Федеральное государственное бюджетное образовательное учреждение
высшего образования «Кемеровский государственный университет»
Новокузнецкий институт (филиал)

Факультет информатики, математики и экономики
Кафедра информатики и вычислительной техники им. В. К. Буторина

О. А. Штейнбрехер, О.И. Новоселова

Программирование. Часть 2

Методические указания по выполнению практических работ по дисциплине «Программирование» для обучающихся по направлению подготовки 09.03.03 Прикладная информатика Профиль «Прикладная информатика в экономике»

Новокузнецк
2020

УДК [378.147.88:004.4](072)
ББК 74.484(2Рос-4Кем)я73+ 32.972я7
Ш88

Ш88 «Программирование. Часть 2. Методические указания по выполнению практических работ по дисциплине: метод. указ (текст. электрон. изд.)/ О.А. Штейнбрехер, О.И. Новоселова ; Новокузнец. ин-т (фил.) Кемеров. гос. ун-та – Новокузнецк: НФИ КемГУ, 2020. – 31 с.

Приводятся методические указания по выполнению практических работ по дисциплине «Программирование» для второго семестра обучения.

Методические указания предназначены для студентов всех форм обучения направления 09.03.03 «Прикладная информатика».

Рекомендовано
на заседании кафедры
информатики и вычислительной
техники им. В. К. Буторина
23 ноября 2020 года.
Заведующий кафедрой

А. В. Маркидонов

Утверждено
методической комиссией факультета
информатики, математики и экономики
17 декабря 2020 года.
Председатель методкомиссии

Г.Н. Бойченко

УДК [378.147.88:004.4](072)
ББК 74.484(2Рос-4Кем)я73+ 32.972я7

Текст представлен в авторской редакции

Оглавление

ПРЕДИСЛОВИЕ

Дисциплина «Программирование» относится к основному разделу образовательной программы. Дисциплина изучается в течение трех семестров.

В результате освоения данной дисциплины у обучающегося должны быть сформированы компетенции основной профессиональной образовательной программы бакалавриата (далее - ОПОП):

ОПК-7 – Способен разрабатывать алгоритмы и программы, пригодные для практического применения.

В результате обучения обучающийся должен:

знать:

- классификацию программных средств, языков программирования;
- основные парадигмы программирования;
- понятия и методы алгоритмизации;
- основы и методы структурного программирования;
- основные понятия объектно-ориентированного программирования;
- основы теории алгоритмов и основы теории сложности;

уметь:

- разрабатывать алгоритмы для решения прикладных практических задач;
- разрабатывать программы для реализации прикладных практических задач;
- обосновывать выбор стандартных алгоритмов для решения практических задач;
- осуществлять выбор стандартных средств для программной реализации алгоритмов и программ;

владеть:

- методами алгоритмизации, оценки сложности алгоритмов;
- графическим способом описания алгоритмов;
- методами структурного программирования;
- навыками реализации алгоритмов и программ, с учетом сложности алгоритмов

Первый семестр посвящен вопросам алгоритмизации, теории алгоритмов, моделям вычисления и основам программирования. Для изучения обучающимся предполагается изучение C++.

Второй семестр посвящен изучению нескалярных типов данных языков программирования. Кроме этого, во втором семестре предполагается изучение основ языка C#. Выбор его вторым языком обусловлен родственым синтаксисом.

В таблице 1 представлено распределение баллов за семестр.

Таблица 1 - Балльно-рейтинговая оценка результатов учебной работы обучающихся по видам(БРС) 2 семестр

Учебная работа (виды)	Сумма баллов	Виды и результаты учебной работы	Оценка в аттестации	Баллы (17 недель)
Текущая учебная работа в семестре (Посещение занятий по расписанию и выполнение заданий)	**60**	Лекционные занятия (конспект) (9 занятий)	**0,5 балла** посещение 1 лекционного занятия	4
		Практические работы (отчет о выполнении) (9 работ).	**2 балл** - посещение 1 практического занятия и выполнение работы на 51-65% **4 балла** – посещение 1 занятия и существенный вклад на занятии в работу всей группы, самостоятельность и выполнение работы на 85,1-100%	18 - 36
		Индивидуальные задания (отчет о выполнении) (4 работы)	**За одну КР от 2 до:** **3 баллов** (выполнено 51 - 65% заданий) **4 балла** (выполнено 66 - 85% заданий) **5 баллов** (выполнено 86 - 100% заданий)	8 - 20
Итого по текущей работе в семестре				31 - 60
Итого по промежуточной аттестации (экзамену)				20 – 40 б.
Суммарная оценка по дисциплине: Сумма баллов текущей и промежуточной аттестации				51 – 100 б.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Скалярные и нескалярные типы данные

По аналогии с математикой, типы данных делят на скалярные (примитивные) и нескалярные (агрегатные). Значение нескалярного типа (нескалярное значение) имеет множество видимых пользователю компонентов, а значение скалярного типа (скалярное значение) не имеет такового. Примерами нескалярного типа являются массивы, списки и т. д.; примеры скалярного типа — «целое», «логическое» и т. д.

Структурные (агрегатные) типы не следует отождествлять со структурами данных: одни структуры данных непосредственно воплощаются определёнными структурными типами, но другие строятся посредством их композиции, чаще всего рекурсивной.

По другой классификации типы делятся на самостоятельные и зависимые. Важными разновидностями последних являются ссылочные типы, частным случаем которых, в свою очередь, являются указатели.

Указатели

При выполнении инициализации переменной, ей автоматически присваивается свободный адрес памяти, и, любое значение, которое присваивается переменной, сохраняется по этому адресу в памяти.

Указатель – переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. Указатель может ссылаться на переменную или функцию. Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению, то есть, используя имя переменной. В основном указатели используются для организации динамического распределения памяти.

