Саркисян Г.Ф.

ПРОГРАММИРОВАНИЕ

Часть I: Структурное программирование

ПЕРЕДАЧА ПАРАМЕТРОВ В ФУНКЦИЮ ПО ЗНАЧЕНИЮ

ПЕРЕДАЧА ПАРАМЕТРОВ В ФУНКЦИЮ ПО ЗНАЧЕНИЮ

Основным способом обмена информацией между **вызываемой** и **вызывающей** функциями является **механизм передачи параметров**.

Существует **два способа** передачи параметров в функцию: **по адресу** и **по значению**.

ПЕРЕДАЧА ПАРАМЕТРОВ В ФУНКЦИЮ ПО ЗНАЧЕНИЮ

При передаче по значению выполняются следующие действия:

1. вычисляются значения выражений, стоящие на месте фактических параметров;

2. в стеке выделяется память под формальные параметры функции;

3. каждому формальному параметру присваивается значение фактического параметра, при этом проверяются соответствия типов и при необходимости выполняются их преобразования.

ПЕРЕДАЧА ПАРАМЕТРОВ В ФУНКЦИЮ ПО ЗНАЧЕНИЮ

При передаче параметров в функцию по значению, вызваемая функция не имеет непосредственной возможности изменить переменную из вызывающей функции.

ПРИМЕРЫ ЗАДАЧ

ПРИМЕРЫ ЗАДАЧ

1.
```
#include<stdio.h>

void kvadrat( int );

void main( void){
     int x;
     printf("x="); scanf("%d",&x);
     kvadrat(x);
}

void kvadrat( int a){
     printf("%d^2=%d\n", a, a*a);
}
```

ПРИМЕРЫ ЗАДАЧ

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

int Summa(int, int);
void Proizv(int, int);
double Delenie(int, int);

void main( void ){
     int x = 5,y = 2, sum;
     double del;
     sum = Summa(x,y);
     printf("%d+%d=%d\n", x,y,sum);
     Proizv(sum,x);
     del = Delenie(x,y);
     printf("%d/%d=%.2lf\n", x,y,del);
}
```

ПРИМЕРЫ ЗАДАЧ

```
int Summa( int a, int b){
     return a+b;
}

void Proizv( int a, int b){
     printf("%d*%d=%d\n", a,b,a*b);
}

double Delenie( int a, int b){
     return ( double )a/b;
}
```

ПРИМЕРЫ ЗАДАЧ

```
3. #include <stdio.h>
int max_3( int, int, int);

void main( void ){
     int x1,x2,x3, max;
     scanf("%d%d%d",&x1,&x2,&x3);

     max = max_3(x1,x2,x3);

printf("%d %d %d max = %d\n", x1,x2,x3,max);


printf("%d %d %d max=%d\n", max_3(x1+x2,max,x3));
}
```

ПРИМЕРЫ ЗАДАЧ

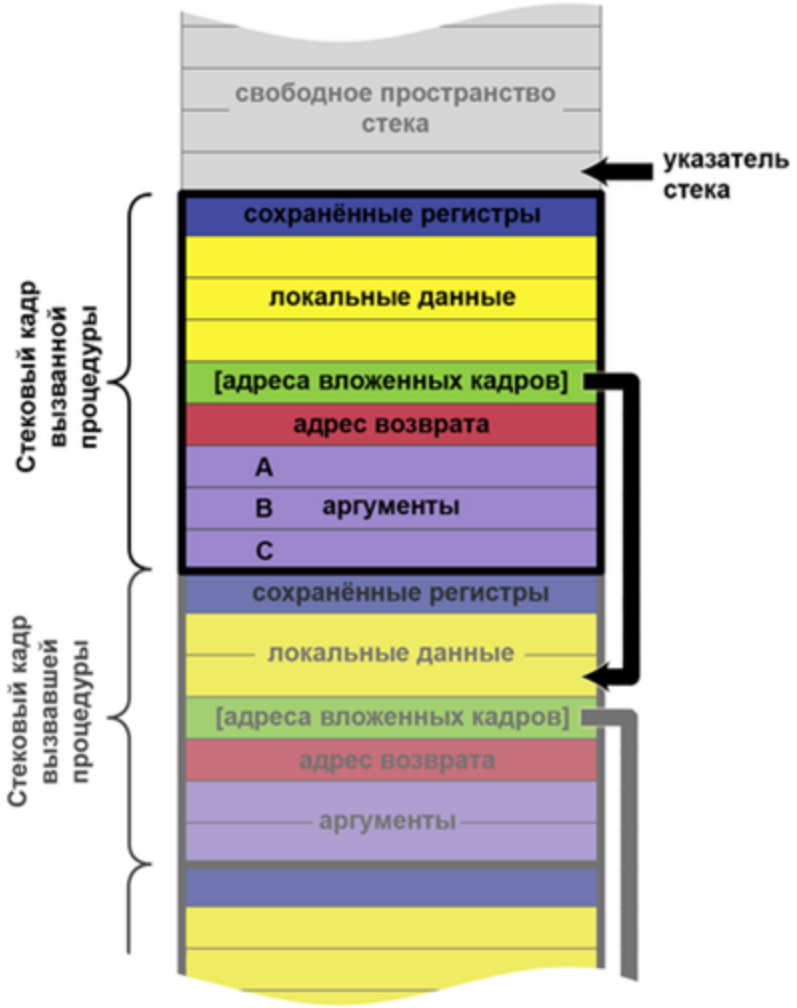
```
int max_3( int a, int b, int c){
      int max = a;

      if (max < b)            max = b;

      if (max < c)            max = c;

