# Линейные алгоритмы

## Переменные в C++

Для хранения данных в программе в языке C++ используются переменные. Фактически переменная представляет именованный участок памяти. Переменная имеет тип, имя и значение. Тип определяет, какие именно данные может хранить переменная.  
  
Перед использованием любую переменную надо определить. Синтаксис определения переменной выглядит следующим образом:

type name;

Где type — это тип данных переменной, а name — это имя переменной.

Не рекомендуются следующие именования:  
● имена, которые начинаются с двух подчеркиваний  
● имена, которые начинаются с подчеркивания, за которым идет заглавный алфавитный символ  
● имена в глобальной области (вне функции), которые начинаются с подчеркивания  
  
Имена, начинающиеся с двух подчеркиваний, зарезервированы для использования реализациями компиляторов и стандартной библиотеки. Они часто используются для внутренних механизмов и макросов компилятора. Пример \_\_variable  
  
Имена которые начинаются с подчеркивания, за которым идет заглавная буква также зарезервированы для реализации библиотек и компиляторов. Это связано с тем, что стандартные библиотеки и компиляторы могут использовать такие имена для своих собственных функций и типов данных. Пример \_Variable.  
  
Использование переменных вне функции может привести к непредсказуемому поведению программы, поскольку они могут конфликтовать с именами, используемыми в стандартной библиотеке или компилятором.

Также нельзя объявить больше одной переменной с одним и тем же именем (Age и age 2 разные переменные).

Примеры определения переменных различных типов:

int age = 25; // Целочисленный тип  
float height = 1.75; // Тип с плавающей точкой  
char initial = 'A'; // Символьный тип  
bool isStudent = true; // Логический тип  
std::string name = "John";// Строковый тип

Ограничения на имена переменных:  
● Имена переменных не могут начинаться с цифры.  
● Имена переменных могут содержать только буквы, цифры и символ подчеркивания.  
● Имена переменных чувствительны к регистру (например, Age и age — разные переменные).  
  
Примеры корректных и некорректных имен переменных:

int \_var; // корректно  
int var123; // корректно  
int 1var; // некорректно  
int var@name; // некорректно  
int Var; // корректно (но отличается от var)

## Типы данных

Каждая переменная имеет определенный тип. И этот тип определяет, какие значения может иметь переменная, какие операции с ней можно производить и сколько байт в памяти она будет занимать.

Целочисленные типы (Integral Types)  
bool: Логический тип, представляющий значения true или false.  
char: Символьный тип, представляющий одиночный символ.  
wchar\_t: Широкий символьный тип, используемый для представления широких символов (обычно используется для Юникода).  
char16\_t: Символьный тип для 16-битных символов Юникода.  
char32\_t: Символьный тип для 32-битных символов Юникода.  
Знаковые целые числа (Signed Integer Types)  
signed char  
short int или short  
int  
long int или long  
long long int или long long  
Беззнаковые целые числа (Unsigned Integer Types)  
unsigned char  
unsigned short int или unsigned short  
unsigned int  
unsigned long int или unsigned long  
unsigned long long int или unsigned long long  
Типы с плавающей точкой (Floating-Point Types)  
float  
double  
long double

## Структура программы на C++

Программа на языке C++ состоит из набора функций, из которых одна, main, является точкой входа для выполнения программы. Важно соблюдать порядок и синтаксис написания программы. Общая структура программы на C++ выглядит следующим образом:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 // тело функции  
 return 0;  
}

Где:  
● #include <iostream> — директива препроцессора, включающая библиотеку для ввода-вывода.  
● using namespace std; — используется для упрощения доступа к стандартному пространству имен.  
● int main() — главная функция, с которой начинается выполнение программы.  
● return 0; — оператор, который завершает выполнение функции main и возвращает значение 0.

Пример программы, которая выводит текст на экран:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 cout << "Hello, World!" << endl;  
 return 0;  
}

В C++ можно объявлять и определять собственные функции, которые могут быть вызваны из функции main или других функций. Пример функции для сложения двух чисел:

#include <iostream>  
  
using namespace std;  
  
// Функция для сложения двух чисел  
int add(int a, int b) {  
 return a + b;  
}  
  
int main() {  
 int result = add(3, 4);  
 cout << "Sum: " << result << endl;  
 return 0;  
}

## Операторы ввода и вывода в C++

Для ввода и вывода данных в C++ используются операторы cin и cout из библиотеки iostream. Оператор cout используется для вывода данных на экран, а оператор cin — для ввода данных с клавиатуры.

