УДК 372.862

ДЕКОМПОЗИЦИЯ И СТРУКТУРИРОВАНИЕ АЛГОРИТМОВ И ПРОГРАММ С++ ПРИ ИЗУЧЕНИИ ИНФОРМАТИКИ В ВЫСШЕЙ ШКОЛЕ

Фокин Р.Р.

ФГКВОУ ВО «Военно-космическая академия имени А.Ф. Можайского», Санкт-Петербург, e-mail: vka@mil.ru

Темы, связанные с улучшением качества преподавания математики и информатики в современной высшей школе, неоднократно рассматривались автором. В данной статье речь идет о преподавании С++. Алгоритмизация и программирование имеют три фундаментальных принципа: ветвление, цикличность, структурирование алгоритмов и программ. При обучении структурирование алгоритмов рассматривается обычно последним, переходя далее к структурированию данных и объектно-ориентированным технологиям. Автор предлагает изучать ветвление и цикличность параллельно со структурированием алгоритма. Тогда сложный для понимания алгоритм может быть разбит на несколько простых. Это и есть декомпозиция. В результате студентам легче понять сложные ветвления и циклы после разбиения на простые. Еще одна идея автора состоит в том, что из знаний правил языка программирования вообще и С++ в частности, полученных на лекциях, никак не следуют навыки разработки алгоритмов и программ – их нужно формировать на практических занятиях, предшествующих лабораторным работам. Также автор предлагает максимально обучать современных студентов визуальному программированию и минимально – обычному классическому программированию. Большинство современных студентов относятся к правополушарному психологическому типу. У них образное мышление преобладает над понятийным.

Ключевые слова: С++, программирование, ветвления, структурирование, декомпозиция, образность обучения, асимметрия мозга

DECOMPOSITION AND STRUCTURING OF C++ ALGORITHMS AND PROGRAMS IN HIGHER SCHOOL COMPUTER SCIENCE

Fokin R.R.

Military Space Academy named after A.F. Mozhaiskiy, Saint-Petersburg, e-mail: rrfokin@yandex.ru

Topics related to improving the quality of teaching mathematics and computer science in modern higher education have been repeatedly considered by this author. This article is about teaching C++. Algorithmization and programming have 3 fundamental principles: branching, Cycling, and structuring algorithms and programs. In training, algorithm structuring is usually considered last, moving on to data structuring and object-oriented technologies. The author suggests studying branching and Cycling in parallel with structuring the algorithm. Then a complex algorithm can be divided into several simple ones. This is the decomposition. As a result, complex branches and loops after splitting into simple ones are easier for students to understand. Another idea of the author is that the knowledge of the rules of the programming language in General and C++ in particular, obtained at lectures, does not follow the skills of developing algorithms and programs – they need to be formed in practical classes preceding laboratory work. The author also suggests that modern students should be taught visual programming as much as possible and classical programming as little as possible. Most modern students belong to the right-hemisphere psychological type. For them, imaginative thinking will prevail over conceptual thinking.

Keywords: C++, programming, branching, structuring, decomposition, learning imagery, brain asymmetry

Актуальность рассматриваемой темы обусловлена большими трудностями обучения студентов программированию на языке С++ при изучении дисциплин области знаний «Информатика» в высшей школе. Знания и умения (компетенции), получаемые студентом при изучении С++, позволяют на следующем этапе легко освоить наиболее популярные языки современного профессионального программирования Java и C#. Изучение программирования на основе других языков делает перспективу освоения студентом компетенций профессионального программиста более отдаленной и проблемной.

Главная цель исследования автора статьи – выяснить причины отмеченного выше явления. Зная причины, можно предлагать пути решения соответствующих проблем. Эти причины многочисленны и разнообразны.

Материалы и методы исследования

В настоящее время указанное выше явление широко обсуждается вместе с другими проблемами обучения современному программированию. Данная статья является продолжением других наших статей. Также она развивает некоторые идеи автора и его коллег [1–3] по преподаванию математики и информатики. Для анализа используется опыт преподавания автором С++ в некоторых вузах Санкт-Петербурга. На основе анализа предлагаются некоторые новые методы и приемы обучения программированию на языке С++. Эти методы

и приемы сравниваются с теми, что используются традиционно. Результаты обучения обобщаются и делаются практические выводы. Используются как индуктивные, так и дедуктивные методы исследования.

