

# Технология программирования на ЭВМ. Переменные. Первая программа.

Баев А.Ж.

Казахстанский филиал МГУ

24 октября 2019

## ПК



HDD (1 Tb)

RAM (4Gb)

CPU (2.3 GHz)

Частота процессора (CPU): 2.3 ГГц

Объем оперативной памяти (RAM): 4 Гб

Жесткий диск (HDD): 1 ТБ

## Где хранятся переменные?

# Переменные

Переменная — это часть оперативной памяти (RAM), в которую можно записывать и из которой можно извлекать значения по имени, которое задает сам программист.

```
int x;
```

— выделить 4 байта RAM, к которым можно обращаться по имени x;

```
double tmp;
```

— выделить 8 байт RAM, к которым можно обращаться по имени tmp.

Обратите внимание на точку с запятой. Данным символом будут заканчиваться большинство действий.

## Имена переменных

Имя переменной может содержать буквы (a-z, A-Z), цифры (0-9) и символ подчёркивания. Но не должно начинаться с цифры:

```
int value;
int VALUE;
int snake_case;
int CamelCase;
int x2y;
int size_2;
```

```
int 2_size;
```

Красным будут отмечены примеры с ошибками.

## Имена переменных

Имена переменных лучше давать такие, чтобы они несли в себе смысл.

Хороший стиль

```
double square;  
double perimeter;  
double semiperimeter;
```

```
double t;  
double t15;  
double xxx;
```

Жёлтым будут отмечены примеры без ошибок, но с плохим стилем.

# Переменные

Можно объявить сразу несколько переменных, если они одного типа:

```
long long a, b, c;  
double first_position, second_position;
```



# Оператор присваивания

В память, за которую отвечает переменная, можно записать значение других переменных:

```
int x, y;  
x = 7;  
y = x;
```



## Оператор присваивания

Обратите внимание, что оператор присваивания вычисляет значение у переменной СПРАВА и записывает результат в переменную СЛЕВА.

```
x = y;
```

Часто в литературе его обозначают стрелкой, чтобы понимать откуда куда копируется значение.

```
x ← y;
```

## Мусорное значение

Какое значение находится в переменной `x`, если ей ничего не присвоить?

```
int x;
```

## Мусорное значение

Какое значение находится в переменной `x`, если ей ничего не присвоить?

```
int x;
```

Неопределенное (может оказаться абсолютно любое случайное число) или, как говорят, «мусорное».

# Инициализация

Оператор присваивания можно использовать непосредственно при объявлении переменной:

```
int x = 7;
double first_coordinate = 12.3;
long long a = 5LL, b = 7LL;
float size = 7.0, value;
```

## Ключевой пример.

Даны 2 переменные a, b:

```
int a = 5, b = 7;
```

Поменять значения между ними с помощью операций присваивания (копирование значения из переменной в переменную). Можно использовать дополнительные переменные.

Дано

a 

5
---

      b 

7
---

Хотим получить

a 

7
---

      b 

5
---

## Ключевой пример.

Даны 2 переменные a, b типа int. Поменять значения между ними.

```
int a = 5, b = 7, c;  
c = a;  
a = b;  
b = c;
```

```
int a = 5, b = 7, c;
```

a	5
---	---

b 7

c	?
---	---

```
c = a;
```

a 7

b	5
---	---

c 7

```
a = b;
```

a 5

b	5
---	---

c 7

```
b = c ;
```

a	5
---	---

b 7

c 7

## Оператор присваивания

В память, за которую отвечает переменная, можно записать значение арифметического выражения с другими переменными:

```
int a = 3, b = 4, c = 5, P;
P = a + b + c;
```

## Ключевой пример.

Даны 2 переменные a, b:

```
int a = 5, b = 7;
```

Поменять значения между ними с помощью операций присваивания и арифметических действий. Нельзя использовать дополнительные переменные.

Дано

a 5      b 7

Хотим получить

a 7      b 5



## Ключевой пример.

Поменять значения между ними с помощью операции присваивания и арифметических действий. Нельзя использовать дополнительные переменные.

```
int a = 5, b = 7;
a = a + b;
b = a - b;
a = a - b;
```

```
int a = 5, b = 7;
```

a	5	b	7
---	---	---	---

```
a = a + b;
```

a	12	b	5
---	----	---	---

```
b = a - b;
```

a	12	b	7
---	----	---	---

```
a = a - b;
```

a	5	b	7
---	---	---	---

Типичная ошибка.

3 = b ;

$$a + 2 = b;$$

## Неявное приведение типа.

```
int x = 2;  
double y;  
y = x;
```

Получим  $y = 2.0$ .

## Неявное приведение типа.

```
double y = 2.7;  
int x;  
x = y;
```

Получим  $x = 2$ .