      return max;
}
```

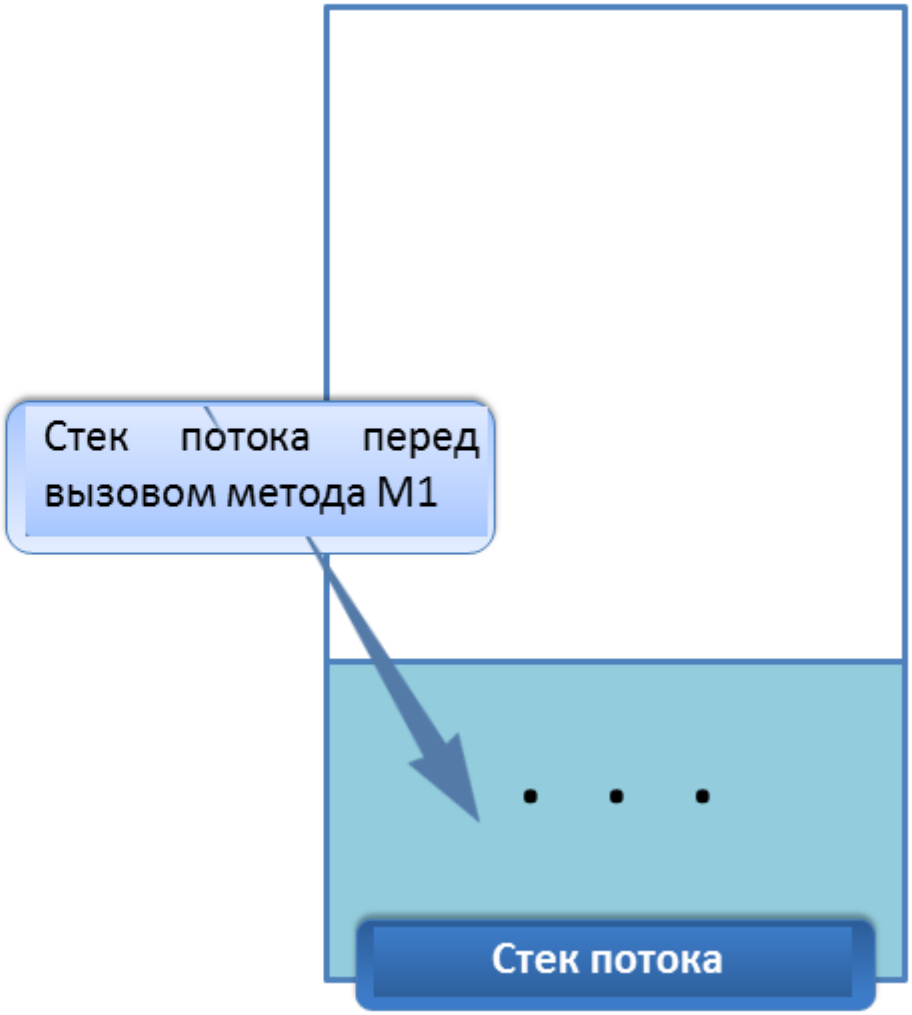
Стековый кадр



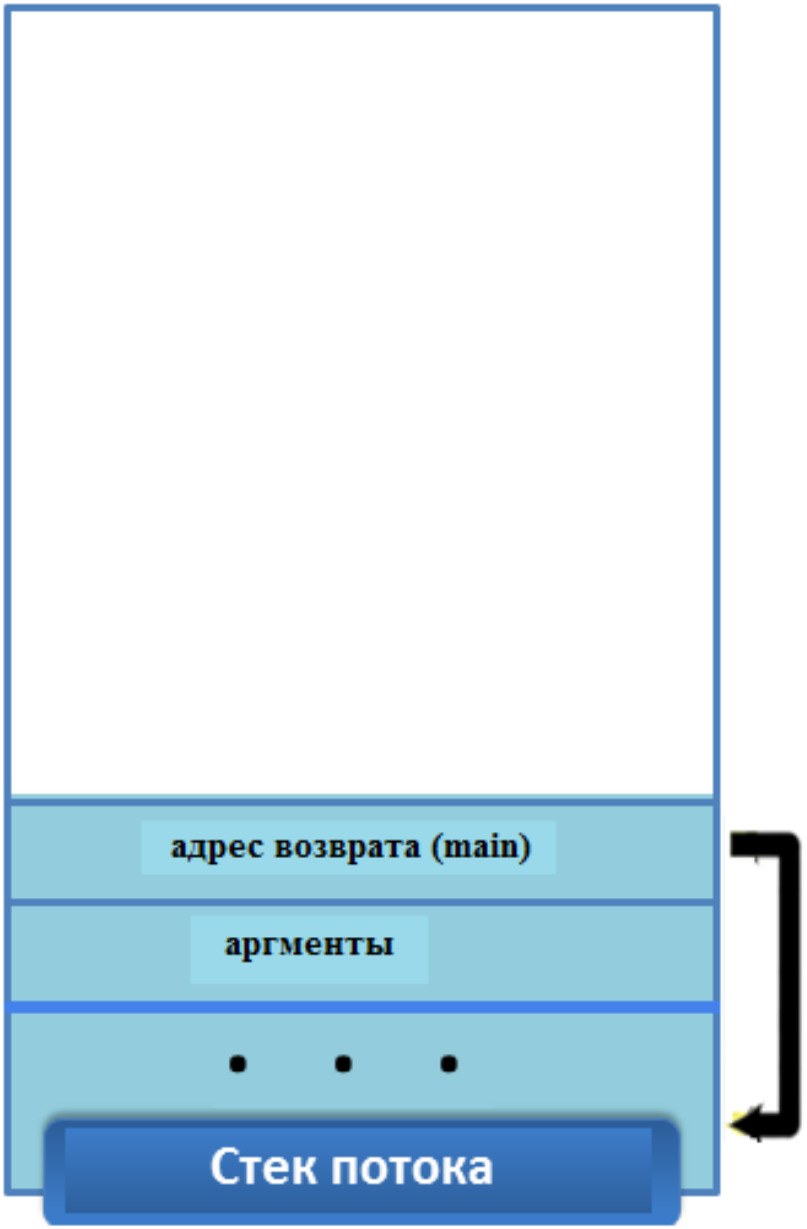
Стековый кадр — механизм передачи аргументов и выделения временной памяти (в процедурах языков программирования высокого уровня) с использованием системного стека

Обычно системный стек используется для сохранения адресов возврата при вызове подпрограмм, а также сохранения/восстановления значений регистров процессора

```
void M1()
{
    char name[4] = "Joe";
     M2(name);
     . . .
     return;
}
```

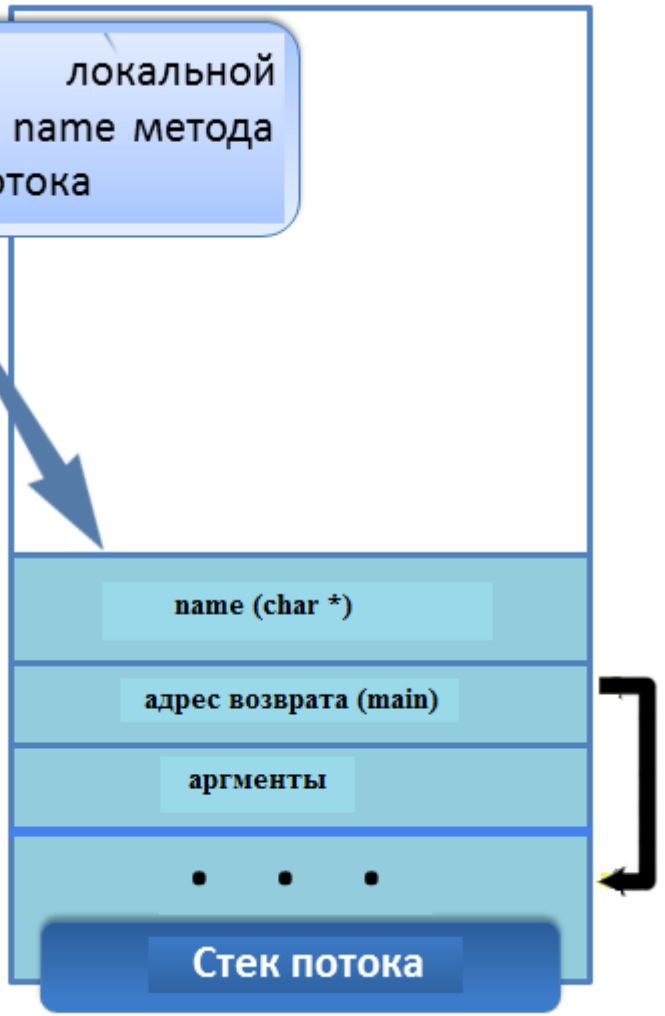