Результаты исследования и их обсуждение

При обучении программированию на С++ традиционно студенты слушают лекции по языку С++ и выполняют лабораторные работы по вариантам с использованием какой-нибудь системы программирования, включающей С++. На лабораторной работе студент по индивидуальному заданию разрабатывает алгоритм в виде блок-схемы [4], а затем – программу на С++, которую и реализует на компьютере.

Первая идея автора состоит в том, что при этом студента не обучают практическим навыкам разработки соответствующих алгоритмов и написания программ. В принципе это должны быть практические занятия с выполнением заданий, аналогичных тем, что студент будет самостоятельно выполнять в ходе лабораторных работ. Помимо конспекта лекций или учебника по С++ студент должен также иметь учебное пособие – практикум с образцами выполнения заданий, аналогичных лабораторным работам.

Вторая идея автора – раннее использование в курсе С++ декомпозиции и структурирования алгоритмов (блок-схем) и программ С++. Это способствует улучшению их понимания студентом. Традиционно декомпозиция и структурирование изучаются в середине курса С++ после операторов ветвления и цикла. Автор предлагает это изучать почти с самого начала курса – параллельно с ветвлениями и циклами. В качестве доказательства, что это возможно, приводим фрагмент нашего практикума – образца выполнения задания на ветвления с использованием декомпозиции и структурирования.

Задание для изучения ветвлений

Ввести с клавиатуры значения *a*, *b*, *y*, *z* и последовательно вычислить:

1) $d = \begin{cases} z^2(y+1) & \text{при} \quad y < a^2 \\ z^2(y+2) & \text{при} \quad a^2 \le y \le b^2; \\ z^2(y+3) & \text{при} \quad y > b^2 \end{cases}$

2) $Z = a^2 - ab + b^2$;

3) $F = \sqrt{d^2 + Z^2}$.

Вывести на экран значение F.

Блок-схема

Единую блок-схему на одном рисунке нарисовать и понять было бы трудно, необходима ее декомпозиция – разбиение на кусочки (рис. 1–5). Внутри главной блок-схемы (рис. 1) мы видим вызовы блок-схем «Начальные действия», «Условные действия», «Конечные действия». Соответствующий значок [4] означает предопределенный процесс (подпрограмму, модуль). Блок-схема «Условные действия» (рис. 3) содержит вызов блок-схемы «Внутренние условные действия».

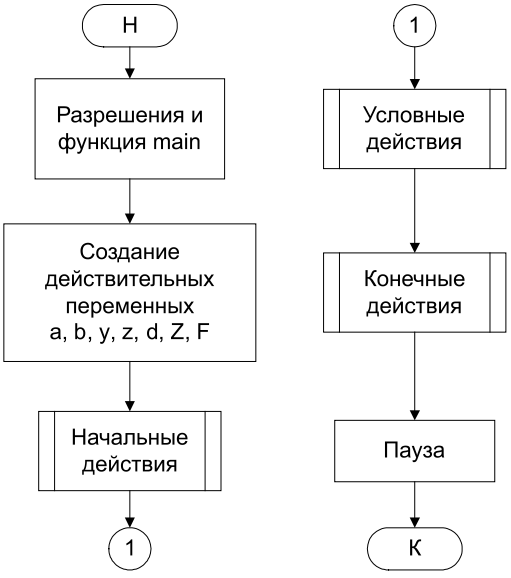

Рис. 1. Главная блок-схема

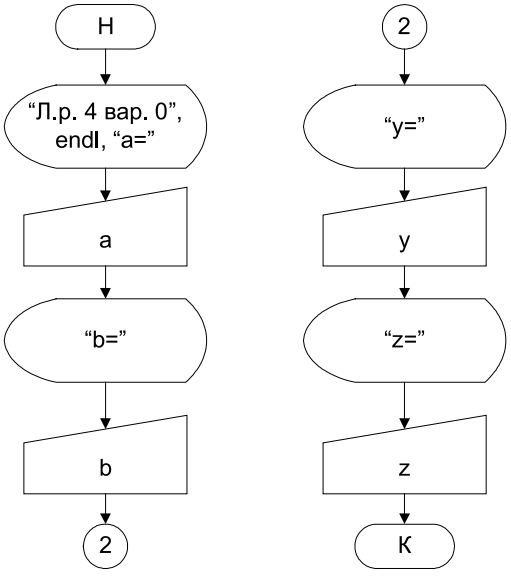

Рис. 2. Блок-схема «Начальные действия»