## Неявное приведение типа.

Получим  $z = 4.5$  или  $z = 4.0$ ?

```
int x = 4, y = 5;
double z;
z = (x + y) / 2;
```

## Неявное приведение типа.

Получим  $z = 4.5$  или  $z = 4.0$ ?

```
int x = 4, y = 5;
double z;
z = (x + y) / 2;
```

1.  $4 + 5 \rightarrow 9$
2.  $9 / 2 \rightarrow 4$
3.  $z = 4 \rightarrow z = 4.0$

## Явное приведение типа.

```
double x = 123.345;
double y = x - int(x);
```

Получим  $y = 123.345 - 123 = 123.345 - 123.000 = 0.345$

# Каркас.

Минимальная программа:

```
int main()  
{  
  
    return 0;  
}
```



# Каркас.

Минимальная программа:

```
int main()  
{  
  
    return 0;  
}
```

Обладает рядом недостатков:

1. ничего не делает;
2. ничего не понятно.

## Каркас.

- `int main()`  
— здесь начинается выполнение программы (основная функция программы);
- `return 0;`  
— здесь завершается выполнение программы (успешно завершённые программы говорят операционной системе 0, когда всё выполняется по плану).

## Наш первый калькулятор.

Калькулятор, который складывает числа:

```
int main()
{
    int a = 3, b = 4, sum;
    sum = a + b;
    return 0;
}
```

## Наш первый калькулятор.

1. Откроем терминал (ctrl + alt + T).
2. Запустим редактор nano и создадим новый файл prog.c.

```
nano prog.c
```

3. Наберём код (внимательнее к скобкам и точкам с запятым)

```
int main()
{
    int a = 3, b = 4, sum;
    sum = a + b;
    return 0;
}
```

4. Сохраним творчество (ctrl+o) и выйдем (ctrl + x).

## Наш первый калькулятор.

- 5 С помощью компилятора gcc превратим текстовые команды из файла prog.c (понятные для людей) в машинные команды в новом файле prog (понятные для сри):

```
gcc prog.c -o prog
```

- 6 Запустим программу:

```
./prog
```

Но где же наша 7?

Но где же наша 7?

Но где же наша 7?

Здесь:



Но где же наша 7?

Здесь:



А давайте её отправим сюда:

```
Терминал
user@user-notebook:~$ ls
Видео
Документы
Загрузки
Изображения
Музыка
Общедоступные
Рабочий стол
Шаблоны
user@user-notebook:~$ █
```



## Наш первый калькулятор.

Нам нужно

1. подключить специальный файл, которые позволяет вводить и выводить информацию в терминал (stdio.h — standard input output header).
2. указать какую переменную (sum) и в каком виде (%d — decimal) распечатать (printf — print format string).

```
#include <stdio.h>
int main()
{
    int a = 3, b = 4, sum;
    sum = a + b;
    printf("%d", sum);
    return 0;
}
```

## Наш первый калькулятор.

1. С помощью компилятора gcc снова превратим текстовые команды из файла prog.c в машинные команды в файле prog:

```
gcc prog.c -o prog
```

2. Запустим программу:

```
./prog
```

3. И увидим что-то такое:

```
user@user-Notebook:~$ nano prog.c
user@user-Notebook:~$ gcc prog.c -o prog
user@user-Notebook:~$ ./prog
7user@user-Notebook:~$
```

А где теперь число?

## Наш первый калькулятор.

Добавим эффект клавиши Enter — переход на новую строку с помощью команды `puts()` (print string).

```
#include <stdio.h>
int main()
{
    int a = 3, b = 4, sum;
    sum = a + b;
    printf("%d", sum);
    puts("");
    return 0;
}
```

## Наш первый калькулятор.

1. С помощью компилятора gcc снова превратим текстовые команды из файла prog.c в машинные команды в файле prog:

```
gcc prog.c -o prog
```

2. Запустим программу:

```
./prog
```

3. И увидим что-то такое:

```
user@user-Notebook:~$ nano prog.c
user@user-Notebook:~$ gcc prog.c -o prog
user@user-Notebook:~$ ./prog
7
user@user-Notebook:~$
```

Вы всё ещё хотите программировать на C?

## Форматный ввод и вывод.

Какие бывают форматы ввода и вывода?

формат	размер	тип	пример
%u	4	unsigned int	2019
%d	4	int	-2019
%x	4	unsigned int	CAFE2019
%llu	8	unsigned long long	12345678987654321
%ll	8	long long	-12345678987654321
%f	4	float	1.23456f
%e	4	float	123456e+05f
%lf	8	double	1.23456
%e	8	double	123456e+05
...	...	...	...

## Форматный ввод и вывод.

Ввод в переменные

```
scanf("format", &a, &b, &c, ...);
```

Обратите внимание на амперсанды (&).

Вывод с переменными

```
printf("format", a, b, c, ...);
```

## Форматный ввод и вывод.

Считать целое число

```
int x;  
scanf("%d", &x);
```

Обратите внимание на амперсанд (&).

