

К. В. Розов

Новосибирский государственный педагогический университет;
гимназия № 4, г. Новосибирск, Россия

РАЗВИТИЕ АЛГОРИТМИЧЕСКОЙ КУЛЬТУРЫ И КУЛЬТУРЫ ПРОГРАММИРОВАНИЯ У УЧАЩИХСЯ ПРОФИЛЬНОГО КЛАССА ПРИ ОБУЧЕНИИ ОСНОВАМ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

Аннотация

В статье приведены структура, содержание и результаты апробации курса программирования на языке C++, разработанного для десятого класса физико-математического профиля и реализующегося в рамках учебного предмета «Информатика». Целью курса является развитие у обучающегося не только знаний и навыков в области программирования, но и его алгоритмической культуры и культуры программирования как важных качеств личности потенциального специалиста в области современных высоких технологий. Этому способствует особый контроль образовательного процесса со стороны учителя, заключающийся в мониторинге деятельности обучающихся по написанию программ и своевременной коррекции этой деятельности. Оценка уровня развития алгоритмической культуры и культуры программирования обучающегося относительно базового уровня их сформированности (при освоении основ алгоритмизации и программирования в девятом классе) осуществлялась на основании ряда критериев, представленных в данной статье. Результаты апробации позволили предположить, что специально организованная деятельность учителя по преподаванию приведенного курса позволяет повысить уровень алгоритмической культуры и культуры программирования старшеклассника при изучении основ программирования на языке C++.

Ключевые слова: алгоритмическая культура, культура программирования, программирование, обучение программированию, C++, информатика, профильное обучение.

DOI: 10.32517/2221-1993-2020-19-6-24-33

1. Введение

Обучающиеся профильных классов физико-математической направленности являются потенциальными студентами вузов технического профиля, где программирование на актуальных на рынке труда высокоуровневых языках будет являться частью либо основой их профессиональной подготовки к будущей профессии. «Социальный заказ» общества и государ-

ства на специалистов в области информационных технологий, в частности программистов и специалистов по анализу данных, определяет актуальность изучения программирования в классах физико-математического профиля на углубленном уровне. Согласно требованиям к предметным результатам освоения углубленного курса информатики ФГОС среднего общего образования, обучающиеся должны владеть универсальным языком программирования высокого уровня, представлениями

Контактная информация

Розов Константин Владимирович,

аспирант кафедры педагогики и психологии ИФМИТО, ассистент кафедры информационных систем и цифрового образования, Институт физико-математического, информационного и технологического образования, Новосибирский государственный педагогический университет, Россия; *адрес:* 630126, Россия, г. Новосибирск, ул. Вилюйская, д. 28;

учитель информатики, гимназия № 4, г. Новосибирск, Россия; *адрес:* 630004, г. Новосибирск, ул. Сибирская, д. 35;

e-mail: konstantin_dubrava@mail.ru

K. V. Rozov

Novosibirsk State Pedagogical University;
Gymnasium 4, Novosibirsk, Russia

THE DEVELOPMENT OF ALGORITHMIC CULTURE AND PROGRAMMING CULTURE OF STUDENTS IN PROFILE CLASSES WHILE STUDING THE BASIS OF PROGRAMMING IN C++

Abstract

The article presents the structure, content and results of approbation of the C++ programming course developed for the 10th grade students of physics and mathematics profile and implemented as part of the academic subject "Informatics". The aim of the course is to develop in the student not only knowledge and skills in programming, but also his algorithmic culture and programming culture as important qualities of a potential IT-specialist. This is facilitated by special control of educational process by the teacher, which consists in monitoring the activities of students in writing programs and timely correction of this activity. The assessment of the level of development of student algorithmic culture and programming culture relative to the basic level of their formation (when mastering the basics of algorithmization and programming in the 9th grade) was carried out on the basis of a number of criteria presented in the article. The results of approbation showed that the specially organized teacher activity makes it possible to increase the level of algorithmic culture and programming culture of high school students when studying the basics of programming in C++.

Keywords: algorithmic culture, programming culture, programming, teaching programming, C++, informatics, profile education.

о базовых типах данных и структурах данных; умением использовать основные управляющие конструкции; владеть навыками и опытом разработки программ в выбранной среде программирования, включая тестирование и отладку программ; владеть элементарными навыками формализации прикладной задачи и документирования программ [8].

На базе МБОУ «Гимназия № 4» г. Новосибирска реализуется физико-математический профиль обучения в X—XI классах. Углубленное изучение программирования в гимназии с использованием языков C++ и Python (помимо классического для школ языка Pascal) призвано сформировать у обучающихся знания и навыки в области современного программирования. При этом важной задачей учителя является формирование и развитие алгоритмической культуры обучающихся и взаимосвязанной с ней культуры программирования.

Разработкой проблемы формирования алгоритмической культуры школьников при обучении математике и информатике занимались М. П. Лапчик [3, 5], Л. Г. Лучко [4], В. М. Монахов [5], А. А. Шрайнер [9] и др. Как отмечает Л. Л. Босова, формирование алгоритмической культуры, согласно ФГОС, является одним из ключевых направлений обучения информатике [1].

Анализ понятия «алгоритмическая культура» и авторское уточнение некоторых признаков сформированности алгоритмической культуры у обучающегося в рамках учебного предмета «Информатика» представлены нами в работе [7]. В настоящей статье рассматривается организация профессионально-ориентированного курса обучения основам программирования на языке C++, разработанного для X класса физико-математического профиля. Представлены результаты освоения этого курса обучающимися, отражающие развитие их алгоритмической культуры и культуры программирования относительно базового уровня, сформированного у школьников при изучении основ алгоритмизации и программирования в IX классе. (Исследование уровня развития у обучающихся умений записи алгоритмов, обработки данных на языке программирования и навыков отладки программ в среде программирования выходит за рамки данной статьи.)