Объявление указателя аналогично объявлению переменной, но перед именем указателя ставится символ *. При объявлении указателей компилятор выделяет несколько байт памяти, в зависимости от типа данных отводимых

для хранения некоторой информации в памяти. Чтобы получить значение, записанное в некоторой области, на которое ссылается указатель нужно воспользоваться операцией разыменования указателя *. Унарная операция & позволяет получить адрес объявленных переменных.

```
int var = 123; // инициализация переменной var числом 123
int *ptrvar = &var;
// указатель на переменную var (присвоили адрес переменной указателю)
```

Указатели можно сравнивать не только на равенство или неравенство, ведь адреса могут быть меньше или больше относительно друг друга. Из арифметических операций, чаще всего используются операции сложения, вычитания, инкремент и декремент, так как с помощью этих операций, например, в массивах, вычисляется адрес следующего элемента. Ниже представлены примеры арифметических операций над указателями.

Над указателями одного типа можно выполнять арифметическую операцию вычитания -. В результате выполнения этой операции получаем целое число, которое указывает, на сколько один адрес памяти смещен от другого в единицах равных длине типа данных, на который указывают указатели.

К указателям можно применять операторы ++ и --. В этом случае значение указателя соответственно увеличится или уменьшится на длину типа данных, на который указывает указатель.

К указателю можно прибавить или вычесть целое число. В этом случае значения указателя соответственно увеличится или уменьшится на это число, умноженное на длину типа данных, на который указывает указатель.

Указатели могут ссылаться на другие указатели (рисунок 1). При этом в ячейках памяти, на которые будут ссылаться первые указатели, будут содержаться не значения, а адреса вторых указателей. Число символов * при объявлении указателя показывает порядок указателя. Чтобы получить доступ к значению, на которое ссылается указатель его необходимо разыменовывать соответствующее количество раз.

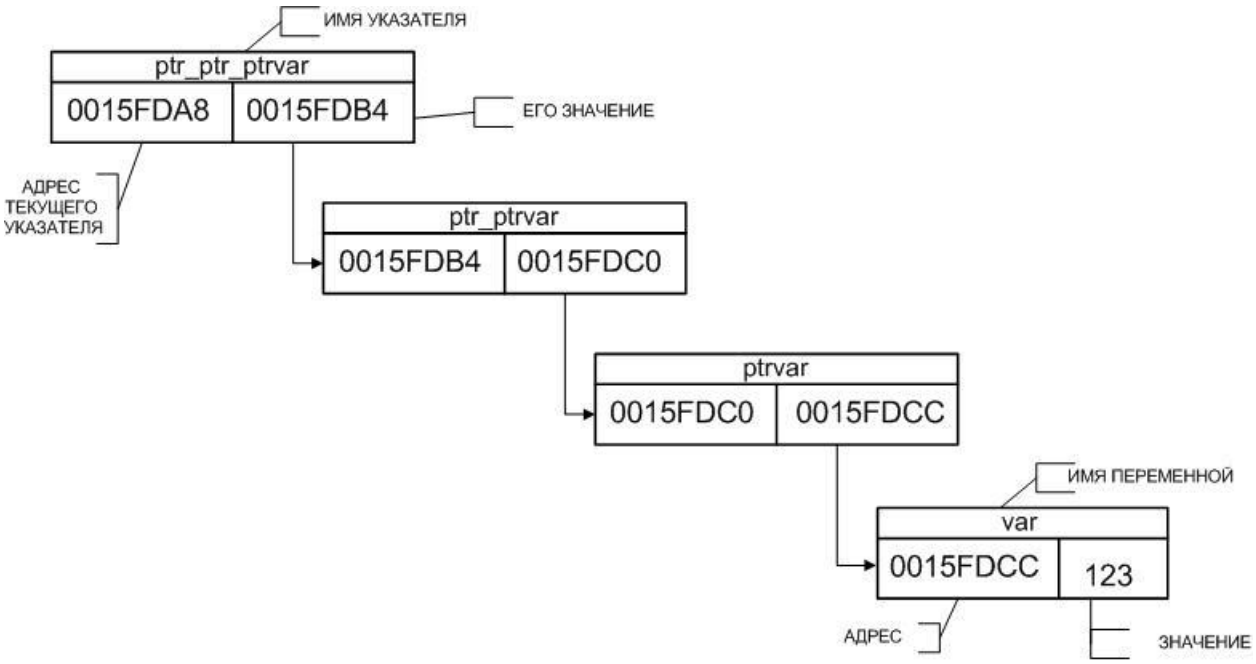

Рисунок 1 – Схема разыменовывания указателя третьего порядка

Массивы

Массив - это совокупность переменных одного типа, к которым обращаются с помощью общего имени. Доступ к отдельному элементу массива может осуществляться с помощью индекса. Количество элементов в массиве называется размерностью массива. Массивы, размерность которых задается при инициализации и остается неизменной, называют статическими.

Массивы, размеры которых могут изменяться во время выполнения программы называются динамическими. Возможность изменения размера отличает динамический массив от статического, размер которого задаётся на момент компиляции программы. Для изменения размера динамического массива язык программирования, поддерживающий такие массивы, должен предоставлять встроенную функцию или оператор.

К динамическим массивам также могут относить массивы переменной длины, размер которых не фиксируется при компиляции, а задаётся при создании или инициализации массива во время исполнения программы. Размер такого массива не может быть изменен после его инициализации.

Динамические массивы могут поддерживаться либо на уровне синтаксиса самого языка программирования, либо на уровне системных

библиотек. Описание динамического массива может синтаксически отличаться от описания статического, но может и быть таким же. Во втором случае, как правило, все массивы в языке являются потенциально динамическими, и только от программиста зависит, использовать ли это свойство в каждом конкретном случае.

Язык С не содержит динамических массивов, но функции стандартной библиотеки malloc, free и realloc позволяют реализовать массив переменного размера. В C++ поддерживаются функции работы с памятью из стандартной библиотеки Си, но их использование не рекомендуется. Массив переменной длины здесь также можно выделить с помощью стандартных команд работы с динамической памятью new и delete. Аналога realloc нет, поэтому изменить размер массива можно только вручную, выделив новую память нужного размера и перенеся в неё данные.

