СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc170837307)

[1 ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ 6](#_Toc170837308)

[Практическая часть 9](#_Toc170837309)

# ВВЕДЕНИЕ

Целью учебной практики является закрепление пройденного материала теоретического курса по дисциплинам ОПОП, получение навыков практического решения прикладных задач, получение первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности. Для получения соответствующих навыков важно не только решение практических задач параллельно с получением и расширением теоретических знаний, но и умение решать практические задачи после освоения теоретических курсов. Такие задачи должны быть комплексными, т. е. для их решения нужно владеть знаниями нескольких дисциплин. Решение достаточного количества таких задач в ходе учебной практики позволит не только усовершенствовать навыки, полученные за период обучения, но и выявить слабые места, которым стоит уделить больше внимания в ходе дальнейшего обучения. Решение практических задач дает возможность понять необходимость вдумчивого и тщательного изучения теоретических основ профессии, что в свою очередь закладывает прочный фундамент для дальнейшего обучения.

# 1 ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ

Алгоритм – четкое предписание исполнителю выполнить определенную последовательность действий, направленных на достижение определённой цели.

От любой другой последовательности действий алгоритм отличают его свойства:

1. Дискретность – разбиение алгоритма на последовательность отдельных законченных действий, шагов. Каждый такой шаг должен быть закончен до выполнения следующего.
2. Точность – однозначность указаний.

Состояние объектов среды исполнителя однозначно определено на каждом шаге. На каждом шаге однозначно определён шаг, который нужно выполнить следующим. Таким образом, при применении алгоритма к одному набору входных данных на выходе каждый раз будет получен один и тот же результат.

1. Понятность – алгоритм должен быть изложен на языке, понятном для исполнителя; таким образом, каждый шаг алгоритма будет трактован однозначно, т. е. состоять из команд, входящих в систему команд исполнителя.
2. Конечность (результативность) – обязательное получение результата за конечное число шагов.

Работа алгоритма должна быть завершена за конечное число шагов в любом случае, даже если решение не найдено. Теоретические аспекты бесконечных алгоритмов в рамках учебной практики не рассматриваются.

1. Массовость – применение алгоритма к решению всего класса однотипных задач.

Способы представления алгоритма

Выделяются следующие формы записи алгоритмов:

* графическая запись (блок-схемы);
* на естественном языке (словесная запись, псевдокод);
* код на языке программирования;
* в виде математической формулы.

Алгоритм в виде блок-схемы представляет собой последовательность связанных между собой функциональных блоков, соответствующих шагам алгоритма. Блоки соединены между собой линиями, определяющими действие, которое должно быть выполнено следующим.

Представление алгоритма в виде блок-схемы строго формализовано. Инструкции к представлению алгоритма таким образом содержатся в ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.

Требования к форме и размерам блоков приведены в ГОСТ 19.003-80. Схемы алгоритмов и программ. Обозначения условные и графические.

При изображении блоков размер стороны a выбирается из ряда 15, 10, 20 мм, допускается увеличение значения параметра a на число, кратное 5. Сторона b соотносится со стороной а таким образом, что b = 1,5a.

Основные блоки представлены в прил. 3.

Любой алгоритм может быть представлен с использованием трех основных алгоритмических структур: следования, ветвления и цикла. Алгоритмы, содержащие несколько алгоритмических структур, называют комбинированными.

Следование – алгоритмическая структура, в которой все команды выполняются последовательно, одна за другой. Алгоритмы, в которых используется только алгоритмическая конструкция «следование», называются линейными.

Ветвление – алгоритмическая структура, в которой в зависимости от значения логического выражения будет выполнено либо одно, либо другое действие.

Существует две формы ветвления: полная и неполная. В полной форме ветвление содержит два действия (последовательности команд), одно из этих действий будет выполнено при значении условия «истина», а второе – при значении условия «ложь». В неполной форме ветвление содержит только одно действие или последовательность команд, которые будут выполнены при значении условия «истина». В прил. 4 приведены структурные схемы полной и неполной форм ветвления.

Алгоритмы, в основе которых лежит алгоритмическая конструкция «ветвление», называют разветвляющимися.

Повторение (цикл) – алгоритмическая конструкция, с помощью которой определенная последовательность действий выполнится необходимое число раз. Алгоритмы, основой которых служит конструкция «повторение», называют циклическими, или циклом. Под телом цикла понимают действия, многократно повторяющиеся в процессе выполнения цикла.

Выделяют два типа циклов (по взаимному расположению тела цикла и условиям продолжения):

1. цикл с постусловием;
2. цикл с предусловием.

При использовании конструкции «цикл с предусловием» проверка условия происходит до выполнения действий тела цикла. Возможна ситуация, в которой тело цикла не выполнится ни разу.

