumber: " << &someNumber << endl;  }    int main()  {  setlocale(LC\_ALL, "rus");    recursionGoDown(2);  return 0;  } |

Передаем в функцию число 2: recursionGoDown(2); Уровней рекурсии будет 3.

[](https://purecodecpp.com/wp-content/uploads/2014/10/rekursiya-c-3.jpg)

Как видно, каждое значение переменной **someNumber** хранится в отдельном отрезке памяти. Во время обратного выхода по рекурсии значения берутся из тех же адресов но в обратном порядке.

Практически каждую поставленную задачу, которую можно решить используя рекурсию. Вы ведь легко можете показать на экран числа от 9 до 0 и от 0 до 9 применив циклы. Помимо этого следует знать, что рекурсия займет больше памяти и будет производить вычисления медленнее, чем итерация.

**Рекурсивное вычисление факториала**

**Исходный код на языке C++**

**#include <iostream>**

**using namespace std;**

**long double fact(int N)**

**{**

**if(N < 0) // если пользователь ввел отрицательное число**

**return 0; // возвращаем ноль**

**if (N == 0) // если пользователь ввел ноль,**

**return 1; // возвращаем факториал от нуля - не удивляетесь, но это 1 =)**

**else // Во всех остальных случаях**

**return N \* fact(N - 1); // делаем рекурсию.**

**}**

**int main()**

**{**

**int N;**

**setlocale(0,""); // Включаем кириллицу**

**cout << "Введите число для вычисления факториала: ";**

**cin >> N;**

**cout << "Факториал для числа " << N << " = " << fact(N) << endl << endl; // fact(N) - функция для вычисления факториала.**

**return 0;**

**}**

Задача № 5 **(оценка 5)**

**А)Вычислить факториал числа 5 и 10 и**

**Б) Вычислить это же без рекурсии**