Язык C++ был выбран, прежде всего, по следующим причинам:

- широкое использование языка в профессиональной среде и, как следствие, при обучении программированию в технических вузах;
- статическая типизация, оказывающая дисциплинирующее воздействие на обучающихся при работе с данными;
- Си-подобный синтаксис, являющийся основой для множества других языков программирования: C#, Java, JavaScript, PHP и др., привыкание к которому может способствовать ускоренной адаптации обучающегося к новым правилам при изучении одного из этих языков в будущем.

На выбор языка также повлияли:

- возможность программировать на нем микроконтроллерную плату Arduino;
- существование учебного пособия по программированию на C++ для профильных классов [6];
- наличие этого языка в КИМ ЕГЭ.

Необходимость использования на уроках авторского курса объясняется тем, что, на наш взгляд, содержание предложенного К. Ю. Поляковым курса [6] в исходном виде является сложно воспринимаемым для обучающихся, у которых предварительное знакомство с основами алгоритмизации и программирования осуществлялось только в IX классе с использованием учебного алгоритмического языка для программирования исполнителей в среде «КуМир» и языка Pascal в среде PascalABC.NET при учебной нагрузке по информатике один академический час в неделю, т. е. для учеников, имеющих сравнительно небольшой опыт программирования.

2. Структура и содержание курса

Курс программирования на языке C++ состоит из 11 тем. Он включает входное тестирование, лекции, практические занятия, проверочные работы по каждой теме, выходное тестирование. Курс рассчитан на 60 академических часов (2 часа в неделю).

В таблице 1 представлена структура курса в общем виде. Для каждой темы указано количество предлагаемых обучающимся задач (заданий по написанию программ), включая задачи из проверочных работ (по два варианта) и задачи для домашней работы.

Таблица 1

Общая структура курса программирования на языке C++ для X класса

№ п/п	Тема	Кол-во задач	Кол-во академ. часов
	Входное тестирование	30 тестовых заданий	1
1	Введение в C++	—	1,5
2	Линейные алгоритмы	24	4,5
3	Ветвление	22	8
4	Циклы	25	6
5	Функции и модули	21	6
6	Одномерные массивы	22	6
7	Двумерные массивы	20	6
8	Строки	24	6
9	Множества	16	6
10	Структуры	10	6
11	Файловый ввод-вывод	5	2
	Выходное тестирование	30 тестовых заданий	1

Тема 1. Введение в C++

В качестве введения в курс учитель представляет обучающимся краткую историю языка C++, его место в многообразии языков программирования и основные характеристики. Ключевыми элементами содержания здесь являются: эволюция языка (Си, объектный Си,

C++); его разработчик — Бьерн Страуструп (Bjarne Stroustrup); год создания первого компилятора — 1983 [11]. Несмотря на уже солидный для языка программирования возраст, C++ на сегодняшний день остается актуальным и широко используемым.

Язык совмещает в себе процедурную и объектно-ориентированную парадигмы программирования, является компилируемым и имеет статическую типизацию. Здесь следует напомнить обучающимся **значения понятий «компилируемый» и «статическая типизация»**, сравнив, соответственно, две технологии трансляции — компиляцию и интерпретацию — и рассмотрев сущность статической и динамической типизации. Следует добавить, что язык C++ имеет шесть стандартов различных версий, номера которых соответствуют году их появления: C++98, C++03, C++11, C++14, C++17 и C++20. Эта информация понадобится обучающимся в будущем, когда они столкнутся с ошибками компиляции при отсутствии синтаксических ошибок, что потребует для устранения этих ошибок изменения настроек компилятора, включение поддержки более нового стандарта языка.

Обучающимся предлагается использовать одну из свободно распространяемых **интегрированных сред разработки программ** (англ. Integrated Development Environment, далее — IDE) на языке C++: Dev-C++ или Code::Blocks. Интерфейс обеих IDE интуитивно понятен. Первую среду разработки рекомендует К. Ю. Поляков [10], вторая изначально имеет более удобные настройки редактора кода и консоли ввода-вывода. По сравнению с Visual Studio, обе IDE занимают гораздо меньше места на внешнем носителе. Их функционала достаточно для изучения основ программирования на языке C++ в школе.

Учителю необходимо **рассмотреть такую особенность сред разработки на языке C++, как генерация базового кода**. При создании нового проекта консольного приложения в средах разработки генерируется базовый код, показанный в таблице 2.

Как видно из таблицы 2, возникает проблема, связанная с наличием базового кода, который в разных средах разработки различается и который необходимо разъяснять обучающимся. Если бы среда не добавляла код автоматически, часть приведенного кода ученикам все равно пришлось бы написать для работы простейшей программы. Даже для обучающихся профильного класса, имеющих опыт программирования (преимущественно

на языке Pascal), код не является очевидным. Подключение модулей (*#include*), указание пространства имен (*using namespace std*), главная функция с возвратом значения (*int main()*, *return*), типы данных (*int*), массив указателей (*char *argv[]*), объекты потоков ввода/вывода (*cin*, *cout*) — каждый из перечисленных элементов требует короткого разъяснения. Более подробно все эти элементы будут рассмотрены в соответствующих темах. Сейчас обучающимся важно знать, что собственный код программы им надо писать между операторными (фигурными) скобками функции *main* до ключевого слова *return*.