```
void M1()
{
    char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```

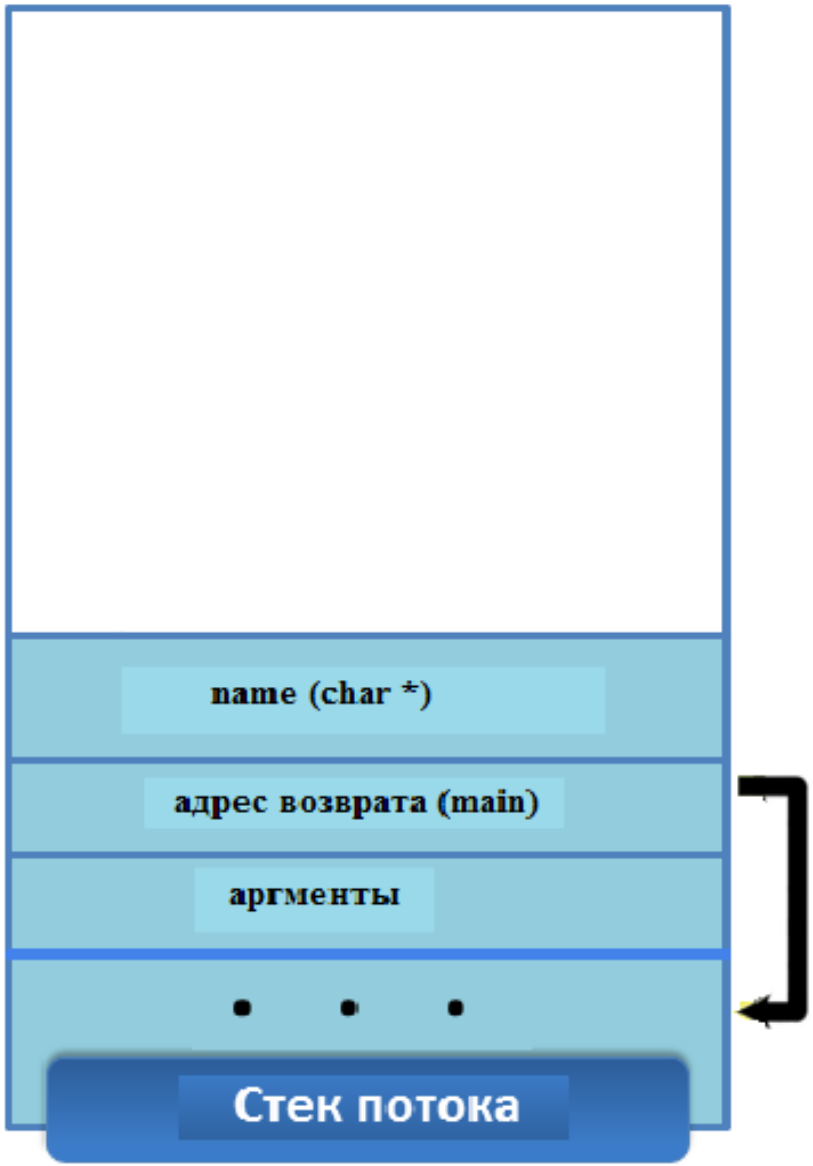


```
void M1()
{
   char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```

Размещение локальной переменной name метода M1 в стеке потока

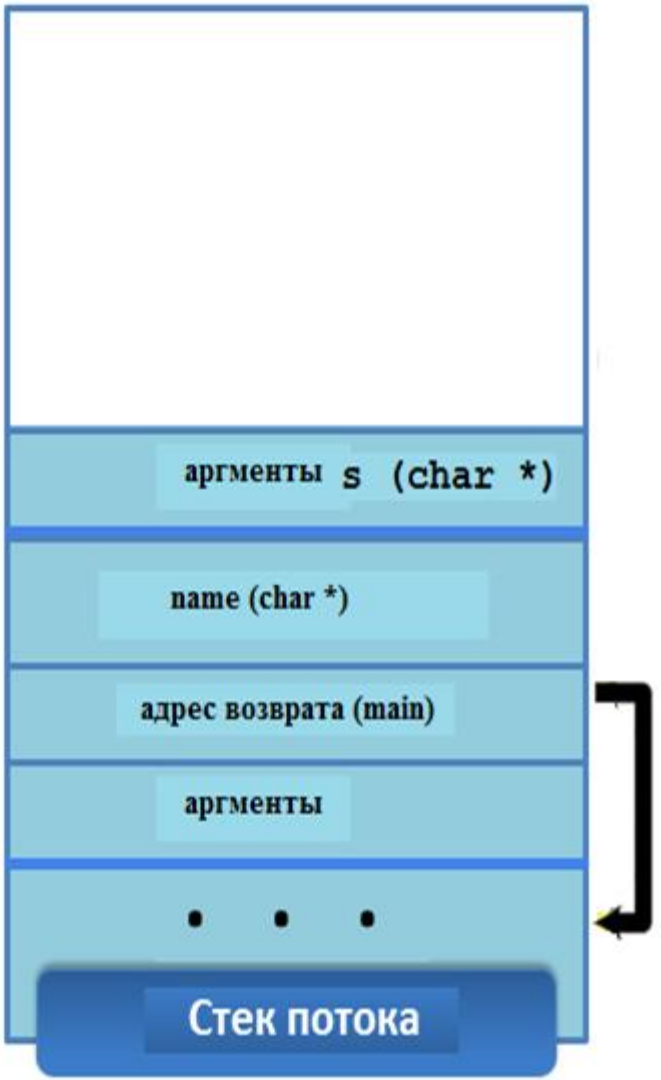


```
void M1()
{
   char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```



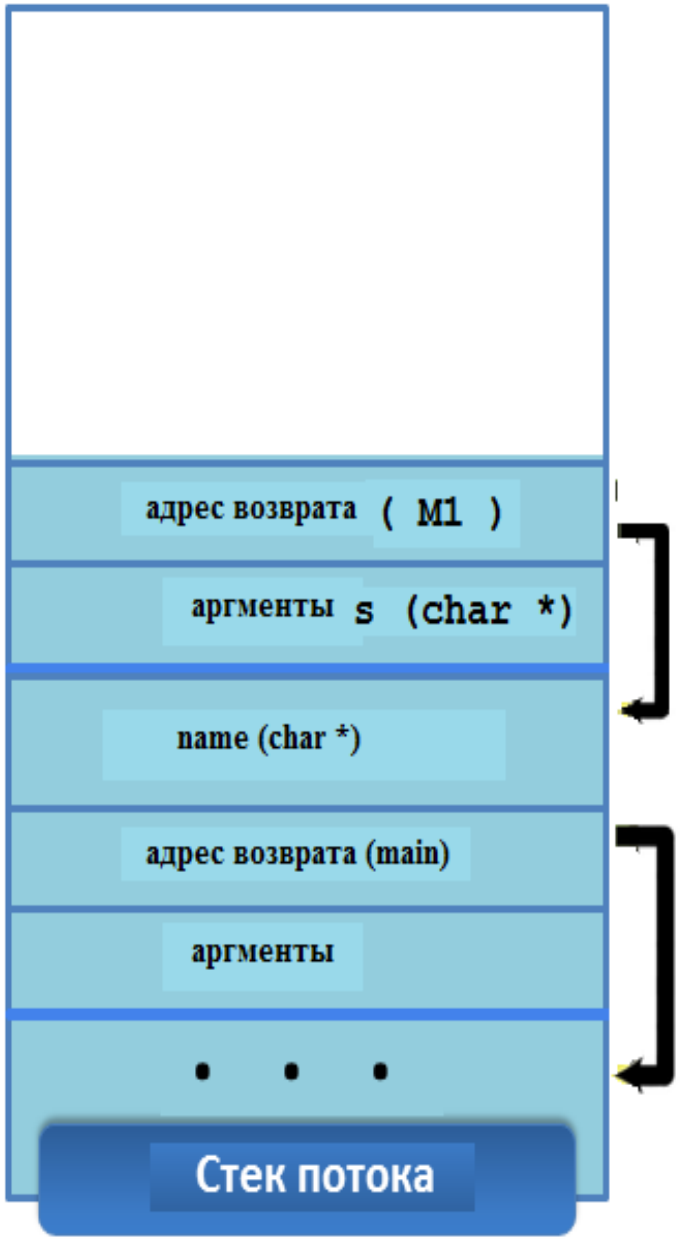
```
void M1()
{
   char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```

