# Программа
# «*Современное WEB-программирование*»

**Входной ассемент**

**Лекция 2.**

**Модуль 3.** «Введение в алгоритмизацию и основы языков web-программирования»

# Введение в алгоритмизацию и программирование на языке С

**Цель:**

познакомить с основами алгоритмизации и синтаксисом языка С, научить писать простые программы и работать с базовыми структурами данных.

# Что такое алгоритмизация?

**Алгоритмизация** — это процесс разработки и описания последовательности шагов, которые необходимо выполнить для решения определённой задачи или достижения конкретной цели.

**Ключевые особенности алгоритмизации:**

- создаёт чёткие инструкции, понятные компьютеру;
- служит основой для написания программного кода;
- обеспечивает эффективное и корректное выполнение задач;
- упрощает отладку и поддержку кода.

**Формы представления алгоритмов:**

- текстовое описание;
- блок-схемы;
- псевдокод;
- другие формализованные представления.

**Основные свойства алгоритмов (как результата алгоритмизации):**

- **Дискретность** — алгоритм состоит из отдельных шагов, выполняемых последовательно.
- **Результативность** — выполнение алгоритма приводит к определённому результату (даже если это сообщение об отсутствии решения).
- **Детерминированность** — инструкции на каждом шаге чётко определены, без разночтений.
- **Массовость** — алгоритм применим к похожим задачам с разными исходными данными (например, один алгоритм решает квадратное уравнение для любых чисел).

# Что такое алгоритмизация?

**Примеры алгоритмизации в повседневной жизни и технологиях:**

- рецепт приготовления блюда;
- инструкция по сборке мебели;
- последовательность действий робота;
- инструкции для автоматизированного станка;
- код компьютерной программы.

**Роль алгоритмизации в IT:**

- оптимизация кода и повышение его эффективности;
- работа с большими данными (сортировка, фильтрация, поиск);
- разработка интерфейсов и логики приложений;
- машинное обучение и анализ данных;
- обработка изображений, распознавание речи и другие задачи искусственного интеллекта.

**Виды алгоритмов, используемых при алгоритмизации:**

- **Ветвящиеся** — включают условия, при которых выполняются разные действия (например, удаление отрицательных чисел из списка).
- **Циклические** — повторяют блок действий заданное количество раз или до выполнения условия.
- **Рекурсивные** — алгоритм вызывает сам себя с другими входными данными (пример — расчёт чисел Фибоначчи).

Таким образом, алгоритмизация — фундаментальный этап в программировании и решении задач любой сложности, позволяющий разбить сложную проблему на последовательность простых шагов.

# Введение в язык C и первая программа

**Особенности языка C:**

- Компилируемый язык среднего уровня
- Высокая производительность
- Близость к аппаратному обеспечению
- Стандартизация: C89, C99, C11

**Простейшая программа:**

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

- Структура программы
- Функция main() - точка входа
- Библиотека stdio.h для ввода-вывода

# Переменные и консольный ввод-вывод

**Основные типы переменных:**
•int - целые числа
•float - числа с плавающей точкой
•double - двойной точности
•char - символы

**Ввод и вывод данных:**

```
int age;
float salary;
printf("Введите возраст: ");
scanf("%d", &age);
printf("Введите зарплату: ");
scanf("%f", &salary);
printf("Возраст: %d, Зарплата: %.2f\n", age, salary);
```

- Форматные спецификаторы: %d, %f, %c, %s
- Оператор взятия адреса: &

# Условные операторы и логические операции

**Полная развилка (if-else):**

```
if (condition) {
    // код если истина
} else {
    // код если ложь
}
```

**Усеченная развилка (if):**

```
if (condition) {
    // код если истина
}
```

**Логические операции:**

- && - И
- || - ИЛИ
- ! - НЕ
- ==, !=, >, <, >=, <= - сравнения

# Язык блок-схем и алгоритмизация

**Основные элементы блок-схем:**

- ○ - Начало/Конец
- ▭ - Процесс
- ◊ - Решение (условие)
- ▭ - Ввод/Вывод
- → - Направление

**Пример алгоритма:**

Начало

↓

Ввод числа n

↓

n > 0? → Нет → Вывод "Отрицательное"

↓ Да

Вывод "Положительное"

↓

Конец

# Цикл DO-WHILE и трассировка

**Синтаксис DO-WHILE:**

```
do {
    // тело цикла
} while (condition);
```

**Особенности:**

- Тело выполняется минимум 1 раз
- Условие проверяется после итерации

**Трассировка - пошаговое выполнение:**

```
int i = 1;
do {
    printf("%d ", i);
    i++;
} while (i <= 5);
// Вывод: 1 2 3 4 5
```

# WinAPI Графика - основные фигуры

**Инициализация графики:**

```
#include <windows.h>

int main() {
    InitGraph();
    // графические функции
    CloseGraph();
    return 0;
}
```

**Базовые фигуры:**

- Rectangle() - прямоугольник
- Ellipse() - эллипс
- Line() - линия
- Arc() - дуга
- Chord() - хорда
- Pie() - сектор

# Кисти и относительные координаты

**Работа с кистями:**

```
// Создание кисти
HBRUSH hBrush = CreateSolidBrush(RGB(255, 0, 0));
SelectObject(hDC, hBrush);

// Заливка фигуры
Rectangle(hDC, x1, y1, x2, y2);
```

**Относительные координаты:**

```
int centerX = 400, centerY = 300;
int radius = 50;

// Рисование относительно центра
Ellipse(hDC,
        centerX - radius,
        centerY - radius,
        centerX + radius,
        centerY + radius);
```

# Функции с параметрами и управление размером

**Создание функций:**

```
void drawCircle(int x, int y, int radius) {
    Ellipse(hDC, x-radius, y-radius, x+radius, y+radius);
}

// Использование
drawCircle(100, 100, 50);
drawCircle(200, 200, 30);
```