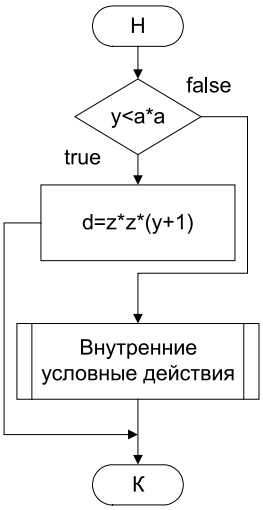

Рис. 3. Блок-схема «Условные действия»

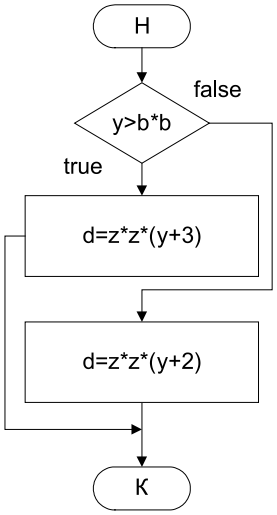

Рис. 4. Блок-схема «Внутренние условные действия»

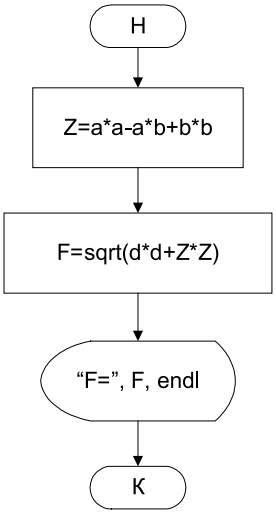

Рис. 5. Блок-схема «Конечные действия»

Конструкция программы

```
#include <iostream>        // разрешение ввода и вывода
#include <locale>          // разрешение национальных алфавитов
#include <cmath>           // разрешение математических функций
#include <string>          // разрешение  работы со строками
using namespace std;       // разрешение стандартного пространства имен
```

Создание глобальных переменных

Прототипы пользовательских функций

```
void main() // заголовок main
{ // между { и } – тело main
    setlocale(LC_ALL, "rus"); // разрешение русского алфавита при выводе
```

Операторы

```
}
```

Описание пользовательских функций

Здесь и далее непосредственно символы программы выделены *курсивом*. Это терминальные символы, если проводить аналогию с грамматиками Н. Хомского, а подчеркнутые элементы требуют дальнейшей конкретизации. Это нетерминальные символы.

Создание глобальных переменных

```
double a, b, y, z, d, Z, F; // создание ГЛОБАЛЬНЫХ действительных переменных
```

Будем использовать только глобальные переменные, чтобы отложить на будущее изучение локальных переменных, формальных и фактических параметров.

Прототипы пользовательских функций

```
void InitialActions();
void ConditionActions();
void InternalConditionActions();
void FinalActions();
```

Программу нужно разбить на отдельные подпрограммы, соответствующие блок-схемам на рис. 1–5. Подпрограммы в C++ называются **функциями**. Имена функций InitialActions, ConditionActions, InternalConditionActions, FinalActions соответствуют блок-схемам с именами «Начальные действия», «Условные действия», «Внутренние условные действия», «Конечные действия». Эти функции – пользовательские, поскольку их имена и соответствующие им алгоритмы придумывает пользователь (мы).

Операторы

```
InitialActions(); // вызов InitialActions
ConditionActions(); // вызов ConditionActions
FinalActions();// вызов FinalActions
system(«pause»); // пауза перед завершением программы
```

Это операторы функции **main**. Главной блок-схеме соответствует функция **main** – **главная** в переводе с английского. Запуск программы – это ее вызов. Остальные функции вызываются операторами вызова этих функций внутри программы.