Рассмотрим объявление и инициализацию статических массивов. В языках С все массивы состоят из соприкасающихся участков памяти. Наименьший адрес соответствует первому элементу. Наибольший адрес соответствует последнему элементу. Массивы могут иметь одну или несколько размерностей.

Имя массива является константным указателем на его первый элемент, то есть указателем, значение которого нельзя изменить. При определении массива его можно проинициализировать. Кроме того, при инициализации массива одновременно с его объявлением, размерность массива может не задаваться в явном виде.

```
int           a[10];
int           a[3] = {0, 1, 2};
int           a[] = {0, 1, 2, -5};
```

Доступ к элементам массива выполняется при помощи оператора индексирования [ ], результатом выполнения которого является значение элемента массива с заданным индексом. На рисунке 2 представлена иллюстрация выделения ячеек памяти под одномерный статический массив.

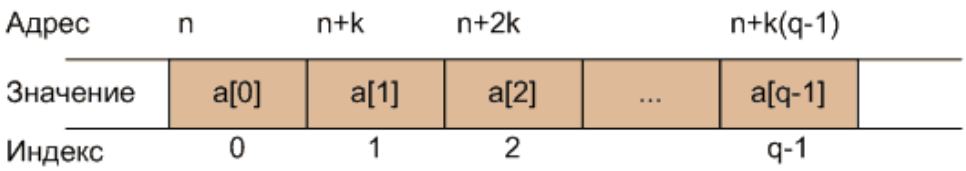

Рисунок 2 – Адресация памяти одномерного статического массива

```
int           n;
int           a[3] = {10, 20, 30};
n = a[0];
// n = 10
a[1] = 40;
```

Объявление и инициализация многомерного массива осуществляется следующим образом: <тип данных> <имя массива> [<количество строк>][<количество столбцов>].

```
int           a[2] [2] = {{1, 2}, {3, 4}};
int           n = a[0] [0];                          // n = 1
```

На рисунке 3 представлено выделение памяти под многомерный статический массив.

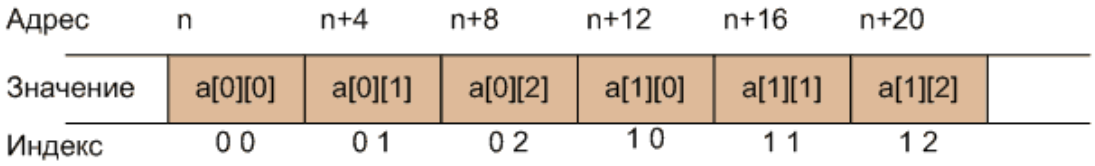

Рисунок 3 – Адресация памяти двумерного статического массива

Для создания массивов переменной длины используется динамическое выделение памяти. Под динамической памятью понимают память, которая выделяется программе во время её работы. Динамическое распределение памяти включает выделение программе памяти по её запросу и последующее освобождение программой этой памяти.

Операции new и delete предназначены для динамического распределения памяти компьютера. Операция new выделяет память из области свободной памяти, а операция delete высвобождает выделенную память. Выделяемая память, после её использования должна высвобождаться, поэтому операции new и delete используются парами. Даже

если не высвобождать память явно, то она освободится ресурсами ОС по завершению работы программы.

Операция new создает объект заданного типа, выделяет ему память и возвращает указатель правильного типа на данный участок памяти. Если память невозможно выделить, например, в случае отсутствия свободных участков, то возвращается нулевой указатель, то есть указатель вернет значение 0. Выделение памяти возможно под любой тип данных: int, float,double, char и т. д.

Для того чтобы динамически создать одномерный массив нужно, во-первых, объявить в программе указатель на тип, соответствующий типу элементов этого массива, а, во-вторых, выделить память под массив.

```
// Создание массива длиной n
  int *mas = new int[n];
  ...
  // Освобождение памяти массива
  delete []mas;
```

Память под многомерные динамические массивы выделяется по строкам, начиная с первого индекса. Это делается для того, чтобы обеспечить применение операции индексирования [ ] столько раз какова размерность многомерного массива. В этом случае тип динамического массива объявляется как указатель, который содержит оператор обращения по адресу ‘*’ столько раз, какова размерность массива.

Реализация динамического массива с изменяемыми размерами, возможна через шаблонный класс std::vector. std::vector имеет множество методов и переопределённых операторов. Они позволяют обращаться по индексу, изменять в любую сторону размер массива, использовать его как стек. Управляемым является не только актуальный размер, но и ёмкость вектора, что позволяет оптимизировать процесс выделения памяти. Стандарт C++ требует от реализации обязательного выполнения условий:

- все элементы вектора должны храниться подряд в порядке увеличения индекса в целостном блоке оперативной памяти;
- должно быть гарантировано константное время доступа к элементу вектора.

```
std::vector<int> mas; // Создать пустой вектор
mas.reserve(100);
// Выделить вектору память под 100 элементов (размер останется нулевым)
mas.resize(50);       // Задать размер вектора в 50 элементов
mas[i] = i;           // Обращение к элементу по индексу
mas.push_back(100);   // Добавить элемент
x = mas.back;         // Обращение к последнему элементу
mas.pop_back();       // Удалить последний элемент
std::cout << mas.size() << " " << mas.capacity() << "\n";
// Вывести ёмкость и размер
mas.shrink_to_fit();  // Уменьшить ёмкость до размера
```

Строки

Строка - произвольная последовательность символов. Следует различать символ, символьный литерал, строковый литерал и строку как тип данных.

Символ — это встроенный интегральный тип в C/C++, для него допустимы все операции, допустимые для интегральных типов.

Символьные литералы (записываются в одинарных кавычках (прямых апострофах)). Специальный формат для записи символьных литералов – слеш, за которым идет код символа. Такая форма записи необходима, если мы хотим использовать элемент, не отображающийся в печатный символ, например, нуль-терминатор, который представляется так: ’\0’.

Строковый литерал - последовательность символов, заключенная в двойные кавычки.