```
void M2( char * s )
{
    int length = s.Length;
    int tally;
    . . .
    return;
}
```



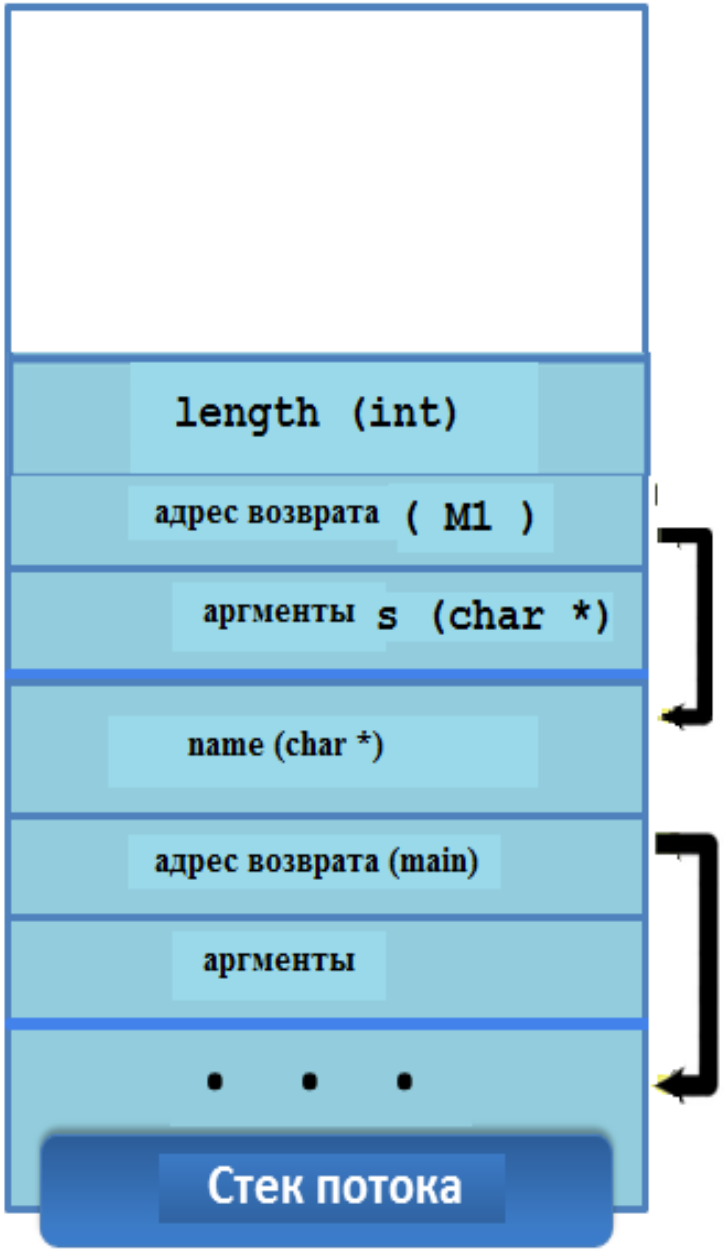
```
void M1()
{
   char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```

```
void M2 (char * s)
{
    int length = s.Length;
    int tally;
    . . .
    return;
}
```