При использовании конструкции «цикл с постусловием» тело цикла будет выполнено как минимум один раз, так как проверка условия происходит после выполнения тела цикла и в зависимости от результата этой проверки будет осуществлён выход из цикла или переход на следующую итерацию.

Для успешной организации алгоритмической конструкции «повторение» следует до входа в цикл задать начальные значения переменных, используемых в цикле. В теле цикла необходимо предусмотреть изменение переменных, анализируемых в условии продолжения цикла.

При решении конкретных задач алгоритм может содержать более одной конструкции «повторение». В зависимости от их взаимного расположения говорят о вложенных или последовательных циклах.

Если один цикл является частью тела другого цикла, то первый цикл называют вложенным, второй – внешним.

Перед решением задачи и написанием кода на языке программирования необходимо представить графическое решение в виде блок-схемы.

## Практическая часть

Задание: даны длины ребер а, b, c прямоугольного параллелепипеда. Найдите его объем и площадь поверхности.

Листинг приложения:

int SurfaceArea(int a, int b, int c) {

int S;

S = 2 \* (a \* b + b \* c + a \* c);

return S;

}

int VolumeParallelepiped(int a, int b, int c) {

int V;

V = a \* b \* c;

return V;

}

void TaskIndivid13\_block1() {

int a, b, c, \_v, \_s;

cout << "Введите длину ребра a: ";

cin >> a;

cout << "Введите длину ребра b: ";

cin >> b;

cout << "Введите длину ребра c: ";

cin >> c;

\_s = SurfaceArea(a, b, c);

\_v = VolumeParallelepiped(a, b, c);

cout << "Площадь поверхности = " << \_s << endl << "Объем параллепипеда = " << \_v << endl;

}

Результат выполнения кода:

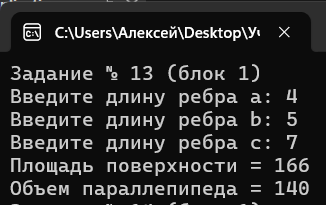


Рисунок 1 – Задание 1 блока

# 2 БАЗОВЫЕ ТИПЫ ЯЗЫКА С++

Перед объявлением переменной или константы необходимо определить, какого она будет типа. Для этого нужно понимать цели, для которых она будет использована, какие значения она может принимать.

Тип данных определяет количество выделяемой памяти под переменную или константу, правила хранения данных этого типа, допустимые операции для данных этого типа.

Количество памяти, выделяемое под переменную, и правила хранения данных определяют диапазон значений данных конкретного типа данных.

Типы данных языка С++ можно разделить на две группы: базовые и пользовательские.

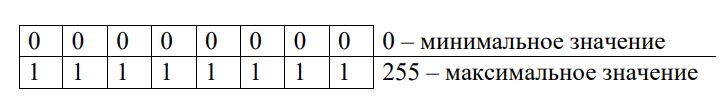
К базовым типам языка относят: void, int, char, float double, bool (прил. 5).

К целым типам int, short, long, char применимы спецификаторы signed (знаковый), unsigned (беззнаковый).

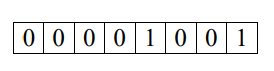
Использование спецификаторов short и long перед типом данных int ведёт к уменьшению или увеличению количества выделяемой памяти.

Целые беззнаковые числа хранятся в прямом двоичном коде. Диапазон таких данных – от нуля до числа, двоичное представление которого состоит из ряда единиц, длина ряда – количество бит, отводимых под переменную соответствующего типа.

Таким образом, диапазон беззнаковых целых типов данных будет определён следующим образом: 0 – 2 n – 1, где n – число разрядов, отводимых под число:



Знаковые целые числа хранятся в дополнительном коде. Старший бит хранит знак: единица – минус, ноль – плюс. Незначащие разряды, следующие за знаковым, заполняются нулями. Например, если под переменную отведено восемь разрядов, то двоичное число 1001 в памяти будет представлено так:



Дополнительный код отрицательного числа m равен 2 k – |m|, где k – количество разрядов в ячейке. Для получения дополнительного кода отрицательного числа необходимо: − модуль отрицательного числа представить в прямом коде; − инвертировать значение всех бит числа, т. е. все нули заменить на единицы, а единицы – на нули; − к полученному обратному коду прибавить единицу. Получим 8-разрядный дополнительный код числа –48:

00110000 – |-48| = 48 – в прямом коде;

11001111 – |-48| – в обратном коде;

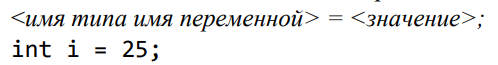
11010000 – -48 – в дополнительном коде.

