Функции. Параметры и аргументы функции. Возврат значения из функции. Объявление и определение функции. Примеры

Функции
Функция – это блок кода в программе, созданная для выполнения отдельной задачи.
Основная цель функции – это декомпозиция.
Декомпозиция – процесс разделения целого (системы/задачи/программы), на отдельные части. Каждая часть выполняет свою микро задачу и в совокупности они образуют систему.

Преимущества:

- Упрощение разработки
- Упрощение тестирования
- Гибкость при внесении изменений в отдельные функции
- Возможность повторно использовать одну и ту же функцию в различных других программах/системах/задачах
  Пример функции:

```
char slon()
{
        return 'Z';
}
```

Сигнатура функции
Сигнатура функции в си это её:

- тип возвращаемых данных `char slon()`
- её параметры `int slon` (int arg)
  Если функция не возвращает значение (тип `void`), то она называется процедурой

Объявление и определение функции

- **Объявление (прототип):** Сообщает компилятору, что существует функция с определённым именем, возвращаемым типом и параметрами.

  ```
  тип_возврата имя_функции(список_параметров);
  ```

  Пример:

  ```
  int add(int a, int b);  // Объявление функции
  ```

- **Определение:** Реализация функции, содержащая её тело.

  ```
  тип_возврата имя_функции(список_параметров) {
  // тело функции
  }
  ```

  Пример:

  ```
  int nu_hz_summa(int a, int b) {
    return a + b;  // Возвращаем сумму
  }
  ```

Параметры и аргументы функции

- **Параметры:** Объявляются в заголовке функции и принимают значения при вызове функции. Это "входные данные" функции.

  ```
  void printMessage(char *message);
  ```

- **Аргументы:** Конкретные значения, передаваемые при вызове функции.

  ```
  printMessage("Hello, World!");
  ```

- **Типы параметров**
  1. *Передача по значению*
     Копия значения аргумента передаётся в функцию. Изменения внутри функции

не затрагивают оригинальные данные.

```
void increment(int x) {
x++;  // Локальная копия, оригинал не изменяется
}
```

```
2. *Передача по указателю*
  Передаётся адрес переменной, поэтому изменения внутри функции
затрагивают оригинальные данные.
  ```c
  void increment(int *x) {
  (*x)++;  // Изменяет оригинальную переменную
  }
```

Возврат значения функции

- **С помощью оператора `return`:** Функция может возвращать значение, используя `return`. Тип возвращаемого значения указывается в объявлении функции.

```
  int multiply(int a, int b) {
  return a * b;
  }
```

- **Функции без возврата(процедуры):** Используют тип `void`.

```
  void slon() {
    printf("ZV ZV ZV! \n");
}
```

- **Множественные `return`:** Функция может иметь несколько операторов `return` для выхода в зависимости от условий.

```
 int max(int a, int b) {
    if (a > b)
        return a;
    else
```
```

```
        return b;
    }
```

Пример программы с функциями и процедурами

```
#include <stdio.h>

int add(int n1, int n2);    //
void check(int n1);         // Объявляем функции и процедуры
void slon();                //
void neSlon();              //

int main() {
    int num1 = -7, num2 = 10;
    int sum = add(num1, num2);      // присваиваем перепенной значение функции
    check(sum);                     // вызываем процедуру
    return 0;
}

int add(int n1, int n2)
{
    return n1 + n2;
}

void check(int n)
{
    if (n > 0)
    {
        slon();
    } else
    {
        neSlon();
    }
}

void slon()
{
    printf("ZV ZV ZV!\n");
}

void neSlon()
{
```

```
    printf("УК РФ Статья 280.3. Дискредитация ВС РФ.");
}
```

Прочие важные моменты

- Область видимости
  - Если не объявлять функцию в начале программы, то вызвать ее можно только если эта функция реализована выше места вызова. То есть, нельзя вызвать функцию на 10 строке, а реализовать ее на какой-нибудь 100 строке.
  - Локальные переменные функции видны только внутри неё.
  - Глобальные переменные(заданные вне функций) видны всем функциям.
- Рекурсия:
  - Функция может вызвать саму себя
  - Пример:

```
int factorial(int n) {
if (n == 0)
    return 1;
return n * factorial(n - 1);
}
```