Пример использования оператора cout для вывода данных на экран:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 cout << "Введите ваше имя: ";  
 return 0;  
}

Пример использования оператора cin для ввода данных с клавиатуры:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 string name;  
 cout << "Введите ваше имя: ";  
 cin >> name;  
 cout << "Привет, " << name << "!" << endl;  
 return 0;  
}

Оператор cin используется для ввода данных с клавиатуры, а оператор cout для вывода данных на экран. Пример программы, которая запрашивает у пользователя его имя и возраст, а затем выводит их на экран:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 string name;  
 int age;  
 cout << "Введите ваше имя: ";  
 cin >> name;  
 cout << "Введите ваш возраст: ";  
 cin >> age;  
 cout << "Ваше имя: " << name << ", ваш возраст: " << age << " лет." << endl;  
 return 0;  
}

## Математические операции в C++

C++ поддерживает различные математические операции, такие как сложение, вычитание, умножение, деление и взятие остатка. Эти операции можно выполнять с использованием операторов.

Примеры математических операций:  
● Сложение (+): a + b  
● Вычитание (-): a - b  
● Умножение (\*): a \* b  
● Деление (/): a / b  
● Взятие остатка (%): a % b  
  
Пример кода:

int a = 10;  
int b = 3;  
int sum = a + b; // Сложение  
int diff = a - b; // Вычитание  
int prod = a \* b; // Умножение  
int quot = a / b; // Деление  
int rem = a % b; // Взятие остатка

Кроме того, существуют операции увеличения (инкремент) и уменьшения (декремент):  
● Инкремент (увеличение на 1): ++a или a++  
● Декремент (уменьшение на 1): --a или a--  
  
Пример кода:

int a = 5;  
a++; // Постфиксный инкремент, a становится 6  
++a; // Префиксный инкремент, a становится 7  
  
a--; // Постфиксный декремент, a становится 6  
--a; // Префиксный декремент, a становится 5

Также поддерживаются операции присваивания с арифметическими операторами:  
● Присваивание с сложением: a += b  
● Присваивание с вычитанием: a -= b  
● Присваивание с умножением: a \*= b  
● Присваивание с делением: a /= b  
● Присваивание с остатком: a %= b  
  
Пример кода:

int a = 10;  
a += 5; // a становится 15  
a -= 3; // a становится 12  
a \*= 2; // a становится 24  
a /= 4; // a становится 6  
a %= 5; // a становится 1

## Математические функции в C++

Для выполнения более сложных математических операций в C++ существует стандартная библиотека cmath, которая предоставляет широкий спектр математических функций.

Некоторые основные математические функции из библиотеки cmath:  
● sqrt(x) — вычисляет квадратный корень числа x.  
● pow(x, y) — возводит x в степень y.  
● sin(x) — вычисляет синус угла x (в радианах).  
● cos(x) — вычисляет косинус угла x (в радианах).  
● tan(x) — вычисляет тангенс угла x (в радианах).  
● abs(x) — вычисляет абсолютное значение числа x.  
● log(x) — вычисляет натуральный логарифм числа x.  
● log10(x) — вычисляет десятичный логарифм числа x.  
● exp(x) — вычисляет e в степени x.  
● fmod(x, y) — вычисляет остаток от деления x на y.  
  
Пример кода, демонстрирующего использование математических функций:

#include <iostream>  
#include <cmath>  
  
using namespace std;  
  
int main() {  
 double x = 9.0;  
 double y = 2.0;  
 double sqrt\_x = sqrt(x); // Квадратный корень  
 double power = pow(x, y); // Возведение в степень  
 double sine = sin(x); // Синус  
 double cosine = cos(x); // Косинус  
 double tangent = tan(x); // Тангенс  
 double absolute = abs(-x);// Абсолютное значение  
 double logarithm = log(x); // Натуральный логарифм  
 double log10\_val = log10(x); // Десятичный логарифм  
 double exponent = exp(y); // Экспонента  
 double modulus = fmod(x, y); // Остаток от деления  
  
 cout << "sqrt(x): " << sqrt\_x << endl;  
 cout << "pow(x, y): " << power << endl;  
 cout << "sin(x): " << sine << endl;  
 cout << "cos(x): " << cosine << endl;  
 cout << "tan(x): " << tangent << endl;  
 cout << "abs(-x): " << absolute << endl;  
 cout << "log(x): " << logarithm << endl;  
 cout << "log10(x): " << log10\_val << endl;  
 cout << "exp(y): " << exponent << endl;  
 cout << "fmod(x, y): " << modulus << endl;  
  
 return 0;  
}

Для использования математических функций, необходимо включить заголовочный файл cmath в начале программы с помощью директивы #include <cmath>. В этом примере демонстрируются различные математические операции, такие как вычисление квадратного корня, возведение в степень, вычисление синуса, косинуса и тангенса, нахождение абсолютного значения, логарифмов и экспоненты, а также вычисление остатка от деления.