Таким образом:

- ‘\0’ – символьный литерал, соответствующий символу с кодом 0;
- ‘0’ – символьный литерал, обозначающий цифру 0 (ASCII - 48);
- “0” – строковый литерал (содержит два символа – 0 и нуль-терминатор).

Возможно следующее представление строк (как типа данных) в языках программирования:

- массив символов;
- нуль-терминированная строка (использование завершающего байта);
- в виде списка (Пролог).

Возможны следующие операции со строками:

- получение символа по номеру позиции (индексу) — в большинстве языков это тривиальная операция;
- конкатенация (соединение) строк;
- получение подстроки по индексам начала и конца;
- проверка вхождения одной строки в другую (поиск подстроки в строке);
- проверка на совпадение строк (с учётом или без учёта регистра символов);
- получение длины строки;
- замена подстроки в строке;
- нахождение минимальной надстроки, содержащей все указанные строки;
- поиск в двух массивах строк совпадающих последовательностей (задача о плагиате).

В языке С поддерживается реализация строк в виде последовательность символов – массив символов с ограничивающим символом (терминатором). В качестве терминатора (ограничивающего символа) выбран символ с кодом 0, как наиболее редко встречающийся в строках – нуль-терминатор. Такие строки называют Си-строками.

Си- строки имеют следующие достоинства.

- Натуральное представление строки (начало строки совпадает с началом последовательности символов)

- Неограниченная длина строки - в случае хранения длины строки отдельно длина строки не может превышать значения, максимального для переменной этого типа.
- Возможен сдвиг начала строки вправо.

При этом в качестве недостатков можно указать следующее.

- Для определения длины строки всякий раз необходимо искать ее конец, сравнивая ее символы с нулевым. Чем длиннее строка - тем больше операций.
- Строки, с которыми используются стандартные функции, не могут содержать в себе нулевые символы.

```
char str1[10];
char str2[10]="Hello";
char str3[10]={'H', 'e', 'l', 'l', 'o', '\0'};
```

При выполнении операций присваивания и действий со строками следует помнить, что Си-строки аналогичны массиву. Поэтому выполнение такого вида операции невозможно: str1 = “Hello”. А выполнение присваивания str1=str2 приведет к работе с указателями.

Для лексикографического сравнения строк используются функции strcmp и stricmp. Первая сравнивает строки с учетом регистра, вторая – без (для латинского алфавита).

Операция конкатенации (сложения строк) осуществляется с помощью операторов strcat(), sprintf().

C++ поддерживает работу с Си-строками и класс string, для использования которого необходимо подключить заголовочный файл string. Рассмотрим подробнее.

Строки можно объявлять и одновременно присваивать им значения. К отдельным символам строки можно обращаться по индексу, как к элементам массива или С-строк. Например, S[0] - это первый символ строки.

```
string S1, S2 = "Hello";
```

Для того, чтобы узнать длину строки можно использовать метод size() строки. Например, последний символ строки S это S[S.size() - 1].

Со строками можно выполнять следующие арифметические операции:

- = - присваивание значения.
- += - добавление в конец строки другой строки или символа.
- + - конкатенация двух строк, конкатенация строки и символа.
- ==, != - посимвольное сравнение.
- <, >, <=, >= - лексикографическое сравнение.

То есть можно скопировать содержимое одной строки в другую при помощи операции S1 = S2, сравнить две строки на равенство при помощи S1 == S2, сравнить строки в лексикографическом порядке при помощи S1 < S2, или сделать сложение (конкатенацию) двух строк в виде S = S1 + S2.

Метод size() возвращает длину строки. Возвращаемое значение является беззнаковым типом (как и во всех случаях, когда функция возвращает значение, равное длине строке или индексу элемента - эти значения беззнаковые). Кроме того, у строк есть метод length(), который также возвращает длину строки.

S.resize(n) - Изменяет длину строки, новая длина строки становится равна n. При этом строка может как уменьшится, так и увеличиться. Если вызвать в виде S.resize(n, c), где c - символ, то при увеличении длины строки добавляемые символы будут равны c.

S.clear() - очищает строчку, строка становится пустой. S.empty() - возвращает true, если строка пуста, false - если непуста.

Файлы

Для работы с файлами необходимо подключить заголовочный файл <fstream>. В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream> — файловый ввод и <ofstream> — файловый вывод.

Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на экран, а в файл. Если ввод/вывод на стандартные устройства выполняется с помощью объектов cin и cout, то для организации файлового ввода/вывода достаточно

создать собственные объекты, которые можно использовать аналогично операторам cin и cout.

Для записи данных в файл требуется осуществить следующие действия:

- создать объект класса ofstream;
- связать объект класса с файлом, в который будет производиться запись (Указанный файл будет создан в текущей директории с программой. Если файл с таким именем существует, то существующий файл будет заменен новым.);
- записать строку в файл;
- закрыть файл.

```
ofstream fout;
fout.open("cppstudio.txt"); // связываем объект с файлом
fout << "Работа с файлами в C++"; // запись строки в файл
fout.close(); // закрываем файл
```

Для того чтобы прочитать файл понадобится выполнить те же шаги, что и при записи в файл с небольшими изменениями:

- создать объект класса ifstream и связать его с файлом, из которого будет производиться считывание;
- прочитать файл;
- закрыть файл.

Существует несколько режимов открытия файла:

- ios_base::in открыть файл для чтения
- ios_base::out открыть файл для записи
- ios_base::ate при открытии переместить указатель в конец файла
- ios_base::app открыть файл для записи в конец файла
- ios_base::trunc удалить содержимое файла, если он существует
- ios_base::binary открытие файла в двоичном режиме

```
ofstream fout("cppstudio.txt", ios_base::app); // открываем файл для добавления информации к концу файла
```

```
fout.open("cppstudio.txt", ios_base::app); // открываем файл для добавления информации к концу файла
```

Режимы открытия файла можно комбинировать с использованием логического ИЛИ.

В случае, если файл не будет найден, компилятор проигнорирует строки, где выполняется работа с файлом. В результате корректно завершится работа программы.