Диапазон целых знаковых чисел – от –2 n – 1 до 2n – 1 – 1, где n – число разрядов, отводимых под переменную. Для хранения вещественных типов данных в С++ определены типы float, double, long double, отличающиеся количеством выделяемой памяти. При хранении этих типов данных часть разрядов отводится для записи порядка числа, остальные разряды – для записи мантиссы. Логические переменные могут принимать значения true и false. Тип таких переменных – bool. Тип данных void не имеет значений, он нужен в ряде ситуаций: например, когда правила языка требуют указания типа данных, но необходимости в таком типе нет (если отсутствует возвращаемое значение функции). Переменную типа void объявить нельзя. При работе с символами используют типы char и wchar\_t. Для хранения символов набора из 256 символов ASCII применяют тип char. Для работы с символами, код которых занимает более одного байта, используют тип wchar\_t. Для объявления переменной необходимо указать её тип и имя, завершив объявление точкой с запятой:



int i;

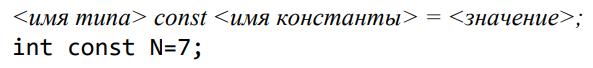
Можно совместить объявление переменной с инициализацией:



При необходимости объявить несколько однотипных переменных их разделяют запятой:

int i, j;

При объявлении константы перед её именем необходимо указать ключевое слово const. Значение константы нужно задать при объявлении:

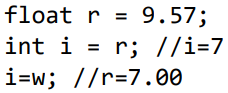


Правила именования в языке С++

Любой идентификатор (имя) должен начинаться с латинской буквы. Кроме того, имя может содержать цифры и символ подчёркивания. Язык программирования С++ регистрочувствительный. Это означает, что int z; и int Z; – объявление двух разных переменных. Имена переменных должны нести смысловую нагрузку, т. е. из названия переменной должно следовать, для каких целей она используется.

Место описания идентификатора задаёт его область действия. Переменная, описанная внутри блока, будет локальной, т. е. видна только в этом блоке от точки объявления и ниже. Блоком называют часть кода, ограниченную фигурными скобками, например, тело цикла. Переменную, объявленную вне блока, называют глобальной. Обращение к такой переменной возможно в модуле, в котором она объявлена, от точки объявления и ниже. Если имя локальной и глобальной переменных совпадают, то локальная перекрывает глобальную, т. е. обращение внутри блока будет вестись к локальной переменной.

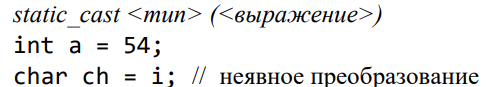
При использовании в выражениях разнотипных операндов применяют явное или неявное приведение типов. При неявном приведении типа в операции присваивания тип правого операнда приводится к типу левого. При приведении вещественного типа к целочисленному дробная часть отбрасывается. При приведении целого типа к вещественному добавляется нулевая дробная часть. Например:



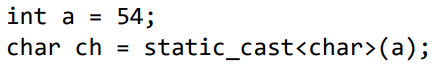
При использовании явного приведения типа перед операндом необходимо указать тип данных, к которому приводят исходный тип операнда. При этом важно понимать, что приведение типа осуществляется только в указанной точке, сама переменная, её значение и способ хранения данных при этом не изменяются. Явное приведение возможно двумя способами. Если есть уверенность, что приведение пройдёт без потери данных, используют подход языка С. Перед идентификатором, тип которого приводится, указывают нужный тип в круглых скобках. Например:



Если пользователь уверен в том, что переполнение не произойдёт, то применяется оператор static\_cast:



Такой подход приведет к предупреждающему сообщению во время компиляции. Для того чтобы избежать этого, лучше сделать так:



Оператор % возвращает остаток от деления, оператор %= присваивает левому операнду остаток от деления левого операнда на правый. Соответственно, в результате выполнения выражения m %= n; целочисленная переменная m примет значение частного текущего значения этой переменной и n.

## Практическая часть

Задание: напишите программу для перевода сантиметров в дюймы. Для справки: в одном дюйме 2,54 сантиметра.

# 3 ОПЕРАТОРЫ ВЕТВЛЕНИЯ ЯЗЫКА С++

# 4 ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ ЯЗЫКА С++

# 5 УКАЗАТЕЛИ И ССЫЛКИ

# 6 МАССИВЫ

# 7 ПОЛЬЗОВАТЕЛЬСКИЕ ТИПЫ ДАННЫХ. СТРУКТУРЫ

# 8 ФУНКЦИИ

# 9 ЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

# 10 ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON

# 11 ОПЕРАТОР ВЕТВЛЕНИЯ В ЯЗЫКЕ PYTHON

# 12 ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ ЯЗЫКА PYTHON

# 13 СТРОКИ

# 14 МАССИВЫ. СПИСКИ

# 15 ФАЙЛЫ

# 16 ПРОЦЕДУРЫ. ФУНКЦИИ