acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CoffeeVariable :

Характеристики переменной

Каждая переменная имеет свое:
- имя (идентификатор);
- значение (число, которое в ней записано);
- тип;
- время жизни;
- область видимости:
  - от декларации и ниже
  - до конца блока, в котором определена

Локальные переменные

- **область видимости :** от декларации до конца блока (функции).
- **время жизни :** время выполнения этого блока.
- **начальное значение по умолчанию :** мусор.

Напишем программу, которая вычисляет сумму 5 чисел.

```
#include <stdio.h>

int main()
{
    int sum;                // начало области видимости sum

    sum = 0;                // попробуем удалить эту строку

    for (int i=0; i < 5; i++) {
        int x;              // начало области видимости x
        scanf("%d", &x);
        sum = sum + x;
    }                       // конец области видимости x
    printf("sum = %d\n", sum);
    return 0;
                            // конец области видимости sum
}
```

- Нельзя использовать переменную до или после **области видимости**

Нельзя написать **x = 0** до или после цикла. Ее не видно.

- Время жизни: в цикле 5 раз:
  - создается переменная x,
  - в нее читается число scanf("%d", &x),

    - ее значение прибавляется к переменной sum,
    - она **разрушается** в конце каждой итерации цикла.
- Начальное значение по умолчанию.

Представим, что в программе нет строки sum = 0. При создании в локальной переменной sum может лежать любое число (память под переменную выделили, но не записали в нее 0).

Тогда результат работы программы тоже - любое число.

Некоторые компиляторы записывают 0 в локальные переменные при создании. Но по стандарту они не обязаны это делать. Тогда программа без sum=0 может работать, скомпилированная на одном компиляторе (у вас на компьютере) и не работать, скомпилированная на другом компиляторе (на сервере проверяющей системы).

 Если у вас на компьютере "все работает", а на другом компьютере "работает неправильно", проверьте все ли переменные инициализированы.

Аргументы функции. Передаются по значению.

- **область видимости :** вся функция.
- **время жизни :** время выполнения функции.
- **начальное значение по умолчанию :** значение аргумента при вызове.

Что будет напечатано и почему? (Функция ничего не возвращает.)

```
#include <stdio.h>

void foo(int x) {
    x = x + 1;
}

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

Ответ:

5

В программе 2 разных переменных с одинаковым именем x. Одна переменная x - локальная переменная в функции main. В нее записывается число 5.

При вызове foo(x) вычисляется значение этой x (5) и вызывается foo(5).

При вызове foo(5) создается переменная x - аргумент функции foo и в нее записывается значение при вызове (5).

Значение переменной x функции foo увеличивается на 1. Переменная x в foo равна 6, а другая переменная x в main по-прежнему 5 (ее никто не менял).

При выходе из функции foo переменная x разрушается (время жизни аргумента функции = времени вызова функции).

Управление возвращается в функцию main. Переменная x функции main по-прежнему содержит 5. Увеличивалась на 1 и разрушалась другая переменная (с таким же именем x).

То же самое повторяется с другим вызовом foo(x).

Если вам не понятно, представьте, что в функции foo аргумент назвали по-другому. Очевидно, что изменение переменной y никак не отразится на переменной x:

```
#include <stdio.h>

void foo(int y) {
    y = y + 1;
}
```

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

Глобальные переменные

 ложатся в статическую область памяти.

- **область видимости :** от декларации до конца **файла.**
  - область видимости можно расширить на другие файлы с помощью extern,
  - поэтому пишут в начале файла.
- **время жизни :** время жизни всей программы.
- **начальное значение по умолчанию :** ноль.

 нельзя вне функций написать x = 5 или scanf("%d", &x), только явно задать значение при объявлении.

Что будет напечатано и почему?

```
#include <stdio.h>

int x = 5;

void foo() {
    x = x + 1;
}

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

Ответ:

7

Переменная x одна. При создании в нее записали число 5.

При вызове foo эта глобальная переменная увеличивается на 1 (стало 6).