Проверка открытия файла может осуществить с использованием функции — is_open(), которая возвращает целые значения: 1 — если файл был успешно открыт, 0 — если файл открыт не был.

Следует заметить, что считывание файла возможно несколькими способами. Так же как и в iostream считывание можно организовать оператором >>, который указывает в какую переменную будет произведено считывание. Считает вещественное, целое и строку. Считывание строки закончится, если появится пробел или конец строки. Стоит отметить, что оператор >> применяется к текстовым файлам. Считывание из бинарного файла производить лучше всего с помощью метода read().

Считывание целой строки до перевода каретки производится так же как и в iostream методом getline(). Если же читать нужно в массив символов char[], то либо get() либо getline() именно как методы.

Метод close() – закрывает файл.

Метод eof() – проверяет достигнут ли конец файла.

Метод seekg() – производит установку текущей позиции в нужную, указываемую числом.

Язык C# (особенности)

Платформа .NET Framework — это технология, которая поддерживает создание и выполнение веб-служб и приложений Windows.

При разработке платформы .NET Framework учитывались следующие цели.

- Обеспечение согласованной объектно-ориентированной среды программирования для локального сохранения и выполнения объектного кода, для локального выполнения кода, распределенного в Интернете, либо для удаленного выполнения.
- Обеспечение среды выполнения кода, минимизирующей конфликты при развертывании программного обеспечения и управлении версиями.
- Обеспечение среды выполнения кода, гарантирующей безопасное выполнение кода, включая код, созданный неизвестным или не полностью доверенным сторонним изготовителем.
- Обеспечение среды выполнения кода, исключающей проблемы с производительностью сред выполнения сценариев или интерпретируемого кода.
- Обеспечение единых принципов разработки для разных типов приложений, таких как приложения Windows и веб-приложения.

Взаимодействие на основе промышленных стандартов, которое гарантирует интеграцию кода платформы .NET Framework с любым другим кодом.

C# — современный объектно-ориентированный и типобезопасный язык программирования. C# относится к широко известному семейству языков C.

C# является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование.

В C# предусмотрена сборка мусора, обработка исключений и типобезопасная структура. Сборка мусора автоматически освобождает память, занятую недостижимыми неиспользуемыми объектами. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и их восстановлению. Типобезопасная структура языка делает невозможным чтение из неинициализированных переменных,

индексацию массивов за пределами их границ или выполнение непроверенных приведений типов.

В C# существует единая система типов. Все типы C#, включая типы-примитивы, такие как int и double, наследуют от одного корневого типа object. Таким образом, все типы используют общий набор операций, и значения любого типа можно хранить, передавать и обрабатывать схожим образом. Кроме того, C# поддерживает пользовательские ссылочные типы и типы значений, позволяя как динамически выделять память для объектов, так и хранить упрощенные структуры в стеке.

В C# основными понятиями организационной структуры являются программы, пространства имен, типы, члены и сборки. Программа на языке C# состоит из одного или нескольких файлов.

В программе объявляются типы, которые содержат члены. Эти типы можно организовать в пространства имен. Примерами типов являются классы и интерфейсы.

К членам относятся поля, методы, свойства и события.

При компиляции программы на C# упаковываются в сборки. Сборка — это файл, обычно с расширением .exe или .dll, если она реализует приложение или библиотеку, соответственно. Сборки содержат исполняемый код в виде инструкций промежуточного языка (IL) и символьную информацию в виде метаданных. Перед выполнением JIT-компилятор среды CLR .NET преобразует код IL в сборке в код, зависящий от процессора.

Сборка полностью описывает сама себя и содержит весь код и метаданные, поэтому в C# не используются директивы #include и файлы заголовков. Чтобы использовать в программе C# открытые типы и члены, содержащиеся в определенной сборке, вам достаточно указать ссылку на эту сборку при компиляции программы.

Пространства имен позволяют иерархически упорядочивать программы и библиотеки C#. Пространства имен содержат типы и другие пространства

имен. Например, пространство имен System содержит несколько типов (в том числе класс Console) и несколько других пространств имен, таких как IO и Collections. Директива using, которая ссылается на пространство имен, позволяет использовать типы из этого пространства имен без указания полного имени. Благодаря директиве using в коде программы можно использовать сокращенное имя Console.WriteLine вместо полного варианта System.Console.WriteLine.

Типы данных

Типы значений в C# подразделяются на простые типы, типы перечисления, типы структур и типы, допускающие значение Null. Ссылочные типы в C# подразделяются на типы классов, типы интерфейсов, типы массивов и типы делегатов.

- Простые типы
  - Целочисленный со знаком: sbyte, short, int,long
  - Целочисленный без знака: byte, ushort, uint,ulong
  - Символы Юникода: char
  - Бинарный оператор IEEE с плавающей запятой: float, double
  - Десятичное значение с повышенной точностью и плавающей запятой: decimal
  - Логическое значение: bool
- Типы перечисления
  - Пользовательские типы в формате enum E {...}
- Типы структур
  - Пользовательские типы в формате struct S {...}
- Типы значений, допускающие значение NULL
  - Расширения других типов значений, допускающие значение null
- Типы классов
  - Исходный базовым классом для всех типов: object
  - Строки Юникода: string
  - Пользовательские типы в формате class C {...}

- Типы интерфейсов
    - Пользовательские типы в формате interface I {...}
- Типы массивов
    - Одно- и многомерные, например, int[] и int[,]
- Типы делегатов
    - Пользовательские типы в формате delegate int D(...)

Система типов в C# унифицирована таким образом, что значение любого типа можно рассматривать как object (объект). Каждый тип в C# является прямо или косвенно производным от типа класса object, и этот тип object является исходным базовым классом для всех типов. Чтобы значения ссылочного типа обрабатывались как объекты, им просто присваивается тип object. Чтобы значения типов значений обрабатывались как объекты, выполняются операции упаковки-преобразования и распаковки-преобразования.