При запуске программы сначала выполняются команды внутри функции *main*. Стоит обратить внимание обучающихся на то, что существует несколько стандартных форм записи функции *main*. Две из них показаны в таблице 2. В Dev-C++ функция имеет несколько параметров, которые нужны только в случае, если предполагается передавать в нее некоторые значения через консоль, из которой был осуществлен запуск программы. В ходе занятий обучающиеся будут запускать программы только напрямую из среды разработки, поэтому использование такой записи функции *main* не является для них актуальным. Здесь обучающимся рекомендуется удалять параметры (аргументы функции), которые все равно не будут использоваться. Программа не должна содержать лишние, неиспользуемые элементы.

Выполнение инструкции *return 0* означает успешное завершение программы (с кодом 0).

Библиотеки, содержащие дополнительные функции, подключаются в верхней части программы до строки *using namespace std*. Сущность пространства имен можно коротко проиллюстрировать на абстрактном примере двух модулей, каждый из которых включает в себя некоторую функцию. В обоих модулях эти функции имеют одинаковые имена, но различную реализацию. При этом оба модуля должны быть подключены к программе, так как каждый из них содержит еще несколько уникальных и необходимых функций. Чтобы компилятор правильно «понял», какую из двух функций с одинаковыми именами использовать, требуется указать, в каком пространстве имен ее надо искать. Оба модуля имеют собственные пространства имен с разными названиями, поэтому мы можем явно указать принадлежность функции определенному модулю. Все это необходимо для первичного понимания обучающимися, что объекты потоков ввода/вывода *cin*, *cout* и функция *endl* — перевод

Таблица 2

Базовый код в Dev-C++ и Code::BlocksDev

Dev-C++	Code::Blocks
<pre>#include <cstdlib> #include <iostream> using namespace std; int main(int argc, char *argv[]) { cout << "Press the enter key..."; cin.get(); return EXIT_SUCCESS; }</pre>	<pre>#include <iostream> using namespace std; int main() { cout << "Hello world!" << endl; return 0; }</pre>

строки с очисткой потока — относятся к пространству имен *std*. Инструкция *using namespace std* означает, что имена следует искать именно в этом пространстве имен. Альтернативный вариант — не использовать данную инструкцию, а указывать пространство имен непосредственно перед использованием имени с помощью оператора расширения области видимости, например: *std::cout*.

Учителю следует запустить программу и продемонстрировать ее рабочее состояние, показать, как выглядит консоль ввода-вывода. В зависимости от используемой среды разработки вид консоли может отличаться. Так, при использовании Dev-C++ консоль после запуска тут же закроется, и учителю необходимо показать способы задержки консоли на экране, чтобы обучающиеся в дальнейшем могли видеть результат работы своих программ. Например, использовать функцию *getch* из библиотеки *conio* перед оператором *return* в функции *main*. При использовании Code::Blocks задержка консоли на экране дополнительными средствами не требуется, однако в консоль автоматически выводится дополнительная информация о том, с каким кодом завершилась программа и сколько времени потребовалось на ее исполнение.

Следующей дидактической единицей содержания курса являются **типы данных**. Рассматриваемые типы: целочисленные (*short*, *int*, *long*), вещественные (*float*, *double*), логический (*bool*), символьный (*char*), строковый (*string*). Тип *char* является целочисленным, но предназначен для обработки символов. Переменная типа *bool* может принимать значения в двух формах: *true/false* (рекомендуется), либо 0/1 (вместо единицы может быть другое, ненулевое числовое значение). Учитель приводит примерные диапазоны значений перечисленных типов и указывает, что некоторые из них зависят от конкретного компилятора или архитектуры компьютера.

Затем необходимо рассмотреть одну из особенностей языка C++ по сравнению с языками Pascal и Python — **модификаторы типов**. Модификаторы используются для изменения границ диапазона типов данных и делятся на две группы: модификаторы диапазона с учетом знака (*signed*, *unsigned*) и модификаторы изменения размера диапазона (*short*, *long*). Модификаторы *signed*, *unsigned*, *long* и *short* могут применяться к целочисленным типам. К типу данных *double* можно применить модификатор *long*. К типу *char* можно применить *signed* или *unsigned*. Существуют также типы данных, название которых содержит комбинацию символов *_t*, например, *int32_t*, *uint32_t*, *int64_t*, *uint64_t*. Такие типы имеют фиксированный размер независимо от компилятора.

Здесь стоит упомянуть о необходимости наличия множества различных похожих типов данных. Она обусловлена, прежде всего, разнообразием устройств, для которых создаются программы на C++, и необходимостью обеспечения совместимости программ между различными устройствами и портируемости кода.

Далее учитель приводит правила объявления **переменных**, демонстрирует примеры создания одной переменной и нескольких переменных одного типа, в том числе с использованием модификаторов. Здесь следует указать учащимся на то, что переменная может быть создана условно в любом месте программы, а не

в специально отведенной для этого области, как было в Pascal. При создании переменной ее сразу можно инициализировать. Стоит напомнить обучающимся правила создания имен идентификаторов (имен переменных и функций) и предупредить, что в языке C++ учитывается регистр символов (букв).

Следующий элемент содержания — **константы**. При этом под константой можно понимать две различные сущности: переменная, объявленная через ключевое слово *const* и доступная только для чтения, и директива компилятору *#define*. Необходимо упомянуть, что в отличие от *const* замещение некоторой константы (числа) в программе с помощью *#define* не занимает места в памяти, однако не подлежит проверке на тип данных, что может привести к ошибкам.