Описания пользовательских функций

```
void InitialActions() // заголовок InitialActions
{ // между { и } – тело InitialActions
    cout << "Изучение ветвлений\n" << "Введите в следующем порядке\n"
    << "значения a, b, y, z:\n";
    cin >> a >> b >> y >> z; // ввод 4 чисел в столбик или в строчку
}
void ConditionActions() // заголовок ConditionActions
{ // между { и } – тело ConditionActions
    if (y<a*a) // если y<a*a
            d = z*z*(y+1); // вычисление d способом для y<a*a
    else // иначе
            InternalConditionActions(); // вызов InternalConditionActions
}
void InternalConditionActions() // заголовок InternalConditionActions
{ // между { и } - тело InternalConditionActions
    if (y>b*b) // если y>b*b
            d = z*z*(y+3); // вычисление d способом для y > b*b
    else // иначе
            d = z*z*(y+2); // вычисление d способом для a<= x < = b
}
void FinalActions() // заголовок FinalActions
{ // между { и } - тело FinalActions
    Z = a*a-a*b+b*b; // вычисление Z
    F = sqrt(d*d+Z*Z); // вычисление F
    cout << "F=" << F << endl; // вывод строки "F =", значения F, строки endl
}
```

Заметим, что описание пользовательской функции ConditionActions содержит оператор вызова пользовательской функции InternalConditionActions, а описание пользовательской функции FinalActions – вызов встроенной функции sqrt, она вычисляет квадратный корень.

Пример работы программы

Изучение ветвлений
Введите в следующем порядке
значения a, b, y, z:
2.5 3.4 4.3 5.2 Enter
F = 143.614
Для продолжения нажмите любую клавишу... Enter

Выделенный *курсивом* текст возникает на экране при запуске нашей программы, *подчеркнутым курсивом* выделено то, что печатает пользователь в диалоге с работающей программой, *Enter* означает нажатие пользователем соответствующей клавиши.

В вузах России обычно используются системы программирования семейства Microsoft Visual Studio, которые так называются, поскольку допускают визуальное программирование. В них встроены языки Visual C++, Visual C#, Visual Basic и другие. Однако визуальное программирование «почти везде» в вузах России почему-то «традиционно» не используется вовсе. Третья идея автора предлагает визуальное программирование как можно более широко использовать в обучении, а использование «традиционного» программирования при этом – минимизировать. Такая методика используется, например, в учебниках [5, 6] В.В. Зиборова.

Автор с успехом их использовал на занятиях со студентами.

Имеются другие статьи [1–3] автора и его коллег, связанные с проблемами изучения математики и программирования современными студентами. Там приводится статистика [2], свидетельствующая о резком росте доли правополушарных студентов за последние три десятка лет. Сейчас даже на математических и технических факультетах более половины студентов – правополушарные. У этих студентов [7], как правило, интуиция развита больше, чем логика, образное мышление преобладает над понятийным, действенное – над абстрактным. Отсюда актуальность применения образных технологий обучения вообще и визуального программирования в частности.

Выводы

Сложность изучения C++ можно существенно снизить: 1) если лекции по C++ и лабораторные работы дополнить обучением практическим навыкам разработки алгоритмов и программ; 2) благодаря раннему использованию методов декомпозиции и структурирования алгоритмов и программ параллельно с изучением ветвлений и циклов; 3) если при обучении широко использовать визуальное программирование.

Список литературы

1. Фокин Р.Р. Социальные, психологические и методические причины трудностей изучения математики и программирования современными студентами // Современные наукоемкие технологии. 2020. № 4. С. 138–142.

2. Фокин Р.Р. Некоторые психологические и статистические аспекты преподавания дисциплин из областей математики и информатики в современной высшей школе // Современные наукоемкие технологии. 2019. № 9. С. 175–179.

3. Абиссова М.А., Атоян А.А. Сервисы обучения RAD-программированию для активизации познавательной деятельности студентов при обучении информатике и математике // Письма в Эмиссия.Оффлайн 2013. № 12. [Электронный ресурс]. URL: www.emissia.org/offline/2013/2118.htm (дата обращения: 15.11.2020).

4. ГОСТ 19.701-90 (ИСО 5807-85) Единая система программной документации (ЕСПД). Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Сб. ГОСТов. М.: Стандартинформ, 2012. 23 с.

5. Зиборов В.В. MS Visual C++ 2010 в среде .NET. СПб.: Питер, 2012. 320 с.

6. Зиборов В.В. MS Visual C# 2010 на примерах. СПб.: БХВ-Петербург, 2011. 432 с.

7. Спрингер С., Дейч Г. Левый мозг. Правый мозг. М.: Книга по требованию, 2013. 254 с.