# Условные алгоритмы

## Условные операторы в C++

Условные операторы используются для выполнения различных операций в зависимости от условий. Основные условные операторы в C++ это if, if-else и switch.

Оператор if:  
Оператор if выполняет блок кода, если условие истинно.

if (condition) {  
 // код, выполняемый если условие истинно  
}

Оператор if-else:  
Оператор if-else выполняет один блок кода, если условие истинно, и другой блок кода, если условие ложно.

if (condition) {  
 // код, выполняемый если условие истинно  
} else {  
 // код, выполняемый если условие ложно  
}

Оператор else if:  
Оператор else if используется для проверки нескольких условий.

if (condition1) {  
 // код, выполняемый если condition1 истинно  
} else if (condition2) {  
 // код, выполняемый если condition2 истинно  
} else {  
 // код, выполняемый если ни одно из условий не истинно  
}

Оператор switch:  
Оператор switch используется для выбора одного из множества возможных блоков кода для выполнения.

switch (variable) {  
 case value1:  
 // код, выполняемый если variable равно value1  
 break;  
 case value2:  
 // код, выполняемый если variable равно value2  
 break;  
 default:  
 // код, выполняемый если ни одно из условий не выполнено  
}

Примеры использования условных операторов в C++:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int number = 10;  
  
 // Пример if-else  
 if (number > 0) {  
 cout << "Число положительное." << endl;  
 } else {  
 cout << "Число отрицательное или равно нулю." << endl;  
 }  
  
 // Пример else if  
 if (number > 0) {  
 cout << "Число положительное." << endl;  
 } else if (number < 0) {  
 cout << "Число отрицательное." << endl;  
 } else {  
 cout << "Число равно нулю." << endl;  
 }  
  
 // Пример switch  
 switch (number) {  
 case 1:  
 cout << "Число равно 1." << endl;  
 break;  
 case 10:  
 cout << "Число равно 10." << endl;  
 break;  
 default:  
 cout << "Число не равно 1 и не равно 10." << endl;  
 }  
  
 return 0;  
}

## Логические операторы в C++

Логические операторы используются для выполнения логических операций над булевыми значениями. Основные логические операторы включают AND (&&), OR (||) и NOT (!).

Оператор AND (&&):  
Возвращает true, если оба операнда истинны.

if (condition1 && condition2) {  
 // код, выполняемый если оба условия истинны  
}

Оператор OR (||):  
Возвращает true, если хотя бы один из операндов истинный.

if (condition1 || condition2) {  
 // код, выполняемый если хотя бы одно из условий истинно  
}

Оператор NOT (!):  
Инвертирует логическое значение операнда.

if (!condition) {  
 // код, выполняемый если условие ложно  
}

Примеры использования логических операторов в C++:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 bool a = true;  
 bool b = false;  
  
 if (a && b) {  
 cout << "Оба значения истинны." << endl;  
 }  
  
 if (a || b) {  
 cout << "Хотя бы одно значение истинно." << endl;  
 }  
  
 if (!b) {  
 cout << "b ложно." << endl;  
 }  
  
 return 0;  
}

# Циклические алгоритмы

## Циклы в C++

Циклы используются для многократного выполнения блока кода. Основные типы циклов в C++ это for, while и do-while.

Цикл for:  
Используется для выполнения блока кода определенное количество раз.

for (initialization; condition; increment) {  
 // код, выполняемый на каждой итерации  
}

Пример цикла for:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 for (int i = 0; i < 5; i++) {  
 cout << "Итерация " << i << endl;  
 }  
 return 0;  
}

Цикл while:  
Используется для выполнения блока кода, пока условие истинно.

while (condition) {  
 // код, выполняемый пока условие истинно  
}

Пример цикла while:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int i = 0;  
 while (i < 5) {  
 cout << "Итерация " << i << endl;  
 i++;  
 }  
 return 0;  
}

Цикл do-while:  
Используется для выполнения блока кода хотя бы один раз, а затем продолжает выполнение, пока условие истинно.

do {  
 // код, выполняемый хотя бы один раз и затем пока условие истинно  
} while (condition);

Пример цикла do-while:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int i = 0;  
 do {  
 cout << "Итерация " << i << endl;  
 i++;  
 } while (i < 5);  
 return 0;  
}

## Вложенные циклы

Вложенные циклы позволяют использовать один цикл внутри другого. Это полезно для работы с многомерными массивами и таблицами данных.