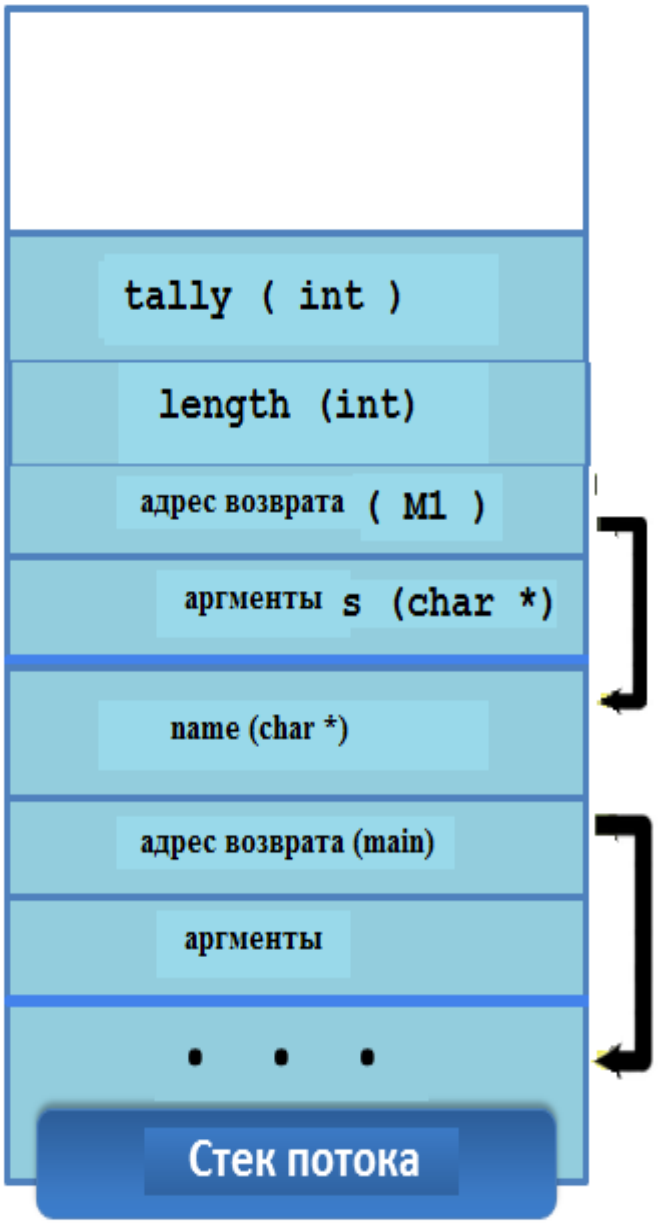
```
void M1()
{
   char name[41 = "Joe";
    M2(name);
    . . .
    return;
}
```

```
void M2 (char * s)
{
    int length = s.Length;
    int tally;
    . . .
    return;
}
```



```
void M1()
{
   char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```

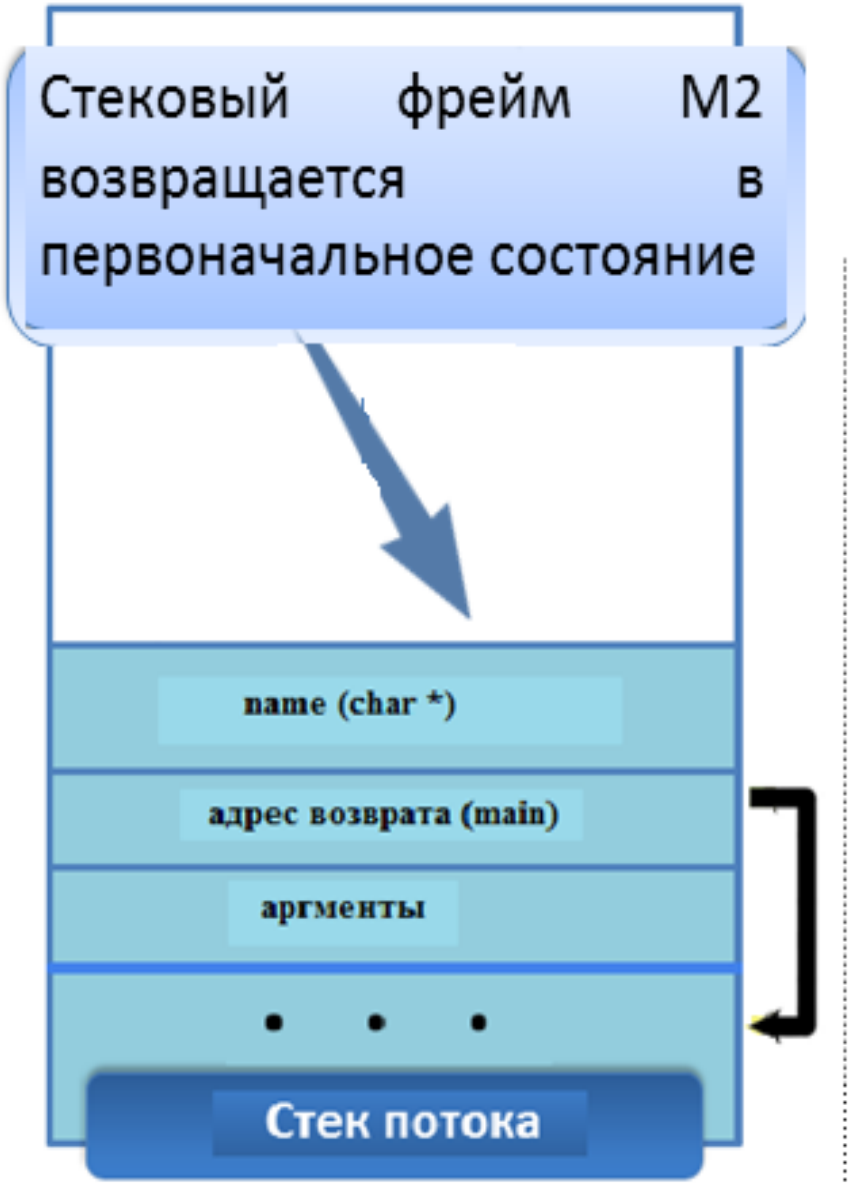
```
void M2 (char * s)
{
    int length = s.Length;
    int tally;
    . . .
    return;
}
```



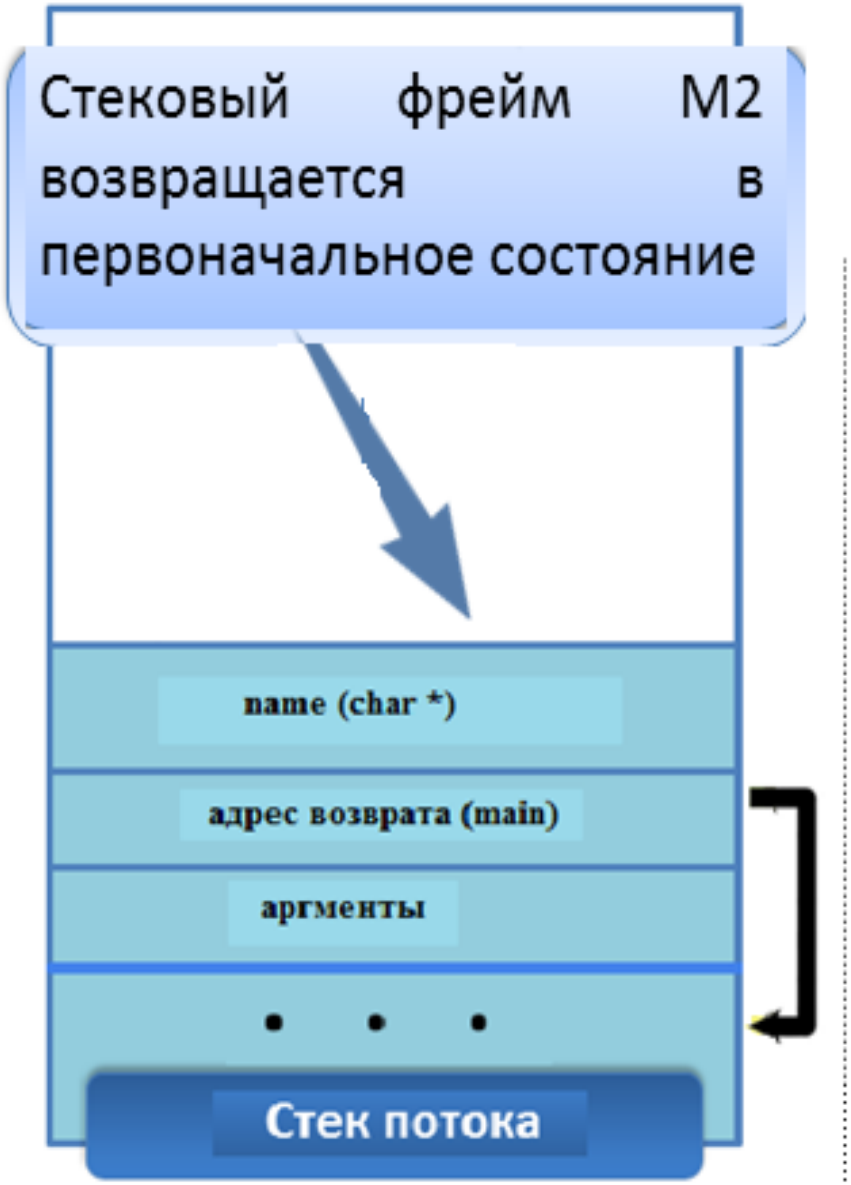
```
void M1()
{
   char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```

```
void M2 (char * s)
{
    int length = s.Length;
    int tally;
    . . .
    return;
}
```