Унифицированная система типов C# фактически позволяет преобразовывать типы значений в объекты "по требованию". Такая унификация позволяет применять универсальные библиотеки, использующие тип object, как со ссылочными типами, так и с типами значений.

Объявление переменных

В C# используются два метода для обеспечения инициализации переменных перед пользованием.

Переменные, являющиеся полями класса или структуры, если не инициализированы явно, по умолчанию обнуляются в момент создания.

Переменные, локальные по отношению к методу, должны быть явно инициализированы в коде до появления любого оператора, в котором используются их значения. В данном случае при объявлении переменной ее инициализация не происходит автоматически, но компилятор проверит все возможные пути потока управления в методе и сообщит об ошибке, если обнаружит любую возможность использования значения этой локальной переменной до ее инициализации.

Операторы ввода-вывода

Консольный вывод осуществляется через методы Console.Write() и Console.WriteLine().

Console.ReadLine() - позволяет получить введенную строку. Console.ReadLine считывает информацию именно в виде строки. Поэтому можем по умолчанию присвоить ее только переменной типа string.

Используются методы явного преобразования, такие как:

- Convert.ToInt32() (преобразует к типу int);
- Convert.ToDouble() (преобразует к типу double);
- Convert.ToDecimal() (преобразует к типу decimal).

Практическая работа №1. Статические массивы

1. Реализовать решение задачи для целочисленных массивов (с использованием статических одномерных массивов):

Вариант	Задание
1	Ввести массив из 10 элементов. Вывести сумму положительных элементов массива.
2	Ввести массив из 15 элементов. Вывести все отрицательные элементы массива
3	Ввести массив из 10 элементов. Вывести числа в обратном порядке
4	Ввести массив из 15 элементов. Вывести квадраты четных чисел.
5	Ввести массив из 10 элементов. Вывести числа, расположенные на положительных местах массива

2. Реализовать решение задачи (с использование статических многомерных массивов) (массив 4 на 4):

Вариант	Задание
1	Заполнить массив только простыми числами, следуя стандартной нумерации
2	Заполнить массив квадратами сумм индексов
3	Заполнить массив целыми числами, так чтобы в четных строках значения возрастали, а в нечетных убывали. Начальное число ввести с клавиатуры
4	Заполнить массив степенями двойки по столбцам
5	Заполнить массив целыми числами подряд, так чтобы четные числа стояли на позициях, где сумма индексов кратна 3.

3. Реализовать свертку трехмерного массива (5x5x5) в одномерный. Значения ввести с клавиатуры.

4. Реализовать свертку двумерного массива (3 на 4) в одномерный. Значения ввести с клавиатуры.
5. Реализовать заполнение двумерного массива (12 на 12) от центра по спирали целыми числами.

Практическая работа №2. Динамические массивы

1. Реализовать алгоритм решения задачи, если размерность матриц задается пользователем (при решении задачи необходимо учитывать возможности введения с клавиатуры некорректных значений):

Вариант	Задание
1	Умножение двух матриц, размерности которых вводятся пользователем с клавиатуры (количество строк и количество столбцов)
2	Нахождение матрицы для нахождения минора матрицы, размерность матрицы которой и коэффициенты миноров заданы пользователем в клавиатуры
3	Сложение двух матриц, размерности которых вводятся пользователем с клавиатуры (количество строк и количество столбцов)
4	Определение ранга матрицы, размерность которой задана пользователем с клавиатуры
5	Определение главной и побочной диагоналей матрицы, размерность которой задана пользователем с клавиатуры

2. Реализовать алгоритм хранения и восстановления разряженной матрицы (матрицы с большим количество м пустых элементов):

Вариант	Задание
1	Формат RR(C)O. В данном формате вместо одного двумерного массива, используются три одномерных. Значения ненулевых элементов матрицы и соответствующие им столбцовые индексы хранятся в этом формате по строкам в двух массивах AN и JA. Массив указателей IA, используется для ссылки на компоненты массивов AN и JA, с которых начинается описание очередной строки.

	Последняя компонента массива IA содержит указатель первой свободной компоненты в массивах AN и JA, т.е. равна числу ненулевых элементов матрицы, увеличенному на единицу.
2	Формат RR(C)U. Формат RR(C)U отличается от RR(C)O тем, что в данном случае соблюдается упорядоченность строк, но внутри каждой строки элементы исходных матриц могут храниться в произвольном порядке.

3. Реализовать алгоритм хранения треугольной матрицы по диагональной схеме (хранятся последовательно главные диагонали).

4. Реализовать алгоритм вычисления средних оценок списка обучающихся, где количество обучающихся задается пользователем в начале работы, а количество оценок у каждого обучающегося отличается.

Практическая работа №3.Указатели, ссылки и динамические структуры

1. Дан указатель: double **p = 0; Реализуйте заполнение и вывод данных.

2. Напишите функцию, которая возвращает ссылку на максимальный из трех своих аргументов. Используя ее, замените значение максимального из трех чисел их средним значением.

3. Напишите функции, возвращающие ссылки на максимальный и минимальный элементы массива. Используйте их для обмена значениями максимального и минимального элементов.

4. Используя массивы, указатели и структуры, реализуйте динамическую структуру и действия с ней.

Вариант	Задание
1	очередь
2	стек

5. Используя массивы, указатели и структуры, реализуйте динамическую структуру однонаправленного связанного списка и действия с ней:

добавление элемента в начало

добавление элемента в конец

добавление элемента в середину списка

ввод списка

удаление первого элемента

удаление последнего элемента

удаление элемента из середины списка

вычисление длины списка

6. Используя массивы, указатели и структуры, реализуйте вывод и заполнение динамической структуры.

Вариант	Задание
1	бинарное дерево

2	двунаправленный список
3	связанный двунаправленный список

7. Для структуры из задания 6, реализуйте поиск максимального и минимального элемента.

8. Реализуйте сортировку пузырьком для одномерного динамического массива

Практическая работа №4. Строки и символы

По каждой задаче должны быть представлены две программные реализации алгоритма - с использованием Си-строк и класса string.

