**Основы программирования**

**1. Базовые типы данных: описание, инициализация переменных. Правила записи констант. Что определяет тип данного.**

**Базовые типы данных** — это фундаментальные типы, встроенные в язык программирования и служащие основой для построения более сложных типов данных.

Базовые типы данных:

**Целочисленные**: int, short, long, byte — для хранения целых чисел различного размера и диапазона

**С плавающей точкой**: float, double — для хранения вещественных чисел с разной точностью

**Символьный**: char — для хранения одного символа (обычно 1 байт)

**Логический**: bool — для хранения истинностных значений (true/false)

Инициализация переменных:

int a = 5; // Обычная инициализация  
int b(5); // Конструкторная инициализация  
int c{5}; // Инициализация списком (C++11)  
int d; // Без инициализации (неопределенное значение)

**Правила записи констант:**

**Целочисленные**: 123, 0xFF (шестнадцатеричная), 075 (восьмеричная), 0b1101 (двоичная в C++14)

**С плавающей точкой**: 3.14, 1.23e-4 (научная запись), 1.0f (явное указание типа float)

**Символьные**: ‘A’, ‘’, ‘,’\0’ (специальные символы)

**Строковые**: “Hello”, R”(C:Files)” (сырые строки в C++11)

**Логические**: true, false

**Что определяет тип данного**:

Размер в памяти (количество занимаемых байтов)

Диапазон возможных значений

Допустимые операции над данными этого типа

Формат представления в памяти

Правила преобразования типов

Поведение операторов с данным типом

**2. Операторы ветвления: правила записи и выполнения. Примеры.**

**Операторы ветвления** позволяют выполнять разные участки кода в зависимости от условий, управляя потоком выполнения программы.

**if-else:**

if (условие) {  
 // Код, если условие истинно  
} else if (другое\_условие) {  
 // Код, если первое условие ложно, а второе истинно  
} else {  
 // Код, если все условия ложны  
}

**Правила выполнения:** - Вычисляется условие (должно приводиться к типу bool) - Если условие истинно, выполняется блок после if - Иначе проверяются условия в блоках else if (если есть) - Если все условия ложны, выполняется блок else (если есть)

**Тернарный оператор:**

результат = условие ? значение\_если\_истина : значение\_если\_ложь;

Это сокращенная форма if-else для присваивания значений.

**switch-case:**

switch (выражение) {  
 case значение1:  
 // Код для значения1  
 break;  
 case значение2:  
 // Код для значения2  
 break;  
 default:  
 // Код по умолчанию  
}

**Правила выполнения switch:** - Вычисляется выражение (целочисленный тип или перечисление) - Управление передается соответствующему case - Выполнение продолжается до break или конца switch - Если нет соответствующего case, выполняется default (если есть)

3**. Операторы цикла: правила записи и выполнения. Примеры.**

**Операторы цикла** предназначены для многократного выполнения блока кода, пока выполняется определенное условие.

**for:**

for (инициализация; условие; инкремент) {  
 // Тело цикла  
}

**Правила выполнения:** - Выполняется инициализация (один раз) - Проверяется условие, если ложно — цикл завершается - Выполняется тело цикла - Выполняется инкремент - Возврат к проверке условия

**while:**

while (условие) {  
 // Тело цикла  
}

**Правила выполнения:** - Проверяется условие, если ложно — цикл завершается - Выполняется тело цикла - Возврат к проверке условия

**do-while:**

do {  
 // Тело цикла  
} while (условие);

**Правила выполнения:** - Выполняется тело цикла - Проверяется условие, если истинно — возврат к выполнению тела - Если условие ложно — цикл завершается

**Особенность do-while:** тело цикла выполняется хотя бы один раз.

**4. Функции в программировании. Определение. Параметры. Область применения.**

**Функция** — это именованный блок кода, который выполняет определенную задачу и может быть вызван многократно из разных частей программы.