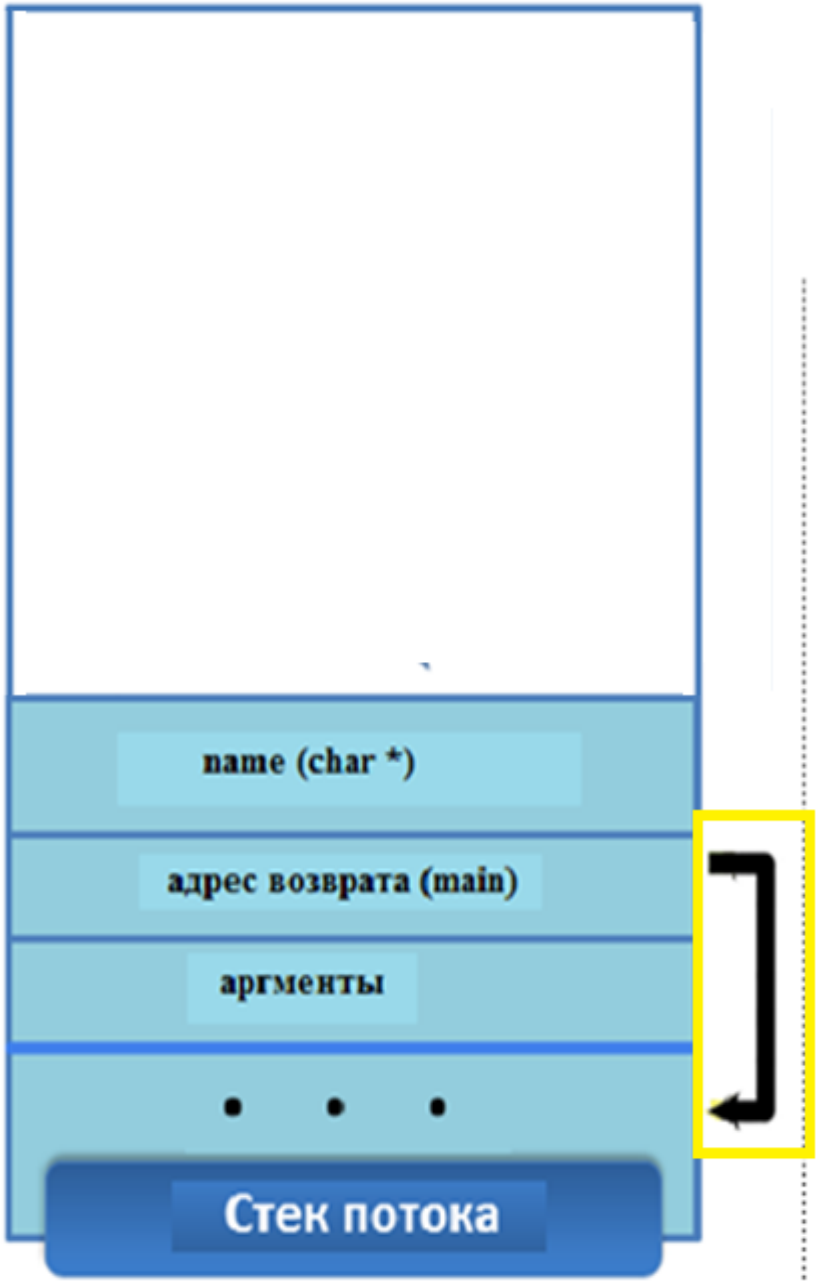
```
void M1()
{
   char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```



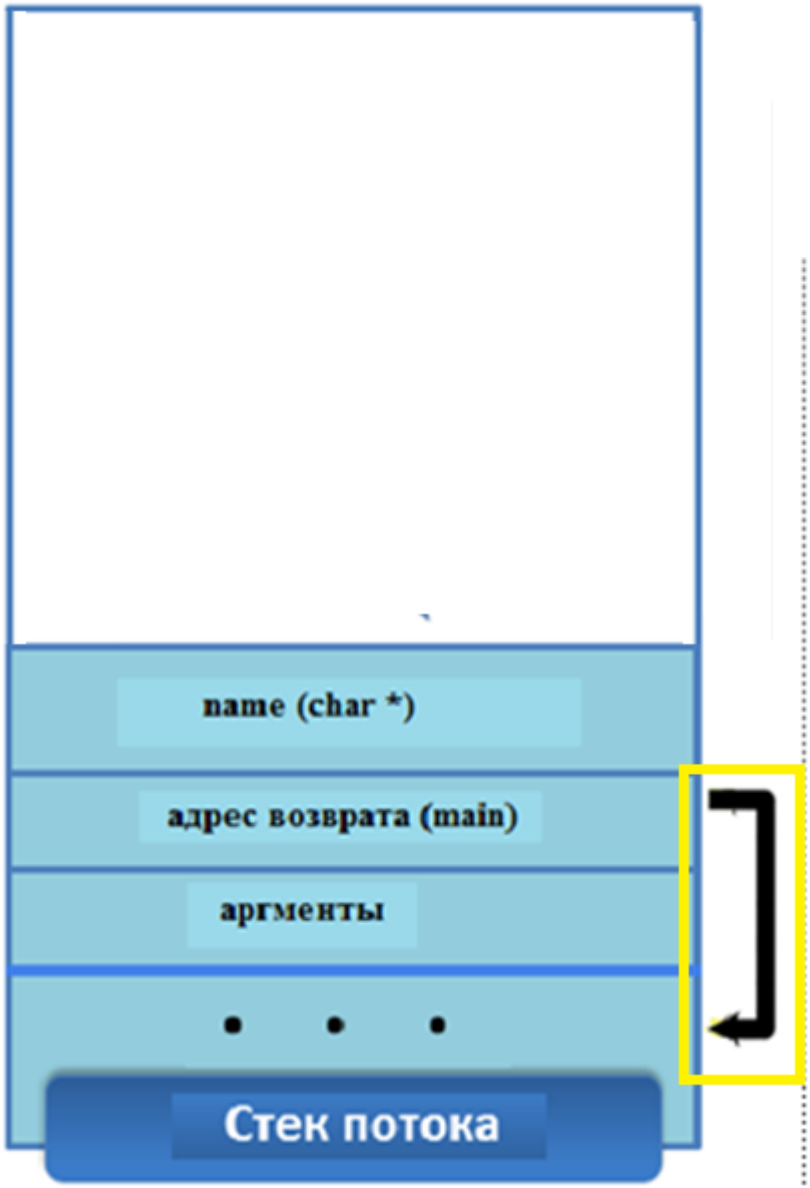
```
void M1()
{
   char name[4] = "Joe";
    M2(name);
    . . .
    return;
}
```