При следующем вызове увеличивается еще на 1 (стало 7).

Дальше идет печать этой же переменной x (печатаем 7).

Что будет, если вместо int x = 5; будет написано int x; ?

Ответ:

2

При создании в переменную запишется не 5, а 0 (значение по умолчанию). Дальше – как в примере.

Имя глобальной переменной совпадает с именем аргумента или локальной переменной

Какая переменная используется, если имя глобальной переменной совпадает с именем аргумента или локальной переменной?

Не надо так делать. Никогда.

Локальная перекрывает аргумент и глобальную переменную.
Аргумент перекрывает глобальную переменную.

Пример очень плохого кода (что будет напечатано и почему?)

```
#include <stdio.h>

int x = 5;

void foo(int x) {
    x = x + 1;
}

int main() {
    int x;
    x = 10;
    foo();
    foo();
    printf("x=%d\n", x);
    return 0;
}
```

Глобальная переменная x = 5 не видна ни в одной функции (ни в foo, ни в main). Поэтому пример работает точно так же, как пример из раздела "Аргументы функций".

Модификатор static

Пример на повторение

Что будет выведено на печать и почему?

```
#include <stdio.h>

void foo() {
    int x = 5;
    x = x + 1;
    printf("x=%d\n", x);
}

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

Ответ:

```
x=6
x=6
```

Добавим static

- static int x;
- **область видимости :**
  - блок для локальных,
  - от декларации до конца файла для НЕ локальных (нельзя увидеть в другом файле, extern не поможет).
- **время жизни :** время жизни всей программы.
- **начальное значение по умолчанию :** ноль.
- **инициализируется один раз**

 Располагается в статической области памяти.

Пример static int x;

Что будет выведено на печать и почему?

```
#include <stdio.h>

void foo() {
    static int x;
    x = x + 1;
```

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

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

Ответ:

```
x=1
x=2
```

В функции foo создается переменная x и инициализируется 0.

При вызове foo() эта переменная +1, печатается x=1.
Она НЕ разрушается в конце вызова функции (сохраняется между вызовами).

При следующем вызове к ней еще раз +1 и печатается x=2.

Пример static int x = 2;

Что будет выведено на печать и почему?

```
#include <stdio.h>

void foo() {
    static int x = 2;
    x = x + 1;
    printf("x=%d\n", x);
}

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

Ответ:

```
x=3
x=4
```

В функции foo создается переменная x и инициализируется 2.

При вызове foo() эта переменная +1, печатается x=3.
Она НЕ разрушается в конце вызова функции (сохраняется между вызовами).

При следующем вызове она не нуждается в "инициализации еще раз", так как уже создана, поэтому x=2 не отрабатывает. К ней еще раз +1 и печатается x=4.

Передача аргумента по адресу

Поменяем значение 2 переменных местами

Есть переменные x и y. Нужно поменять их значения местами.

Если у нас чашка с молоком и чашка с соком, то поменять сок в этих двух чашках невозможно. Нужно взять еще одну чашку.

Аналогично, с помощью новой переменной t поменяем значение переменных местами.

```
#include <stdio.h>

int main() {
    int x, y;
    x = 5;
    y = 3;

    int t;
    t = x;
```

```
    x = y;
    y = t;

    printf("x=%d y=%d\n", x, y);
    return 0;
}
```

Работает. Теперь попробуем написать это в виде отдельной функции swap(x,y).

Что будет выведено на печать и почему?

```
#include <stdio.h>

void swap(int x, int y) {
    int t;
    t = x;
    x = y;
    y = t;
}

int main() {
    int x, y;
    x = 3;
    y = 5;
    swap(x,y);
    printf("x=%d y=%d\n", x, y);
    return 0;
}
```

Ответ:

x=3 y=5

Не работает.

Потому что в функцию при вызове foo(x,y) передаются значения (копии) переменных. На состояние переменных x и y в функции main действия в функции foo не влияют.

Адреса переменных

Что будет выведено на печать и почему?

```

```

Ответ:

x=6
x=6