1. Поиск символа в строке. Строка вводится с клавиатуры, определить, есть ли символ, введенный пользователем в этой строке.
2. Пользователь вводит с клавиатуры строку. Заменить символы согласно правилу. Предусмотреть проверку на наличие символов не входящих в заданный алфавит.

Вариант	Задание		
	Алфавит	Ограничения на введенный текст	Правило замены
1	{0, 1}	Запись в бинарном коде без пробелов	Инвертирование символов
2	строчные символы латинского алфавита, без знаков препинания	Запись символами латинского алфавита без пробелов	Шифр Цезаря – замена символов соседними в алфавите с заданным сдвигом
3	строчные символы латинского алфавита, точки, запятые, пробелы	Запись символами латинского алфавита	Замена запятых точками
4	строчные и прописные символы латинского алфавита, пробелы	Запись символами латинского алфавита	Замена символов после точки на заглавные
5	{0, 1, +, -}	Запись математического выражения	Замена знака на противоположный

3. Пользователь вводит текст в стихотворной форме. Реализовать поиск символа по ключу, состоящему из двух цифр – номера строки и номера символа в строке. Количество символов в строках не одинаково. Реализовать проверку на отсутствие такого символа в стихотворении.
4. Задача обратная задаче 3. Пользователь вводит текст в стихотворной форме. Реализовать поиск всех ключей для введенного символа. В случае, если символа нет в тексте, выводить предупреждающее сообщение.
5. Имеются две строки A и B. Задача — найти такую строку C, которая содержит в себе и A и B в качестве подстрок и является кратчайшей среди всех таких возможных строк.
6. Проверить является ли введенный текст палиндромом.
7. Введен текст. Слова разделены пробелами. Подсчитать количество слов, количество символов в каждом слове и среднее количество символов в слове.
8. Введен текст. Слова разделены пробелами. Предложения – точками. Подсчитать количество предложений, количество слов в предложениях, среднее количество слов в предложении, количество символов в предложении.
9. Введен текст. Подсчитать статистику – количество встречи каждого символа в тексте.
10. Введен текст. Стереть все подряд идущие символы.

Практическая работа №5. Преобразование строк

Реализация всех задач происходит 2 способами – для Си-строк и класса string

Контрольная сумма. Для контроля передачи сообщения в сетях (в целях отслеживания потерь и изменений сообщения) используются контрольные суммы, рассчитанные по некоторым алгоритмам, известным участникам передачи сообщения. При получении сообщения пересчитывается контрольная сумма и сравнивается с переданной суммой. Контрольная сумма добавляется в конец строки.

1. Реализуйте расчет контрольной суммы строки способом бит четности (работа ведется со строками, переведенными в битовый формат). Бит чётности или контрольный разряд формируется при выполнении операции «Исключающее-ИЛИ» последовательно (1 бит с 2, результат с 3, результат с 4 и т.д.).
2. Реализуйте расчет контрольной суммы строки, как сумму переданных бит. CRC = byte(1)+byte(2)+byte(3)+…

Алгоритмы сжатия.

3. Реализуйте сжатие алгоритмом Хаффмана. В алгоритме наиболее часто встречающиеся символы обозначаются более короткими кодами. Для построения алгоритма Хаффмана используется таблица частот символов. Затем строится дерево, где вершинами являются символы, а пути до вершины – кодами. На каждом шаге берутся два символа с минимальной частотой, и объединяются в новые символы (частота которых равна сумме двух частот).
4. Реализуйте метод сжатия на основе счетчика повторяющихся символов. Так строка aabcccccaaa должна превратиться в a2b1c5a3. Если «сжатая» строка оказывается длиннее исходной, метод должен вернуть исходную строку.

Хэш-функции. Хэш-функциями называют функции, осуществляющие преобразование массива входных данных произвольной длины в (выходную)

битовую строку установленной длины, выполняемое определённым алгоритмом. Обратного преобразования методы хэширования не предполагают. Хеш-функции применяются в следующих случаях: при построении ассоциативных массивов; при поиске дубликатов в сериях наборов данных; при построении уникальных идентификаторов для наборов данных; при вычислении контрольных сумм от данных (сигнала) для последующего обнаружения в них ошибок (возникших случайно или внесённых намеренно), возникающих при хранении и/или передаче данных; при сохранении паролей в системах защиты в виде хеш-кода (для восстановления пароля по хеш-коду требуется функция, являющаяся обратной по отношению к использованной хеш-функции); при выработке электронной подписи (на практике часто подписывается не само сообщение, а его «хеш-образ») и др.

Для преобразования исходные строки могут разбиваться на блоки одинаковой длины. Каждый блок обрабатывается хэш-функцией отдельно. Затем результаты стыкуются по некоторому алгоритму (конкатенация строк, побитовое сложение, «исключающее или» и так далее)

5. Реализовать вычисление хэш-функции как остатка от деления на число возможных выходных данных. $h(k) = k \bmod M$. Для работы используется преобразованная в бинарный вид исходная строка текста.

Практическая работа №6. Файлы

1. Реализовать программу, записывающую в файл результаты вычисления корней квадратного уравнения.
2. Реализовать программу, вычисляющую количество символов в файле.
3. Реализовать программу, вычисляющую количество строк в файле.
4. Реализовать программу, считывающую из файла только четные строки.
5. Реализовать программу, дописывающую в список введенную фамилию и инициалы.
6. Для задания 5 предусмотреть проверку на повторные записи.
7. Реализовать программу, стирающую из файла все числовые символы.
8. Текстовый файл представляет собой перечень логинов и паролей. Написать программу, считывающую необходимый пароль по логину.
9. Осуществить копирование первых слов каждой строки предложения файла в другой файл.
10. Реализовать программу, перезаписывающую файл в обратном порядке и создающую копию исходного файла.

Практическая работа №7. Программирование на C#

1. Реализуйте на C# программу, иллюстрирующую неявную типизацию.

2. Реализуйте на C# программу для работы с матрицами.