```
void M1()
{
   char name[4] = "Joe";
      M2(name);
      . . .
      return;
}
```

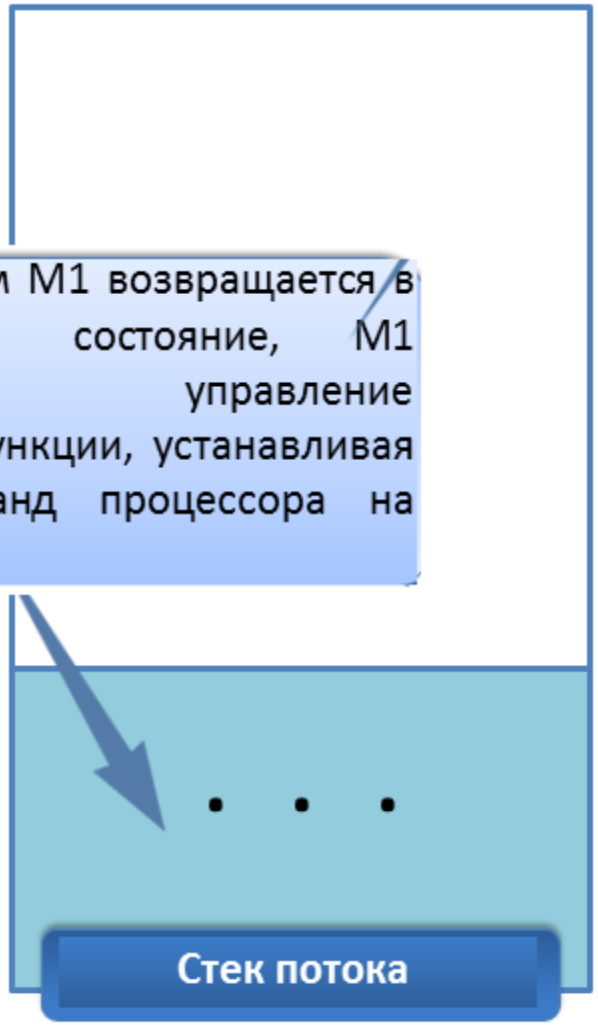


```
void M1()
{
   char name[4] = "Joe";
     M2(name);
     . . .
    return;
}
```



```
void   main ( )
{    . . .
     M1() ;
     . . .
     return;
}
```

Стековый фрейм M1 возвращается в первоначальное состояние, M1 возвращает управление вызывающей функции, устанавливая указатель команд процессора на адрес возврата



ПРИМЕРЫ ЗАДАЧ

Функция напрямую, т.е. с помощью оператора перехода **`return`**, может вернуть только одно значение.

Вернуть несколько значений из функции невозможно, но можно возвращать не значения, а записать эти значение сразу в ячейки локальной памяти.

Для этого необходимо передавать параметры в функцию не по значению, а **по адресу**.

Передать **адрес переменной** можно с помощью **указателя** или по средствам **ссылки**.

ПОНЯТИЕ УКАЗАТЕЛЯ

ПОНЯТИЕ УКАЗАТЕЛЯ

Указатель – это переменная, в которой хранится адрес области памяти.

Указатель не является самостоятельным типом, он всегда связан с каким-то другим типом.

Указатели делятся на две категории:

1. *указатели на объекты (или переменные);*
2. *указатели на функции*.

ПОНЯТИЕ УКАЗАТЕЛЯ

Определение указателей на переменные имеет вид:

```
Type *Point_ID;
```

Type – это тип, на который будет ссылаться указатель. **Тип может быть любым, кроме ссылки.**

Когда компилятор обрабатывает оператор **определения указателя**, то в оперативной памяти под **имя** (**Point_ID**) *указателя* выделяется место как для целочисленной переменной, т.к. адрес области памяти – это целое число.

ПОНЯТИЕ УКАЗАТЕЛЯ

***** – определяет тип указателя;

имя (Point_ID) – это переменная, которой хранится адрес переменной (ячейки), на которую будет указывать указатель (всегда целого типа);

***имя (Point_ID)** – это содержимое адреса, на который указывает указатель (т.е. переменная, которая соответствует типу на который указывает указатель).

ПОНЯТИЕ УКАЗАТЕЛЯ

```
char* pc;

int* pi;

double* pd;

sizeof(pc) == sizeof(pi)== sizeof(pd)==
sizeof(int)

sizeof(*pc) == sizeof(char)

sizeof(*pi) == sizeof(int)

sizeof(*pd) == sizeof(double)
```

ПОНЯТИЕ УКАЗАТЕЛЯ

```
int *i;  //i хранит адрес ячейки с целым числом

int *x, y;  // x хранит адрес, а y – целое число!

double *f, *ff;// два указателя на ячейки double

char *c;  // указатель на ячейку, хранящую символ
```

ПОНЯТИЕ УКАЗАТЕЛЯ

Некоторые способы определения указателей:

```
int* pi;
```
// указатель на целую переменную.

```
const int* pci;
```
// указатель на целую константу.

```
int* const cpi;
```
// указатель-константа на переменную целого типа.

```
const int* const cpc;
```
// указатель-константа на целую константу.

```
int** a;
```
//который отличается от простого указателя тем, что хранит адрес ячейки, которая сама хранит адрес некоего объекта в памяти.

ПОНЯТИЕ УКАЗАТЕЛЯ

Если **модификатор** `const` находится:

1. слева от звездочки, константным является то, на что ссылается указатель и он запрещает изменение значения, на которое ссылается указатель.

2. справа, то сам указатель является константным, это запрещает изменение значения самого указателя.

3. с обеих сторон, то константно и то, и другое.

ОСНОВНЫЕ ОПЕРАЦИИ

ОСНОВНЫЕ ОПЕРАЦИИ

Основные операции, связанные с применением указателей:

1. инициализация;
2. присваивание;
3. чтения и записи по адресу;
4. арифметические операции;
5. операция сравнения.

ОСНОВНЫЕ ОПЕРАЦИИ. ИНИЦИАЛИЗАЦИИ УКАЗАТЕЛЯ

I. ИНИЦИАЛИЗАЦИИ УКАЗАТЕЛЯ

1. С помощью операции получения адреса

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

Символ '&', стоящий перед именем переменной, означает операцию получения адреса этой переменной.

ОСНОВНЫЕ ОПЕРАЦИИ. ИНИЦИАЛИЗАЦИИ УКАЗАТЕЛЯ

Операция **&** применима только к переменным и элементам массива.

Конструкции вида **&(x-1)** и **&3** являются незаконными.

Нельзя также получить адрес регистровой переменной.

ОСНОВНЫЕ ОПЕРАЦИИ. ИНИЦИАЛИЗАЦИИ УКАЗАТЕЛЯ

2. С помощью проинициализированного указателя

```
int a = 5;
int* p = &a;
int* r = p;
```

3. Адрес присваивается в явном виде

```
char* cp = (char*)0x B800 0000;
```

`0X B800 0000` – шестнадцатеричная константа,

`(char*)` – операция приведения типа.

ОСНОВНЫЕ ОПЕРАЦИИ. ИНИЦИАЛИЗАЦИИ УКАЗАТЕЛЯ

4. Присваивание пустого значения:

