Функции

Аргументы функции

В Си все аргументы функции передаются «по значению», т.е. в виде копии.

Авторы языка: «Благодаря этому свойству обычно удается написать более компактную программу, содержащую меньшее число посторонних переменных, поскольку параметры можно рассматривать как должным образом инициализированные локальные переменные.»

Аргументы функции

```
#include <stdio.h>

int power(int base, int n)
{
    int result = 1;

    // base = 2, n = 5

    while (n-- > 0)
        result *= base;

    // n = -1

    return result;
}
```

```
int main(void)
{
    int a, n = 5;

    // n = 5

    a = power(2, n);

    printf("%d^%d = %d\n",
                    2, n, a);
    // 2^5 = 32

    return 0;
}
```

Аргументы функции

Передача аргументов «по значению» создает сложности при реализации функций, которые возвращают несколько параметров одновременно.

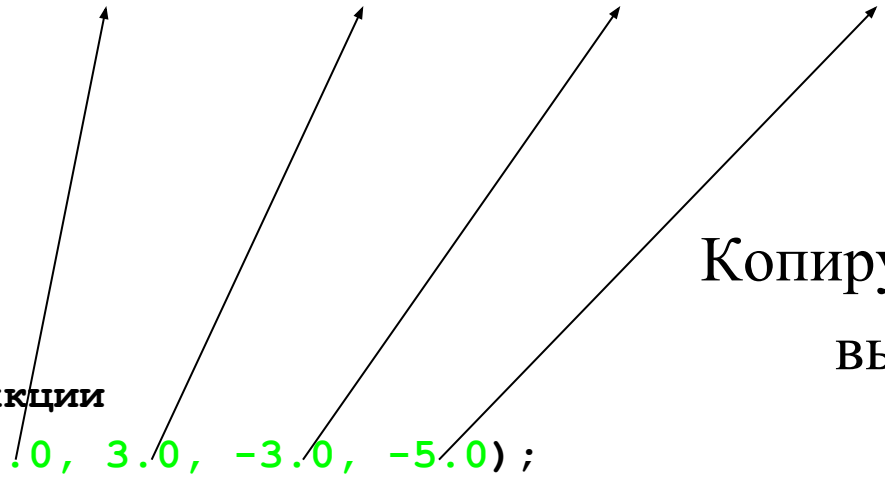
```
#include <stdio.h>

void decompose(
        float f,
          int int_part,
            float frac_part)
{
    int_part  = f;
    // int_part = 3
    frac_part = f - int_part;
    // frac_part = 0.14159
}
```

```
int main(void)
{
    int i = 0;
    float f = 0.0f;

    decompose(3.14159, i, f);

    printf("%d %f\n", i, f);
    // 0 0.000000

    return 0;
}
```

«Параметры» vs «аргументы»

Параметры (формальные параметры)

```
// Определение функции
float length(float x1, float y1, float x2, float y2)
{
    ...

int main(void)
{
    ...
    // Вызов функции
    l = length(5.0, 3.0, -3.0, -5.0);
    ...
```



Копируются при вызове

Аргументы (фактические параметры)

Пример 1

```
#include <stdio.h>
#include <math.h>

float length(float x1,float y1,
             float x2, float y2)
{
    float dx, dy;

    dx = x2 - x1;
    dy = y2 - y1;

    return sqrt(dx * dx +
                dy * dy);
}
```

```
int main(void)
{
    float l;

    l = length(5.0, 3.0,
                    -3.0, -5.0);

    printf("Length is %f\n", l);

    return 0;
}
```

Выполнение вызова функции

1. Выделяется область памяти доступная только вызываемой функции (length).
2. В этой области памяти создаются переменные-параметры (x1, y1, x2, y2) и локальные переменные (dx, dy).
3. Переменным-параметрам присваиваются начальные значения, в качестве которых выступают аргументы из точки вызова (x1 = 5.0, y1 = 3.0, x2 = -3.0, y2 = -5.0).
4. Инициализируются локальные переменные. В случае отсутствия такой инициализации локальные переменные (dx, dy) содержат «мусор».

Выполнение вызова функции

5. Выполняется тело функции.
6. Вычисленное значение возвращается из функции (в нашем случае попадает в переменную l).
7. Выделенная область памяти разрушается.

Выполнение программы

http://pythontutor.com/c.html#mode=edit

(https://goo.gl/kNNpZh, либо рисунке в архиве)

2. Управление получает функция main. Для функции main выделяется специальная область. В ней располагается переменная l. Ее значение неопределено («мусор»).
4. Управление получает функция length. Для функции length выделяется специальная область. В ней располагаются переменные-параметры x1, y1, x2, y2 (их значения инициализированы аргументами) и локальные переменные dx, dy (содержат «мусор»).

Выполнение программы

6. Управление возвращается в функцию main. Специальная область для функции length разрушается. Значение, вычисленное этой функцией, попадает в переменную l.