Завершить рассмотрение типов данных учитель может демонстрацией фактического размера переменных различных типов, используя оператор *sizeof*.

Ввод и вывод в программе консольного приложения осуществляется с помощью стандартных потоков ввода (*cin*) и вывода (*cout*). Нет необходимости углубляться в понятие потока, достаточно дать обучающимся знать, что *cin* и *cout* не просто функции для ввода/вывода, как *readln* и *writeln* в Pascal. Чтобы обучающимся было проще запомнить, в какую сторону должны указывать «стрелки» (комбинации символов *>>* после *cin* и *<<* после *cout*), можно привести следующую аналогию: так как *ввод* происходит *в* переменную, стрелка указывает *в нее*; так как *вывод* происходит *из* переменной на экран, то стрелка показывает *от* переменной в сторону вывода. Для упрощения понимания обучающимися принципов работы с объектом выходного потока можно сказать, что стрелки используются вместо привычных запятых для разделения элементов разных типов. Для корректного вывода надписей на русском языке (символов кириллицы) в консоль потребуется использовать дополнительную инструкцию *setlocale*.

Тема 2. Линейные алгоритмы

Чтобы обучающиеся могли начать писать простейшие линейные программы, осталось представить им **арифметические операторы и операторы присваивания**. Следует акцентировать внимание на том, что обычный оператор присваивания имеет вид символа «*=*» без двоеточия, как в Pascal. В C++ также присутствуют краткие операторы присваивания вместе с арифметическими операциями (*+=*, *-=*, **=*, */=*, *%=*), которые отсутствуют в классическом Pascal, но есть в Python. В C++ есть особые операторы инкремента и декремента (*++* и *--*), которых нет ни в Pascal, ни в Python. Важным отличием C++ является отсутствие отдельного оператора для целочисленного деления. Компилятор определяет тип результата в зависимости от того, какие типы имеют делимое и делитель. Для целочисленного деления требуется, чтобы и делимое, и делитель были целыми, иначе будет произведена операция вещественного деления. Если все же необходимо произвести вещественное деление с целыми числами, то можно видоизменить запись константы (числа), добавив к ней точку и ноль либо выполнив приведение типа переменной к вещественному.

Для решения некоторых задач требуется использовать **функции** для вычисления квадратного корня,

модуля числа, возведения числа в степень, тригонометрические функции. Здесь обучающиеся в первый раз пробуют подключить и использовать библиотеку, так как набор таких функций содержится в библиотеке *cmath*.

Чтобы вывести число в определенном формате, с заданным общим количеством знаков и количеством знаков после запятой, требуется подключить библиотеку *iomanip* и использовать манипуляторы *setw* и *setprecision* соответственно.

Тема 3. Ветвление

Изучение темы «Ветвление» начинается с рассмотрения операторов сравнения (<, >, <=, >=, ==, !=), где особое внимание заслуживает оператор строгого равенства. Именно неверное написание этого оператора является наиболее распространенной ошибкой у обучающихся, так как написание одного символа «=» вместо двух, во-первых, является более интуитивным, во-вторых, применялось в языке Pascal. Проблему усугубляет то, что при написании одного символа равенства в условии компилятор не выдает ошибку, а условие всегда будет истинным, что не всегда просто заметить в процессе тестирования программы.

После этого рассматриваются три оператора для организации ветвления.

Условный оператор *if* требует записи условия в фигурных скобках, а тело условного оператора при количестве инструкций больше одной требует использования операторных скобок (фигурных скобок). Следует акцентировать внимание обучающихся на том, что в отличие от языка Pascal перед оператором *else* (при его наличии) необходимо ставить точку с запятой.

Тернарный условный оператор удобно рассматривать после оператора *if*. Практика показала, что при решении задач обучающимися этот оператор используется довольно редко, однако в задачах на заполнение двумерного массива двумя значениями некоторые ученики начинают активно его применять, заметив, что его использование вместо оператора *if* в таких задачах упрощает код. Это можно интерпретировать как один из признаков развития у таких учащихся алгоритмической культуры.

Работу оператора множественного выбора *switch* можно продемонстрировать на простой задаче: пользователь вводит номер дня недели, программа выводит название этого дня. Здесь важно обратить внимание обучающихся на то, что для выполнения инструкций только для одного случая (*case*) следует использовать оператор *break*.

Далее учитель напоминает учащимся **основные логические операции** и демонстрирует форму их записи на языке C++. Для формирования навыков построения сложных условий, требующих использования нескольких логических операторов, используются задачи проверки попадания точки в закрашенную область на координатной плоскости.

Тема 4. Циклы

Изучение темы «Циклы» начинается с рассмотрения оператора цикла *for*, применяющегося в том случае, когда известно необходимое количество итераций. Оператор состоит из трех частей, записывающихся в скобках: область инициализации счетчика, условие

работы цикла, область изменения значения счетчика. Переменная-счетчик может быть создана заранее либо в области инициализации. В области изменения значения счетчика происходит его уменьшение или увеличение с требуемым шагом с помощью какого-либо арифметического оператора и присваивания.

Цикл с предусловием, реализующийся с помощью оператора *while*, является более универсальным, но применяется преимущественно в тех случаях, когда количество итераций заранее неизвестно.

Цикл с постусловием, реализующийся с помощью оператора *do..while*, отличается от *while* тем, что тело цикла выполнится хотя бы один раз. При этом, в отличие от аналогичного оператора *repeat-until* в языке Pascal, постусловие является условием *продолжения* цикла (как и предусловие в *while*), а не окончания.

Тема завершается рассмотрением операторов *break* для досрочного завершения цикла и *continue* для пропуска итерации.