Вывести целое число

```
int x = 7;  
printf("%d", x);
```

## Форматный ввод и вывод.

Считать нецелое число

```
double x;
scanf("%lf", &x);
```

Обратите внимание на амперсанд (&).

Вывести нецелое число

```
double x = 7.5;
printf("%lf", x);
```



## Более сложные форматы.

Вывести несколько чисел

```
int x = 2, y = 3;
printf("%d_ %d", x);
```

На выводе будет

```
2 3
```

Вывести текст и числа

```
int x = 10, y = 6;
printf("a=%d_ and_ b=%d.", x, y);
```

На выводе будет

```
a=10 and b=6.
```

## Перевод на новую строку.

```
int x = 7;
printf("%d\n", x);
```

```
int x = 7;
printf("%d", x);
puts("");
```

# Задача 1.

Даны 2 целых числа от  $-1000000$  до  $1000000$ . Найти их сумму.

Ввод	3 4	-1001 3020
Вывод	7	2019

## Задача 1.

```
#include <stdio.h>
int main()
{
    int a, b, sum;
    scanf("%d", &a);
    scanf("%d", &b);
    sum = a + b;
    printf("%d", sum);
    printf("\n");
    return 0;
}
```

## Задача 1.

```
#include <stdio.h>
int main()
{
    int a, b, sum;
    scanf("%d□%d", &a, &b);
    sum = a + b;
    printf("%d\n", sum);
    return 0;
}
```

## Задача 2.

Даны 3 вещественных положительных числа таких, что существует треугольник с данными сторонами. Найти квадрат площади треугольника и вывести с точностью 2 знака после запятой.

Ввод	3.0 4.0 5.0	6.0 7.0 8.0
Вывод	36.00	413.44

## Задача 2.

```
#include <stdio.h>
int main()
{
    double a, b, c, p, S;
    scanf("%lf_%lf_%lf", &a, &b, &c);
    p = (a + b + c) / 2.0;
    S = p * (p - a) * (p - b) * (p - c);
    printf("%.2lf\n", S);
    return 0;
}
```

## Задача 3.

Дано положительное вещественное число. Найти первую цифру дробной части числа.

Ввод	123.567	0.012
Вывод	5	0



## Задача 3.

```
#include <stdio.h>
int main()
{
    double x;
    int ans;
    scanf("%lf", &x);
    ans = (int)(x * 10) % 10;
    printf("%d\n", ans);
    return 0;
}
```

### Задача 4.

Даны 2 целых положительных числа. Вычислить  $\frac{2}{\frac{1}{a} + \frac{1}{b}}$ .

Ввод	2 4	7 7
Вывод	2.67	7.00

### Задача 4.

### Неправильное решение

```
#include <stdio.h>

int main()
{
    int a, b;
    double answer;
    scanf("%d %d", &a, &b);
    answer = 2 / (1 / a + 1 / b);
    printf("%.2lf\n", answer);
    return 0;
}
```

В обоих примерах будет операция вида  $2 / (0 + 0)$  — то есть деление на ноль:

Исключение в операции с плавающей точкой  
(стек памяти сброшен на диск)

## Задача 4.

### Правильное решение

```
#include <stdio.h>
int main()
{
    int a, b;
    double answer;
    scanf("%d %d", &a, &b)
    answer = 2.0 / (1.0 / a + 1.0 / b);
    printf("%.2lf\n", answer);
    return 0;
}
```

## Задача 4.

### Правильное решение

```
#include <stdio.h>
int main()
{
    int a, b;
    double answer;
    scanf("%d %d", &a, &b)
    answer = 2 / (1.0 / a + 1.0 / b);
    printf("%.2lf\n", answer);
    return 0;
}
```

## Работа над ошибками

```
int main()
{
    x = 5;
    printf("%d", x);
    return 0;
}
```

Компиляция данной программы:

```
gcc prog.c -o prog -Wall
```

Вместо правильного «ничего» на экране что-то появилось:

```
prog.c: In function 'main':
prog.c:3:2: error: 'x' undeclared
          (first use in this function)
    x = 5;
    ^
```

- Объясните какие объявления переменной корректны.  
а) `int A1;` б) `float msu.kz;` в) `double a_b_c = 3.0;`
- Вычислите значения переменных после выполнения  

```
int x = 10, y = 5;
x = y - 2;
y = 10 + 5 * x;
x = y * 2 % x / 2;
```
- Напишите код минимальной программы, которая ничего не делает, и команду для компилирования.
- Напишите код программы. Дано целое число от 0 до 1000. Вывести последнюю цифру.
- Объясните какие записи корректны для вычисления  $\frac{2}{\frac{1}{a} + \frac{1}{b}}$ , если  $a$  и  $b$  целые числа:  
а) `2 / (1.0 / a + 1.0 / b);`  
б) `2.0 / (1 / a + 1 / b);`  
в) `2 / (1 / a + 1.0 / b);`