**Определение функции:**

тип\_возврата имя\_функции(параметры) {  
 // Тело функции  
 return значение;  
}

**Компоненты функции:** - Тип возвращаемого значения (void для функций без возврата) - Имя функции - Список параметров (может быть пустым) - Тело функции (блок кода) - Оператор return (кроме функций типа void)

**Типы параметров:**

**Формальные параметры** — переменные, объявленные в заголовке функции

**Фактические параметры** — значения, передаваемые при вызове функции

**Передача по значению** — функция получает копию переменной

**Передача по ссылке** — функция работает с оригинальной переменной

**Параметры по умолчанию** — используются, если аргумент не указан

**Область применения:**

Модульность и структурирование кода

Уменьшение дублирования кода (повторное использование)

Абстракция и сокрытие деталей реализации

Облегчение отладки (изолированное тестирование)

Рекурсивные алгоритмы

Создание библиотек и API

**5. Массивы, многомерные массивы: описание, инициализация, обращение к массиву.**

**Массив** — структура данных, представляющая собой последовательность элементов одного типа, расположенных в памяти непрерывно.

**Основные характеристики массивов:**

Все элементы одного типа

Фиксированный размер (определяется при создании)

Доступ к элементам по индексу

Непрерывное размещение в памяти

Индексация обычно начинается с 0

**Одномерные массивы:**

int arr[5]; // Без инициализации  
int arr[5] = {1, 2, 3, 4, 5}; // С полной инициализацией  
int arr[5] = {1, 2}; // Частичная инициализация (остальные элементы = 0)  
int arr[] = {1, 2, 3}; // Размер определяется автоматически

**Обращение к элементам:**

int x = arr[0]; // Чтение первого элемента  
arr[1] = 10; // Запись во второй элемент

**Многомерные массивы:**

int matrix[3][4]; // Двумерный массив 3x4  
int matrix[2][3] = {{1,2,3},{4,5,6}}; // С инициализацией

**Обращение к элементам двумерного массива:**

int x = matrix[1][2]; // Элемент во второй строке, третьем столбце  
matrix[0][0] = 10; // Запись в элемент первой строки, первого столбца

**6. Принципы модульности в программировании.**

**Модульность** — принцип организации программы как совокупности отдельных независимых модулей, каждый из которых выполняет определенную функцию и имеет четко определенный интерфейс.

**Основные принципы:**

**Разделение ответственности**:

Каждый модуль должен отвечать за одну конкретную функцию

Модуль имеет четко определенное назначение

**Инкапсуляция**:

Скрытие внутренних деталей реализации

Предоставление чистого интерфейса для взаимодействия

**Слабая связанность (низкая связность)**:

Минимизация зависимостей между модулями

Изменения в одном модуле не должны влиять на другие

**Высокая сцепленность (когезия)**:

Элементы модуля должны быть тесно связаны между собой

Модуль должен быть внутренне согласованным

**Повторное использование**:

Модули должны быть пригодны для применения в разных частях программы или проектах

**Реализация модульности:**

Функции и процедуры

Разделение кода на файлы (.h и .cpp)

Библиотеки

Пространства имен (namespaces)

Классы и объекты

Интерфейсы и абстрактные классы

**7. Указатели: описание, операции разадресации и взятия адреса, адресная арифметика.**

**Указатель** — переменная, которая хранит адрес в памяти, где находится объект определенного типа.

Описание указателей:

int\* ptr; // Указатель на int  
char\* str; // Указатель на char  
void\* generic; // Указатель на неизвестный тип

**Основные операции:**

**Операция взятия адреса (&)**:

Возвращает адрес переменной

Пример: int\* ptr = &var;

**Операция разадресации (\*):**

Возвращает значение, находящееся по адресу в указателе

Пример: int value = \*ptr;

int x = 10;  
int\* p = &x; // p содержит адрес x  
int y = \*p; // y получает значение 10  
\*p = 20; // изменяет значение x на 20

**Адресная арифметика:**

Операции над указателями с учетом размера типа данных:

**Инкремент/декремент**:

ptr++ — перемещает указатель на размер типа вперед

ptr-- — перемещает указатель на размер типа назад

**Сложение/вычитание с целым**:

ptr + n — смещает указатель на n элементов вперед

ptr - n — смещает указатель на n элементов назад

**Разность указателей**:

ptr2 - ptr1 — возвращает количество элементов между указателями

**Пример с массивами:**

int arr[5] = {10, 20, 30, 40, 50};  
int\* p = arr; // p указывает на первый элемент  
int x = \*(p + 2); // x = 30 (третий элемент)  
p++; // p указывает на второй элемент  
int y = \*p; // y = 20