Тема 5. Функции и модули

В рамках темы «Функции и модули» обучающиеся знакомятся с правилами создания двух видов подпрограмм: функций, возвращающих значения (аналог *function* в Pascal), и функций, не возвращающих значения (аналог *procedure* в Pascal). Возвращающая значение функция содержит оператор возврата *return*, который завершает выполнение функции и возвращает (передает) вычисленное в ее теле значение (результат) в место вызова этой функции. Не возвращающая значение функция обозначается типом (возвращаемого значения) *void* и не должна содержать *return*. Возвращающие значения функции обычно имеют как минимум один параметр, не возвращающие могут не иметь параметров. При передаче параметра в функцию C++ делает копию значения этого параметра. Чтобы этого не происходило, можно воспользоваться указателями и передавать параметры по адресу. Используя указатели, можно изменить значение переменной с помощью не возвращающей значение функции. Так учащиеся знакомятся с переменными-указателями и оператором взятия адреса в языке C++.

Далее обучающиеся узнают, как создать собственный модуль (библиотеку), создав файл заголовка (.h) и файл реализации (.cpp), и правила его подключения к программе. Код всех созданных функций учащиеся размещают в созданном ими модуле, затем используют этот модуль для тестирования работы функций и при решении задач по другим темам.

В завершении темы обучающиеся знакомятся с понятием рекурсии и создают несколько рекурсивных функций: вычисления факториала, *n*-го члена последовательности чисел Фибоначчи, наибольшего общего делителя двух целых чисел.

Тема 6. Одномерные массивы

В начале изучения темы «Одномерные массивы» обучающиеся знакомятся с двумя способами создания одномерного статического массива: объявление массива заранее известного размера и динамическое создание массива с помощью оператора *new*. Особенностью массива в C++ является то, что индекс его первого элемента всегда равен 0.

При изучении алгоритмов обработки массивов удобно заполнять массивы случайными числами.

Прежде чем перейти к рассмотрению способов заполнения одномерного массива, учитель знакомит обучающихся с особенностями работы генератора случайных чисел *rand* из библиотеки *cstdlib*. Для того чтобы при перезапуске программы генерировались новые значения, в функцию *srand*, устанавливающую начало цепочки случайных чисел, передается текущее системное время, возвращаемое функцией *time* из библиотеки *time*.

Далее учитель приводит способы заполнения одномерного массива: с помощью клавиатуры, случайными числами, по правилу (с помощью формулы). Необходимо обратить внимание учащихся на то, что компилятор не сообщает о выходе за границы массива в цикле — за этим надо следить самостоятельно для предотвращения ошибок времени выполнения.

Особое внимание необходимо уделить алгоритмам поиска максимального и минимального значений в массиве, так как данные алгоритмы являются базовыми для многих более сложных алгоритмов обработки массивов, встречающихся в том числе на ЕГЭ по информатике.

Для формирования понимания общих принципов сортировки массивов данных рассматриваются следующие алгоритмы: сортировка обменом (пузырьковая сортировка), быстрая сортировка.

Завершается тема изучением применения контейнера *vector* в качестве динамического одномерного массива, т. е. массива, в котором можно изменять количество элементов в ходе выполнения программы. Здесь же обучающиеся знакомятся с понятием «итератор» и функциями *begin*, *end* для получения итераторов, указывающих соответственно на первый и следующий за последним элементы вектора.

Тема 7. Двумерные массивы

Содержание темы «Двумерные массивы» в основном повторяет содержание предыдущей темы с поправкой на добавление дополнительной мерности. Однако характер задач меняется. В начале изучения темы учитель дает задания на заполнение массива по определенному правилу: в виде различных узоров из двух символов, последовательностей чисел.

Далее задачи приобретают характер поиска элементов, строк и столбцов в массиве по заданному условию. При этом важно обратить внимание обучающихся на роль каждого из двух циклов, применяющихся для обработки массива, и алгоритм корректного вывода массива на экран при наличии в нем чисел различной разрядности.

Тема 8. Строки

Для работы со строками используется тип данных (класс) *string* из пространства имен *std*. Следует сообщить обучающимся, что существует множество строковых типов данных в C++, добавляемых из различных библиотек. Тип *string* выбран как наиболее простой, не требующий подключения дополнительных модулей. По сути, это одномерный массив символов (*char*). Для обеспечения корректной обработки символов кириллицы требуется использование функций *SetConsoleCP* и *SetConsoleOutputCP* из модуля *windows* (в ОС Windows) с установкой кодировки cp1251. При этом не следует использовать *setlocale*.

Необходимо обратить особое внимание учащихся, что ввод строки, содержащей пробелы, необходимо осуществлять с помощью функции *getline*. При этом, если вводу строки предшествовал ввод числа или символа с помощью *cin*, необходимо использовать метод *cin.ignore* для символа «\n», чтобы ввод строки был корректным. Кроме символа «\n» существуют и другие специальные (управляющие) символы, среди которых следует выделить наиболее значимый для обработки строк — символ конца строки «\0».

Класс *string* содержит следующие методы:

- получение длины строки (*length / size*);
- проверка строки на отсутствие в ней символов (*empty*);
- копирование подстроки (*substr*);
- вставка подстроки (*insert*);
- поиск первого вхождения подстроки в строку слева (*find*) и справа (*rfind*);
- удаление подстроки (*erase*);
- очистка строки (*clear*).

Также рассматриваются способы преобразования строки в число и числа в строку и изменение регистра символов английского алфавита. Перечисленных методов достаточно для решения всех задач курса по работе со строками.