Вариант	Задание
1	Произведение произвольных матриц, введенных пользователем
2	Транспонирование матрицы, хранящейся в сжатом виде (матрица с большим количеством нулевых элементов)
3	Поиск максиминного элемента (максимального среди минимальных по строкам)
4	Вычисление матрицы для поиска главных миноров (без вычисления определителей)
5	Вычисление суммы элементов матрицы, максимального и минимального элемента матрицы

3. Реализуйте на C# программу для сортировки массива строк, введенного пользователем.

4. Пользователь вводит результаты тестирования по нескольким показателям для группы обучающихся. Реализуйте на C# программу, вычисляющую средний балл для каждого обучающегося, для каждой дисциплины и в общем по группе. Используйте для реализации массив массивов.

5. Реализуйте на C# программу шифрования текста методом замены (например, шифр Цезаря) и вычисления контрольной суммы полученного шифротекста. Рассчитайте статистические показатели встречи каждого символа в исходном тексте и в зашифрованном.

Практическая работа №8. Windows-form приложения на C#

1. (1 балл) Реализовать форму, обеспечивающую выбор цвета фона полей ввода. (Например, поле с названием цвета и кнопка для реализации события)
2. (1 балл) Реализовать форму для ввода матрицы произвольного размера (по выбору пользователя) и расчета её ранга.
3. (1,5 балл) Реализовать простейший калькулятор с помощью приложения windows-form.
4. (2 балл) Реализовать приложение, строящее график функции с учетом коэффициентов.

Вариант:	Функция
1	$A \cdot \cos(B \cdot x)$
2	$A \cdot \sin(x+B)$
3	$A+\cos(B+x)$
4	$A \cdot \sin(x \cdot B)$
5	$A \cdot \sin(x)+B$

Практическая работа №9. Обработка исключений в C#

1. Реализовать программу, в ходе которой могут возникать исключения следующих типов:

- **DivideByZeroException**: представляет исключение, которое генерируется при делении на ноль
- **ArgumentOutOfRangeException**: генерируется, если значение аргумента находится вне диапазона допустимых значений
- **ArgumentException**: генерируется, если в метод для параметра передается некорректное значение
- **IndexOutOfRangeException**: генерируется, если индекс элемента массива или коллекции находится вне диапазона допустимых значений
- **InvalidCastException**: генерируется при попытке произвести недопустимые преобразования типов
- **NullReferenceException**: генерируется при попытке обращения к объекту, который равен null (то есть по сути неопределен)

2. Реализовать обработку данных исключений, выводящие всю информацию – тип исключения, причины ошибки и т.д. Обработчик должен завершать работу.

3. Изменить обработчик так, чтобы пользователю была предложена возможность изменить введенные значения или введенные значения изменялись на значения по умолчанию.

4. Реализовать обработку остальных исключений (Exception ex)

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Основная учебная литература

1. Трофимов, В.В. Алгоритмизация и программирование : учебник / В.В. Трофимов, Т.А. Павловская. – Москва : Издательство Юрайт, 2019. – 137 с. – ISBN 978-5-534-07834-3. – URL: https://www.biblio-online.ru/viewer/algoritmizaciya-i-programmirovanie-423824. - (дата обращения: 22.03.2020). – Текст : электронный.

Дополнительная литература

1. Крупский, В. Н. Теория алгоритмов. Введение в сложность вычислений : учебное пособие для бакалавриата и магистратуры / В. Н. Крупский. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 117 с. — (Авторский учебник). — ISBN 978-5-534-04817-9. — URL: https://biblio-online.ru/bcode/444131 (дата обращения: 26.11.2019). — Текст : электронный.

2. Судоплатов, С. В. Математическая логика и теория алгоритмов : учебник и практикум для академического бакалавриата / С. В. Судоплатов, Е. В. Овчинникова. — 5-е изд., стер. — Москва : Издательство Юрайт, 2019. — 255 с. — (Высшее образование). — ISBN 978-5-534-00767-1. — URL: https://biblio-online.ru/bcode/432018 (дата обращения: 26.11.2019). — Текст : электронный.

3. Язык программирования Си++. Курс лекций : учебное пособие / Фридман А.Л. – Москва : ИНТУИТ.РУ «Интернет-университет Информационных Технологий», 2004. – 262 с. – ISBN 5-9556-0017-5. - URL: http://biblioclub.ru/index.php?page=book_view_red&book_id=233058. - (дата обращения: 22.03.2020). – Текст : электронный.

4. Сузи, Р.А. Язык программирования Python: учебное пособие / Р.А. Сузи. – 2-е изд., испр. – Москва: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 327 с. - ISBN 978-5-94774-711-9. – URL:

http://biblioclub.ru/index.php?page=book_view_red&book_id=233288. - (дата обращения: 22.03.2020). – Текст : электронный.

5. Мирошниченко, И.И. Языки и методы программирования: учебное пособие / И.И. Мирошниченко, Е.Г. Веретенникова, Н.Г. Савельева. – Ростов н/Д: Издательско-полиграфический комплекс Рост. гос. экон. ун-та (РИНХ), 2019. – 188 с. - ISBN 978-5-7972-2604-8. – URL: http://biblioclub.ru/index.php?page=book_view_red&book_id=567706. - (дата обращения: 22.03.2020). – Текст : электронный.

СОВРЕМЕННЫЕ ПРОФЕССИОНАЛЬНЫЕ БАЗЫ ДАННЫХ И СПРАВОЧНЫЕ СИСТЕМЫ

CITForum.ru : on-line библиотека свободно доступных материалов по информационным технологиям на русском языке : сайт. – 2001 – URL: http://citforum.ru (дата обращения: 22.03.2020). – Текст: электронный.

eLIBRARY.RU : научная электронная библиотека : сайт. – Москва, 2000 - . – URL: http://www.elibrary.ru (дата обращения: 22.03.2020). – Режим доступа: для зарегистрир. пользлвателей. – Текст: электронный.

Единое окно доступа к образовательным ресурсам : сайт. – Москва, 2005 - . – URL: http://window.edu.ru/ (дата обращения: 22.03.2020). –Текст: электронный.