**Управление размером фигур:**

```
void drawScaledShape(int x, int y, float scale) {
    int size = 50 * scale;
    Rectangle(hDC, x, y, x + size, y + size);
}
```

# Рекурсия и оператор SWITCH

**Рекурсивные функции:**

```
int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}
```

**Оператор SWITCH:**

```
switch (variable) {
    case 1:
        // код для case 1
        break;
    case 2:
        // код для case 2
        break;
    default:
        // код по умолчанию
}
```

# Работа с клавиатурой и таймером

## Обработка клавиатуры:

```
if (GetAsyncKeyState(VK_LEFT)) {
    // перемещение влево
}
if (GetAsyncKeyState(VK_SPACE)) {
    // действие по пробелу
}
```

## Цикл WHILE с таймером:

```
while (gameRunning) {
    startTime = GetTickCount();

    // игровая логика

    // контроль FPS
    while (GetTickCount() - startTime < 16); // ~60 FPS
}
```

# Массивы и структуры

## Одномерные массивы:

```
int numbers[10];        // объявление
numbers[0] = 5;         // инициализация
for (int i = 0; i < 10; i++) {
    printf("%d ", numbers[i]);
}
```

## Структуры:

```
struct Point {
    int x;
    int y;
};

struct Point p1;
p1.x = 100;
p1.y = 200;
```

# Двумерные массивы и вложенные циклы

**Двумерные массивы:**

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

**Вложенные циклы:**

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%d ", matrix[i][j]);
    }
    printf("\n");
}
```

# Указатели и работа с памятью

**Основы указателей:**

```
int value = 10;
int *ptr = &value;      // указатель на value

printf("Значение: %d\n", *ptr);  // разыменование
printf("Адрес: %p\n", ptr);
```

**Динамическая память:**

```
int *arr = (int*)malloc(10 * sizeof(int));
// работа с массивом
free(arr);  // освобождение памяти
```

# Работа с файлами и случайные числа

**Сохранение в файл:**

```
FILE *file = fopen("data.txt", "w");
fprintf(file, "%d %f %s\n", number, value, text);
fclose(file);
```

**Загрузка из файла:**

```
FILE *file = fopen("data.txt", "r");
fscanf(file, "%d %f %s", &number, &value, text);
fclose(file);
```

**Псевдослучайные числа:**

```
srand(time(NULL));          // инициализация
int randomNum = rand() % 100; // число от 0 до 99
```

# Сортировка и работа со строками

**Алгоритмы сортировки:**

```
// Пузырьковая сортировка
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                // обмен элементов
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

**Работа со строками:**

```
char text[100];
strcpy(text, "Hello");      // копирование
strcat(text, " World");     // конкатенация
int len = strlen(text);     // длина строки
```

# Введение в объектно-ориентированное программирование

- **Объектно-ориентированное программирование (ООП)** — подход, при котором программа состоит из объектов, взаимодействующих между собой.

- **Суть ООП:** связать поведение сущности с её данными, спроецировать объекты реального мира в программный код.

- **Пример из жизни:** телевизор (объект) имеет свойства (марка, размер) и методы (включение, переключение каналов).

- **Задача ООП:** упростить разработку и поддержку кода за счёт модульности и повторного использования.

- **Ключевые термины:** объект, класс, метод, свойство.

# Основные концепции ООП - классы и объекты

**Класс vs Объект:**

- **Класс** - шаблон/описание (как чертеж дома)
- **Объект** - экземпляр класса (как конкретный дом)

**Сравнение языков:**

- **C** - процедурный, нет ООП
- **C++** - мультипарадигмальный, поддерживает ООП
- **Java** - чисто объектно-ориентированный

**Пример класса:**

```
class Student {
    // поля класса
    private String name;
    private int age;

    // методы класса
    public void study() {
        System.out.println(name + " is studying");
    }
}
```

# Три столпа ООП - инкапсуляция, наследование, полиморфизм

**Инкапсуляция:**

•Сокрытие внутренней реализации
•Контроль доступа через методы
•Модификаторы доступа:
    •private - только внутри класса
    •protected - класс + наследники
    •public - полный доступ

**Наследование:**

```
class Person {
    protected String name;
}
class Student extends Person {
    // Student наследует поле name
}
```

**Полиморфизм:**

- Один интерфейс - множество реализаций
- Переопределение методов

# Структуры данных - односвязные списки

**Принцип односвязного списка:**

[Данные | Next] → [Данные | Next] → [Данные | NULL]

**Реализация узла:**

```
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
```

**Основные операции:**

- Добавление в начало/конец
- Удаление элемента
- Поиск элемента
- Обход списка

# Двусвязные списки

**Принцип двусвязного списка:**
NULL ← [Prev | Данные | Next] ↔ [Prev | Данные | Next] → NULL

**Реализация узла:**

```
class DoubleNode {
    int data;
    DoubleNode prev;
    DoubleNode next;

    DoubleNode(int data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}
```

- **Преимущества:**
- Обход в обоих направлениях
- Более эффективное удаление
- Удобство навигации

# Библиотека STL в C++ и практическое применение

**STL (Standard Template Library):**

- Готовые реализации структур данных
- Шаблонные классы

**Основные контейнеры:**

```
#include <list>
#include <vector>
#include <map>

std::list<int> myList;          // двусвязный список
std::vector<int> myVector;      // динамический массив
std::map<string, int> myMap;    // ассоциативный массив
```

- **Практическое использование:**
- Выбор структуры под задачу
- Эффективность операций
- Готовые алгоритмы сортировки, поиска

**Итог:** Понимание ООП и структур данных - основа современного программирования