Тема 9. Множества

Изучение темы «Множества» начинается с повторения основных определений теории множеств — отношение принадлежности, отношение включения, понятие пустого множества, подмножества, операции над множествами (пересечение, объединение, разность) — с графической демонстрацией в виде кругов Эйлера.

Работа с множествами в C++ осуществляется с помощью контейнеров *set* (множество) и *multiset* (мультимножество). Последний может содержать одинаковые элементы.

Переменную типа «множество» можно инициализировать с помощью фигурных скобок. Пустая пара фигурных скобок — пустое множество.

В C++ реализована автоматическая сортировка элементов множества. По умолчанию элементы сортируются по возрастанию. Для сортировки по убыванию требуется явно указать параметр *greater* вместо неявно указанного *less* при описании переменной.

Чтобы использовать операции над множествами — функции *set_intersection* (пересечение), *set_union* (объединение), *set_difference* (разность), — необходимо подключить дополнительную библиотеку *algorithm*.

Доступно несколько методов обработки множества:

- добавление элементов (*insert*);
- удаление элементов (*erase*);
- очистка множества (*clear*);
- проверка на принадлежность элемента множеству (*find*);
- проверка на подмножество (*includes*).

Способов вывода элементов множества на экран существует несколько. В рамках курса рассматриваются два основных:

- вывод с помощью цикла *for* и ключевого слова *auto*;
- использование функции *copy* вместе с итератором *ostream_iterator* из библиотеки *iterator*.

Тема 10. Структуры

В рамках темы «Структуры» изучаются тип данных «перечисление» (*enum*) и тип данных *struct* как совокупность переменных (полей), объединенных общим именем, по аналогии с типом *record* в Pascal, хотя в C++ *struct* фактически является классом и может содержать не только поля, но и методы.

Теоретическое изучение *struct* ограничивается рассмотрением способов объявления структурной переменной и организацией доступа к полям структуры.

Знакомство обучающихся с классами (*class*), понятием метода класса, спецификаторами доступа (*public*, *private*), понятием экземпляра класса (объекта) реализуется в рамках домашнего задания. Учащиеся могут по желанию решить одну из задач на структуры с помощью классов.

Тема объектно-ориентированного программирования требует специально организованного изучения и выходит за рамки курса для X класса.

Тема 11. Файловый ввод-вывод

Завершается курс темой «Файловый ввод-вывод», изучение которой особенно полезно для обучающихся, участвующих в олимпиадах по программированию. В рамках темы рассматриваются потоки файлового ввода-вывода *ifstream* и *ofstream*, режимы открытия файлов.

На практическом занятии ученики реализуют чтение данных из текстового файла и запись в текстовый файл в ранее решенных задачах по различным темам.

3. Критерии оценки уровня развития алгоритмической культуры и культуры программирования

В процессе проведения практических занятий учитель контролирует написание программ обучающимися и вносит корректировки, относящиеся к развитию у обучающихся алгоритмической культуры и взаимосвязанной с ней культуры программирования [7]. Подразумевается, что все ученики уже обладают базовой алгоритмической культурой, предполагающей «понимание сущности алгоритма и его свойств; умение составить и записать алгоритм для конкретного исполнителя с помощью определенных средств и методов описания; знание основных алгоритмических структур — линейной, условной и циклической; умение воспринимать и исполнять разрабатываемые фрагменты алгоритма» [2, с. 16]. Приведенные ниже критерии оценки алгоритмической культуры позволяют оценить уровень ее дальнейшего развития у обучающегося.

Основным **требованием, относящимся к алгоритмической культуре**, является оптимальное построение алгоритма программы по следующим критериям: количество используемой памяти, количество операций (скорость вычислений). Конкретные требования зависят от решаемой задачи.

Учитель контролирует выбор учащимися типов данных. Например, если программа оперирует числами от 0 до 1000, то достаточно использовать тип *short* вместо типа *int*. Обучающиеся также должны иметь в виду, что

операции с целыми числами выполняются быстрее, чем с вещественными.

Важно контролировать также целесообразность выбора той или иной структуры данных для хранения некоторого массива значений (статический массив, динамический массив, вектор, строка, множество и т. д.).

Сокращение операций призвано ускорить программу. Например, от обучающихся требуется соблюдение такого простого правила: если какую-то константу можно вычислить заранее, надо это сделать, а не «заставлять» компьютер вычислять ее. Так, в задаче по переводу количества бит в килобайты использование двух операций деления ($x/8/1024$) нежелательно, достаточно использовать только одну ($x/8192$).

В зависимости от задачи учитель, очевидно, должен контролировать общую оптимальность алгоритма ее решения: использование *else* при работе с несколькими взаимоисключающими условиями, рациональное количество итераций в цикле, количество проходов по массиву и т. д.

Требованиями, относящимися к культуре программирования, являются:

- выравнивание кода программы;
- присвоение осмысленных имен переменным и функциям;
- обеспечение корректности ввода данных;
- комментирование кода.