Пример вложенных циклов:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 for (int i = 0; i < 3; i++) {  
 for (int j = 0; j < 3; j++) {  
 cout << "i = " << i << ", j = " << j << endl;  
 }  
 }  
 return 0;  
}

# Одномерные массивы

## Одномерные массивы в C++

Массивы в C++ используются для хранения набора значений одного типа в одной переменной. Одномерные массивы являются наиболее простой формой массивов и представляют собой линейную последовательность элементов.

Определение массива:  
Массив определяется с указанием типа элементов и их количества.

type arrayName[arraySize];

Пример определения массива:

int numbers[5]; // Массив из 5 целых чисел

Инициализация массива:  
Массив можно инициализировать при определении.

int numbers[5] = {1, 2, 3, 4, 5};

Доступ к элементам массива:  
Для доступа к элементам массива используется индекс. Индексы начинаются с нуля.

int firstNumber = numbers[0]; // Доступ к первому элементу  
numbers[1] = 10; // Изменение значения второго элемента

Пример программы, использующей массив:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int numbers[5] = {1, 2, 3, 4, 5};  
  
 for (int i = 0; i < 5; i++) {  
 cout << "Element " << i << ": " << numbers[i] << endl;  
 }  
  
 return 0;  
}

## Операции с массивами

Массивы позволяют выполнять различные операции, такие как перебор элементов, поиск, сортировка и т.д.

Пример перебора массива с использованием цикла for:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int numbers[5] = {1, 2, 3, 4, 5};  
  
 for (int i = 0; i < 5; i++) {  
 cout << "Element " << i << ": " << numbers[i] << endl;  
 }  
  
 return 0;  
}

Пример поиска элемента в массиве:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int numbers[5] = {1, 2, 3, 4, 5};  
 int key = 3;  
 bool found = false;  
  
 for (int i = 0; i < 5; i++) {  
 if (numbers[i] == key) {  
 found = true;  
 break;  
 }  
 }  
  
 if (found) {  
 cout << "Элемент найден." << endl;  
 } else {  
 cout << "Элемент не найден." << endl;  
 }  
  
 return 0;  
}

# Двумерные массивы

## Двумерные массивы в C++

Двумерные массивы представляют собой массивы массивов и используются для хранения данных в табличной форме. Каждый элемент двумерного массива также является массивом.

Определение двумерного массива:  
Двумерный массив определяется с указанием типа элементов, количества строк и столбцов.

type arrayName[rows][cols];

Пример определения двумерного массива:

int matrix[3][4]; // Двумерный массив 3x4

Инициализация двумерного массива:  
Двумерный массив можно инициализировать при определении.

int matrix[3][4] = {  
 {1, 2, 3, 4},  
 {5, 6, 7, 8},  
 {9, 10, 11, 12}  
};

Доступ к элементам двумерного массива:  
Для доступа к элементам двумерного массива используются два индекса: индекс строки и индекс столбца. Индексы начинаются с нуля.

int firstElement = matrix[0][0]; // Доступ к первому элементу  
matrix[1][2] = 10; // Изменение значения элемента во второй строке и третьем столбце

Пример программы, использующей двумерный массив:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int matrix[3][4] = {  
 {1, 2, 3, 4},  
 {5, 6, 7, 8},  
 {9, 10, 11, 12}  
 };  
  
 for (int i = 0; i < 3; i++) {  
 for (int j = 0; j < 4; j++) {  
 cout << "Element at (" << i << ", " << j << "): " << matrix[i][j] << endl;  
 }  
 }  
  
 return 0;  
}

## Операции с двумерными массивами

С двумерными массивами можно выполнять различные операции, такие как перебор элементов, поиск, сортировка и т.д.

Пример перебора двумерного массива с использованием вложенных циклов:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int matrix[3][4] = {  
 {1, 2, 3, 4},  
 {5, 6, 7, 8},  
 {9, 10, 11, 12}  
 };  
  
 for (int i = 0; i < 3; i++) {  
 for (int j = 0; j < 4; j++) {  
 cout << "Element at (" << i << ", " << j << "): " << matrix[i][j] << endl;  
 }  
 }  
  
 return 0;  
}

Пример поиска элемента в двумерном массиве:

#include <iostream>  
  
using namespace std;  
  
int main() {  
 int matrix[3][4] = {  
 {1, 2, 3, 4},  
 {5, 6, 7, 8},  
 {9, 10, 11, 12}  
 };  
 int key = 7;  
 bool found = false;  
  
 for (int i = 0; i < 3; i++) {  
 for (int j = 0; j < 4; j++) {  
 if (matrix[i][j] == key) {  
 found = true;  
 break;  
 }  
 }  
 }  
  
 if (found) {  
 cout << "Элемент найден." << endl;  
 } else {  
 cout << "Элемент не найден." << endl;  
 }  
  
 return 0;  
}