Пример 2

```
#include <stdio.h>

void swap(int a, int b)
{
    int temp = a;

    a = b;
    b = temp;
}
```

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

    printf("Before swap: %d,"
                  " %d\n", x, y);

    swap(x, y);

    printf("After swap: %d,"
                  "%d\n", x, y);

    return 0;
}
```

Выполнение программы

```
1  void swap(int a, int b)
2  {
3    int temp = a;
4    a = b;
5    b = temp;
6  }
7
8  int main(void)
9  {
10   int x = 5, y = 3;
11
12   swap(x, y);
13
14   return 0;
15 }
```

main: x = 5, y = 3

1.

```
1  void swap(int a, int b)
2  {
3    int temp = a;
4    a = b;
5    b = temp;
6  }
7
8  int main(void)
9  {
10   int x = 5, y = 3;
11
12   swap(x, y);
13
14   return 0;
15 }
```

main: x = 5, y = 3

swap: a = 5, b = 3, temp = 5

2.

```
1  void swap(int a, int b)
2  {
3    int temp = a;
4    a = b;
5    b = temp;
6  }
7
8  int main(void)
9  {
10   int x = 5, y = 3;
11
12   swap(x, y);
13
14   return 0;
15 }
```

main: x = 5, y = 3

swap: a = 3, b = 5, temp = 5

3.

```
1  void swap(int a, int b)
2  {
3    int temp = a;
4    a = b;
5    b = temp;
6  }
7
8  int main(void)
9  {
10   int x = 5, y = 3;
11
12   swap(x, y);
13
14   return 0;
15 }
```

main: x = 5, y = 3

4.

Пример 3

```
#include <stdio.h>

void redouble(int a)
{
    a *= 2;
}
```

```
int main(void)
{
    int a = 5;

    printf("Before redouble:"
                    " %d\n", a);

    redouble(a);

    printf("After redouble:"
                    " %d\n", a);

    return 0;
}
```

Выполнение программы

```
1  void redouble(int a)
2  {
3    a *= 2;
4  }
5
6  int main(void)
7  {
8    int a = 5;
9
10   redouble(a);
11
12   return 0;
13 }
```

main: a (int) = 5

1.

```
1  void redouble(int a)
2  {
3    a *= 2;
4  }
5
6  int main(void)
7  {
8    int a = 5;
9
10   redouble(a);
11
12   return 0;
13 }
```

main: a (int) = 5

redouble: a (int) = 10

3.

```
1  void redouble(int a)
2  {
3    a *= 2;
4  }
5
6  int main(void)
7  {
8    int a = 5;
9
10   redouble(a);
11
12   return 0;
13 }
```

main: a (int) = 5

redouble: a (int) = 5

2.

```
1  void redouble(int a)
2  {
3    a *= 2;
4  }
5
6  int main(void)
7  {
8    int a = 5;
9
10   redouble(a);
11
12   return 0;
13 }
```

main: a (int) = 5

4.

Выводы

1. Параметры-переменные и локальные переменные не отличаются только одним: параметрам-переменным автоматически присваиваются начальные значения, равные значениям в точке вызова функции.
2. В функции изменяется параметр-переменная, а не переменная, переданная в качестве аргумента. Функция «забывает» о переменной-аргументе сразу же после инициализации параметра-переменной.
3. Имена параметров-переменных могут совпадать с именами переменных, передаваемых в качестве аргументов.

Аргументы функции

Для решений этой проблемы необходимо каким-то образом предоставить функции доступ к переменным, которые передаются ей в качестве аргументов.

Это можно сделать, передав в функцию не значения переменных, а адреса, по которым эти переменные располагаются в памяти.

Указатели (введение)

Переменная-указатель – это переменная, которая содержит адрес.

Переменная-указатель описывается как и обычная переменная. Единственное отличие – перед ее именем необходимо указать символ «*»:

```
int *p;
```

Это определение сообщает компилятору, что p – это переменная-указатель, которая может указывать на переменные целого типа.

Указатели (базовые операции)

В языке Си есть две операции, которые предназначены для использования именно с указателями.

Чтобы узнать адрес переменной, необходимо воспользоваться *операцией получения адреса* «&». Если x – это переменная, то &x – адрес переменной x в памяти.

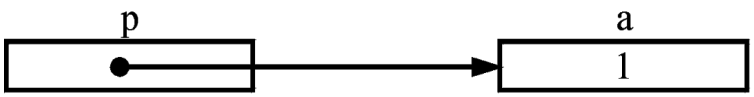
Чтобы получить доступ к объекту, на который указывает указатель, необходимо использовать *операцию разыменования* «*». Если p – переменная-указатель, то *p представляет объект, на который сейчас указывает p.

Указатели (базовые операции)

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
`&`	Адрес	`&X`	Префиксная	15	Справа налево
`*`	Разыменование	`*X`			

Пример использования базовых операций