```
int* N=NULL;

int* R=0;
```

Указатель рекомендуется инициализировать сразу при определении либо адресом переменной, либо пустым значением.

ОСНОВНЫЕ ОПЕРАЦИИ. ОПЕРАЦИЯ ПРИСВАИВАНИЯ

II.ОПЕРАЦИЯ ПРИСВАИВАНИЯ

Переменные различных типов можно присвоить друг другу, при этом происходит неявное приведение типов (т.е. при увеличении размерности значение переменной сохраняется, а при уменьшении разрядности переменной происходит потеря данных или переполнение).

Указатели различных типов, тоже можно присвоить друг другу, но при этом необходимо произвести явное приведение типов.

III. ОПЕРАЦИЯ ЧТЕНИЯ И ЗАПИСИ ПО АДРЕСУ

Операция ‘*’ –это унарная операция разыменования адреса.

Операция `*Point_ID` – это содержимое памяти ПК по адресу, на который ссылается указатель `Point_ID`.

ОСНОВНЫЕ ОПЕРАЦИИ. ОПЕРАЦИЯ ЧТЕНИЯ И ЗАПИСИ ПО АДРЕСУ

Если операция * `Point_ID` находится в

1. **левой части** операции присваивания, то выполняется **запись по данному адресу**;
2. **правой части** операции присваивания, то выполняется **операция чтения по адресу**.

Если в первом случае данные могут быть испорчены (опасная операция) , то во втором – нет.

ОСНОВНЫЕ ОПЕРАЦИИ. ОПЕРАЦИЯ ЧТЕНИЯ И ЗАПИСИ ПО АДРЕСУ

```
double x = 67,*px,  y;
px = &x;
printf("\n%.2lf\n",*px);

*px = *px + 3;
printf("%.2lf\n",x);
```

Если указатель **p** указывает на переменную **x**, то

px – это адрес переменной x, т.е. **px == &x**

***px** – это другое имя переменной **x**

IV. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Арифметические операции применимы только к указателям **одного типа**:

1. Инкремент *увеличивает значение указателя на величину размера типа.*

```
char* pc;     int* pi;     double* pd;
    .....
pc++;  //значение увеличится на 1, т.е. sizeof ( char)
pi++;  //значение увеличится на 4 или 2, т.е.  sizeof ( int)
pd++;  //значение увеличится на 8, т.е. sizeof ( double)
```

ОСНОВНЫЕ ОПЕРАЦИИ. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

2. ***Декремент*** уменьшает значение указателя на величину размера типа*.*

3. ***Разность двух указателей*** *– это* разность их значений, деленная на размер типа в байтах*.*

4. ***Суммирование*** *двух указателей* ***не допускается****.*

5. Можно ***суммировать указатель со значение целого типа.***

V. ОПЕРАЦИИ СРАВНЕНИЯ

Операции сравнения применимы только к указателям одного типа.

Результат операции сравнения целое число. Если это число 1 – это **`true`**, в остальных случаях **`false`**.

Результат операции “==” будет **`true`** если указатели указывают на одну и туже область памяти.

ОСНОВНЫЕ ОПЕРАЦИИ. ОПЕРАЦИИ СРАВНЕНИЯ

Результат операции “>=” будет **true** если переменная на которую указывает первый указатель расположен в памяти под более старшим или тем же адресом, чем переменная на которую указывает второй указатель.

Результат операции “<” будет **true** если переменная на которую указывает первый указатель расположен в памяти под более младшим или тем же адресом, чем переменная на которую указывает второй указатель и т.д.

ОСНОВНЫЕ ОПЕРАЦИИ

Примеры:

1.

```
int x = 5 , y = 7, *p1= &x, *p2= &y;
*p1 = 10;       *p2 = 20;
printf("%d %d ", x, y);

p1 = p2;
printf("%d %d ", *p1, *p2);

*p1 = 30;
printf("%d %d ", *p1, *p2);

printf("%d %d ", x, y);
```

ОСНОВНЫЕ ОПЕРАЦИИ

2. Пример передачи параметра в функцию по адресу

```
void fff(int, int, int*, int*, int*, double*);

void main(void){
      int x = 5 , y = 2, sum, pr, raz;
      double del;

      fff( x, y, &sum, &pr, &raz, &del);

      printf("%d + %d=%d\n", x, y, sum);
      printf("%d - %d=%d\n", x, y, raz);
      printf("%d * %d=%d\n", x, y, pr);
      printf("%d / %d=%.2lf\n", x, y, del);
}
```

ОСНОВНЫЕ ОПЕРАЦИИ

```
void fff(int a, int b, int*p1, int*p2, int*p3,
                                    double*p4){
      *p1 = a+b;
      *p2 = a*b;
      *p3 = a-b;
      *p4 = (double)a/b;
}
```

ОСНОВНЫЕ ОПЕРАЦИИ

3.

```
void F(int*, int, int*);
void main(void){
      int a = 10, b = 20, c = 30;

      F(&a, b, &c);
      printf("%d %d %d\n",a,b,c);
}
void F(int*x, int y, int* z) {
      *x = 1;
       y = 2;
      *z = 3;
}
```

ОСОБЫЙ ТИП УКАЗАТЕЛЯ VOID*

ОСОБЫЙ ТИП УКАЗАТЕЛЯ VOID*

Особый тип указателя `void*`

1. определяет адрес некоторого объекта, но *не содержит* информации *о типе* объекта.

2. чтобы просмотреть содержание данного адреса, необходимо выполнить операцию приведения указателя к определенному типу.

```
void *pv;
sizeof(*pv) == illegal indirection
sizeof(*(Type*)pv) == sizeof( Type )
```

ОСОБЫЙ ТИП УКАЗАТЕЛЯ VOID*

4.

```
int a = 2;     double b = 78.45;

void* point;

point = &a;

printf("a=%d\n", *(int*)point);


point = &b;

printf("b=%.2lf\n", *(double*)point);
```

ССЫЛКИ

ССЫЛКИ

Ссылка это и есть, в принципе, указатель, только скрытый и является еще одним именем для переменной.

Ссылки должны быть обязательно **инициализированы не нулевым значением**, так как NULL ссылок не бывает, а **ссылка должна содержать адрес, на переменную (объект), на которую она будет ссылаться**.

Ссылка, по сути, является просто вторым именем переменной (объекта).

ССЫЛКИ

```
Type & ID_ссылки = ID_объекта;
```

```
int i = 10;

int &p = i; // определение ссылки

const char & CR='\n'; //определение ссылки
                      // на константу
```

ССЫЛКИ

Правила работы со ссылками:

1. Ссылка должна быть явно проинициализирована при ее описании, кроме следующих двух случаев:

   1.1 если она является параметром функции

   1.2 описана с спецификатором **`extern`**.

2. После инициализации ссылке не может быть присвоено другое значение.

ССЫЛКИ

3. Операция над ссылкой приводит к изменению величины, на которую она ссылается.

4. Не существует указателей на ссылки, массивов ссылок (т.к. под массив выделяется память при компиляции) и ссылок на ссылки.

5. Функции не могут быть перегружены, если описание их параметров отличается только наличием ссылки.

ССЫЛКИ

5.

```
void fff(int, int, int&, int&, int&, double&);
void main(void){
      int x = 5 , y = 2, sum, pr, raz;
      double del;
      fff( x, y, sum, pr, raz, del);
      printf("%d + %d=%d\n", x, y, sum);
      printf("%d -%d=%d\n", x, y, raz);
      printf("%d * %d=%d\n", x, y, pr);
      printf("%d / %d=%.2lf\n", x, y, del);
}
void fff(int a, int b, int&p1, int&p2, int&p3,
                                        double&p4){
      p1 = a+b;   p2 = a*b;   p3 = a-b;
      p4 =(double)a/b;
}
```

СПАСИБО ЗА ВНИМАНИЕ!

ВОПРОСЫ ?

ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

Автор: Саркисян Гаяне Феликсовна