Для итоговой оценки развития у обучающихся алгоритмической культуры были выделены пять критериев:

- стремление к оптимальному использованию типов и структур данных (при решении большей части задач учащийся выбирает оптимальные и адекватные задаче типы и структуры данных);
- стремление к оптимальному количеству операций (при решении большей части задач учащийся избегает использования избыточных арифметических операций и операций присваивания);
- понимание строгости выполнения инструкций формальным исполнителем (учащийся понимает, что программа выполняет именно те инструкции, которые написал он; она не может ничего делать «сама», интерпретировать его команды «не так» и т. п. Он осознает ответственность за неверную работу программы, не перекладывает эту ответственность на формального исполнителя);
- способность к выделению объектов, участвующих в алгоритме, до написания кода (учащийся способен предсказать большую часть необходимых переменных и их типы до написания кода для эффективного управления памятью);
- понимание алгоритма, записанного на другом высокоуровневом языке программирования (учащийся способен понять алгоритм, записанный на другом высокоуровневом языке; может перевести код, записанный на другом высокоуровневом языке, на знакомый ему язык программирования). Данный критерий проверяется с помощью задач по программированию из КИМ ЕГЭ. Обучающимся предлагаются программы, написанные на языках, которые они ранее не изучали: Basic, Python или учебный алгоритмический язык.

Для итоговой оценки развития у обучающихся культуры программирования были выделены также пять критериев:

- выравнивание кода (учащийся самостоятельно выравнивает код при решении всех задач в соответствии с общими рекомендациями);
- информативность имен идентификаторов (учащийся дает имена переменным и функциям в соответствии с их смыслом или назначением в программе при решении всех задач);
- обеспечение корректности ввода (при решении большинства задач учащийся реализует обработку некорректного пользовательского ввода, если ввод строго определенных значений не гарантируется условием задачи);
- реализация вывода сообщений о состоянии программы и консольного пользовательского интерфейса (при решении большинства задач учащийся реализует текстовые сообщения для пользователя,

позволяющие пользователю понять, какие данные ожидает получить программа, ввел ли он корректные данные, какой результат он получил, если отсутствие таких сообщений не регламентируется условием задачи);

- комментирование кода (учащийся оставляет комментарии в программе, отражающие назначение фрагментов кода, а не их альтернативное написание на естественном языке).

Словосочетание «всех задач» в пояснениях означает, что выполнение критерия носит постоянный, а не периодический характер.

4. Результаты апробации курса

Ниже приведены результаты педагогического наблюдения за процессом развития алгоритмической культуры (табл. 3) и культуры программирования (табл. 4) у обучающихся при освоении представленного курса

Таблица 3

Результаты педагогического наблюдения за развитием алгоритмической культуры у обучающихся X класса при освоении курса программирования на языке C++

Код обучающегося	Алгоритмическая культура									
	Стремление к оптимальному использованию типов и структур данных		Стремление к оптимальному количеству операций		Понимание строгости выполнения инструкций формальным исполнителем		Способность к выделению объектов, участвующих в алгоритме, до написания кода		Понимание алгоритма, записанного на другом высокоуровневом языке	
	до	после	до	после	до	после	до	после	до	после
1						+				+
2	+	+		+	+	+	+	+	+	+
3		+			+	+				+
4		+		+	+	+	+	+		+
5				+	+	+		+	+	+
6		+		+	+	+	+	+	+	+
7				+		+				+
8	+	+		+	+	+	+	+	+	+
9		+		+	+	+		+	+	+
10				+		+		+		+
11	+	+		+	+	+	+	+	+	+
12	+	+		+	+	+	+	+	+	+
13	+	+				+	+	+	+	+
14	+	+	+	+		+	+	+	+	+
15	+	+				+		+	+	+
16	+	+		+	+	+	+	+	+	+
17	+	+	+	+	+	+		+	+	+
18						+		+		
19		+	+	+	+	+	+	+		+
20	+	+		+	+	+			+	+
21			+	+		+	+	+	+	+

Таблица 4

Результаты педагогического наблюдения за развитием культуры программирования у обучающихся X класса при освоении курса программирования на языке C++

Код обучающегося	Культура программирования									
	Выравнивание кода		Информативность имен идентификаторов		Обеспечение корректности ввода		Реализация вывода сообщений пользователю о состоянии программы		Комментирование кода	
	до	после	до	после	до	после	до	после	до	после
1	+	+		+				+		
2	+	+		+	+	+	+	+	+	+
3		+	+			+	+	+		+
4	+	+	+	+				+		+
5		+	+	+	+	+	+	+		
6	+	+	+	+	+	+	+	+		+
7		+		+		+		+		
8	+	+	+	+		+	+	+	+	+
9	+	+		+	+	+	+	+		+
10	+	+		+		+		+		
11	+	+	+	+	+	+	+	+		+
12	+	+		+	+	+	+	+	+	+
13	+	+					+	+	+	+
14	+	+	+	+	+	+	+	+		+
15		+	+	+		+		+		
16	+	+	+	+	+	+		+	+	+
17	+	+	+	+		+	+	+	+	+
18		+		+				+		
19	+	+		+		+		+	+	+
20		+	+	+		+	+	+		+
21	+	+			+		+	+		+

программирования на языке C++, проводившегося в течение 2019/2020 учебного года. Результаты были зафиксированы в конце сентября («до») и в конце апреля («после»).

Уровень развития алгоритмической культуры и культуры программирования определяется количеством выполненных критериев (отмеченных символом «+»):

0 — базовый;
1–2 — низкий;
3–4 — средний;
5 — высокий.

На рисунке показано соотношение уровней развития алгоритмической культуры (слева) и культуры программирования (справа) в начале и после освоения курса.