```
int a = 1;
int *p = &a;            // p теперь указывает на a
```




```
a = 3;

printf("%d %d\n", a, *p); // 3 3

*p = 5;

printf("%d %d\n", a, *p); // 5 5

printf("%p\n", p);     // адрес переменной a
```

Операция разыменования

Справа от операции присваивания

```
int a = 5, b;
int *p = &a;
b = *p;
```

«Взять» значение переменной, адрес которой хранится в указателе p.

Слева от операции присваивания

```
int a = 5, b;
int *p = &b;
*p = b;
```

«Положить» значение в переменную, адрес которой хранится в указателе p.

Аргументы функции: использование указателей

```
#include <stdio.h>

void swap(int *pa, int *pb)
{
    int temp = *pa;

    *pa = *pb;
    *pb = temp;
}
```

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

    printf("Before swap: %d,"
                  " %d\n", x, y);

    swap(&x, &y);

    printf("After swap: %d,"
                  " %d\n", x, y);

    return 0;
}
```

Выполнение программы

```
1  void swap(int *pa, int *pb)
2  {
3    int temp = *pa;
4    *pa = *pb;
5    *pb = temp;
6  }
7
8  int main(void)
9  {
10   int a = 5, b = 3;
11
12   swap(&a, &b);
13
14   return 0;
15 }
```

main: a (int) 5; b (int) 3

1.

```
1  void swap(int *pa, int *pb)
2  {
3    int temp = *pa;
4    *pa = *pb;
5    *pb = temp;
6  }
7
8  int main(void)
9  {
10   int a = 5, b = 3;
11
12   swap(&a, &b);
13
14   return 0;
15 }
```

main: a (int) 5; b (int) 3

swap: pa (pointer); pb (pointer); temp (int) 5

2.

```
1  void swap(int *pa, int *pb)
2  {
3    int temp = *pa;
4    *pa = *pb;
5    *pb = temp;
6  }
7
8  int main(void)
9  {
10   int a = 5, b = 3;
11
12   swap(&a, &b);
13
14   return 0;
15 }
```

main: a (int) 3; b (int) 5

swap: pa (pointer); pb (pointer); temp (int) 5

3.

```
1  void swap(int *pa, int *pb)
2  {
3    int temp = *pa;
4    *pa = *pb;
5    *pb = temp;
6  }
7
8  int main(void)
9  {
10   int a = 5, b = 3;
11
12   swap(&a, &b);
13
14   return 0;
15 }
```

main: a (int) 3; b (int) 5

4.

Аргументы функции: использование указателей

```
#include <stdio.h>

void decompose(
       float f,
         int *int_part,
            float *frac_part)
{
  *int_part  = f;
  *frac_part = f - *int_part;
}
```

```
int main(void)
{
  int i;
  float f;

  decompose(3.14159, &i, &f);

  printf("%d %f\n", i, f);

  return 0;
}
```

Рекурсия

Функция называется *рекурсивной*, если она вызывает саму себя.

Например, следующая функция рекурсивно вычисляет факториал, используя формулу n! = n * (n - 1)!

```
int fact(int n)
{
    if (n == 0)
        return 1;

    return n * fact(n - 1);
}
```

Рекурсия

Отобразим последовательность запусков при вызове функции fact(3):

```
fact(3) = 3 * fact(2)      // приостановка выполнения fact(3)
  fact(2) = 2 * fact(1)    // приостановка выполнения fact(2)
    fact(1) = 1 * fact(0)  // приостановка выполнения fact(1)
      fact(0) = 1
    fact(1) = 1            // возобновление выполнения fact(1)
  fact(2) = 2              // возобновление выполнения fact(2)
fact(3) = 6                // возобновление выполнения fact(3)
```

Рекурсия

```
1  #include <stdio.h>
2
3  int fact(int n)
4  {
5      if (n == 0)
6          return 1;
7
8      return n * fact(n - 1);
9  }
10
11 int main(void)
12 {
13     int n = 3;
14
15     printf("%d! = %d\n", n, fact(n));
16
17     return 0;
18 }
```

main
n int 3

fact
n int 3

fact
n int 2

fact
n int 1

fact
n int 0

Рекурсия

В рекурсии простейшей формы рекурсивный вызов расположен в конце функции. Такая рекурсия называется *хвостовой*.

Хвостовая рекурсия является простейшей формой рекурсии, поскольку она действует подобно циклу.

В большинстве случаев при реализации предпочтение отдается циклу:

- рекурсивный вызов использует больше памяти, поскольку создает свой набор переменных.
- рекурсия выполняется медленней, поскольку на каждый вызов функции требуется определенное время.

Литература

1. А. Богатырев «Руководство полного идиота по программированию (на языке Си)» (главы "Функции", "Как происходит вызов функции", "Указатели", "Использование указателей")
2. С. Прата «Язык программирования Си» (глава 9)
3. Б. Керниган, Д. Ритчи «Язык программирования Си» (разделы 4.1, 4.2, 4.10, 5.1, 5.2)
4. Черновик стандарта C99

29