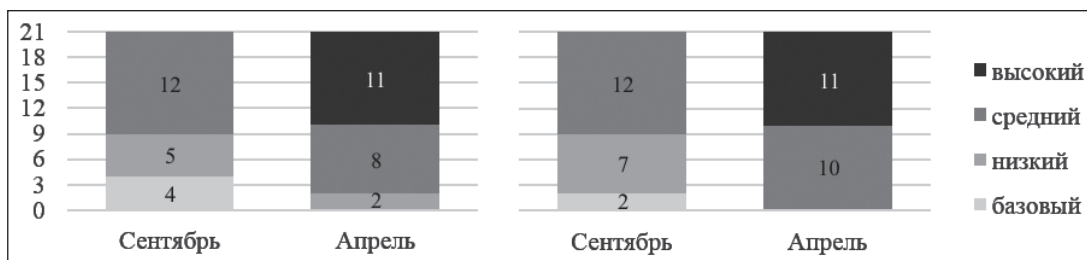


Рис. Соотношение уровней развития алгоритмической культуры (слева) и культуры программирования (справа) в начале и после освоения курса

5. Выводы

Результаты апробации позволили предположить, что специально организованная деятельность учителя по преподаванию приведенного в статье курса программирования на языке C++ позволяет повысить уровень развития алгоритмической культуры и культуры программирования старшеклассника. Проверка данного предположения требует дополнительного исследования.

Список использованных источников

1. Босова Л. Л. О курсе информатики в российской школе // Известия Кыргызской академии образования. 2015. № 3. С. 332–335.
2. Босова Л. Л., Босова А. Ю. Информатика. 7–9 классы. Методическое пособие. М.: БИНОМ. Лаборатория знаний, 2016. 464 с. <http://lbz.ru/metodist/iunk/informatics/files/bosova-7-9-met.pdf>
3. Лапчик М. П. Использование общеобразовательных аспектов программирования для ЭВМ в совершенствовании среднего математического образования: автореф. дис. ... канд. пед. наук. М.: НИИ СиМО АПН СССР, 1974. 26 с.
4. Лучко Л. Г. Формирование алгоритмической культуры учащихся в процессе обучения базовому курсу информатики: дис. ... канд. пед. наук. Омск, 1999. 152 с.
5. Монахов В. М., Лапчик М. П., Демидович Н. Б., Червошкина Л. П. Формирование алгоритмической культуры школьника при обучении математике: пособие для учителей. М.: Просвещение, 1978. 94 с.
6. Поляков К. Ю. Программирование. Python. C++: учебное пособие. В 4 ч. Ч. 1. М.: БИНОМ. Лаборатория знаний, 2019. 144 с.
7. Розов К. В., Розова М. С. Формирование и развитие алгоритмической культуры в рамках учебного предмета «Информатика» // Вестник педагогических инноваций. 2019. № 4. С. 98–104.
8. Федеральный государственный образовательный стандарт среднего общего образования. <https://fgos.ru/>
9. Шрайнер А. А. Повышение качества математического образования учащихся посредством формирования и развития их алгоритмической культуры: автореф. дис. ... канд. пед. наук. Новосибирск, 1997. 18 с.
10. Dev-C++ оболочка для программирования в Windows // Сайт Константина Полякова. <http://kpolyakov.spb.ru/school/c/devcpp.htm>
11. Seed G. M. An Introduction to Object-Oriented Programming in C++: with Applications in Computer Graphics. Springer Science & Business Media, 2012. 972 p.

НОВОСТИ

Опубликованы проекты контрольных измерительных материалов ЕГЭ 2021 года

Федеральный институт педагогических измерений (ФИПИ) опубликовал на своем сайте проекты документов, регламентирующих структуру и содержание контрольных измерительных материалов (КИМ) единого государственного экзамена (ЕГЭ) в 2021 году. На сайте представлены материалы по 15 предметам, которые сдаются в форме ЕГЭ.

Наиболее существенные изменения произошли в КИМ ЕГЭ по информатике и ИКТ, что связано с переводом этого экзамена с 2021 года в компьютерную форму. Это позволило включить в КИМ задания на практическое программирование: составление и отладка программы в выбранной участником среде программирования, работа с электронными таблицами и информационный поиск. Таких заданий в работе девять, т. е. треть от общего количества.

Остальные 18 заданий сохраняют глубокую преемственность с КИМ ЕГЭ прошлых лет, когда экзамен проводился на бумажных бланках. При этом они адаптированы к новым условиям сдачи экзамена в тех случаях, когда это было необходимо. Так, например, задание 6 КИМ 2021 года является преемником задания 8 модели КИМ предыдущих лет. В заданиях этой линии нужно было выполнить фрагмент программы вручную, что в условиях доступности компьютера со средами программирования делает задание тривиальным. Поэтому при сохранении

тематики задания была скорректирована постановка вопроса в сторону анализа соответствия исходных данных программы заданному результату ее работы.

В отличие от бланковой модели экзамена, в 2021 году выполнение заданий по программированию допускается на языках программирования (семействах языков): C++, Java, C#, Pascal, Python, школьный алгоритмический язык. Из примеров фрагментов кода в заданиях, в связи с не востребоваанностью, исключены примеры на Бейсике.

Также ФИПИ опубликовал проекты документов, определяющих структуру и содержание КИМ основного государственного экзамена (ОГЭ) в 2021 году. На сайте представлены материалы по 14 предметам, которые сдаются в формате ОГЭ. Из-за эпидемиологической ситуации ОГЭ в 2020 году проведен не был. Тем не менее в проекты КИМ 2021 года внесен ряд изменений по сравнению с КИМ, разработанными для проведения экзаменов в 2020 году. По информатике изменения структуры и содержания КИМ ОГЭ по сравнению с 2020 годом отсутствуют.

ФИПИ приглашает экспертное и профессиональное сообщества принять участие в обсуждении проектов экзаменационных материалов 2021 года. Все замечания и предложения принимаются на электронный адрес fpri@fipi.ru до 1 октября 2020 года.

(По материалам, предоставленным пресс-службой Рособрнадзора)