Passagem de parâmetros/argumentos em subprogramas

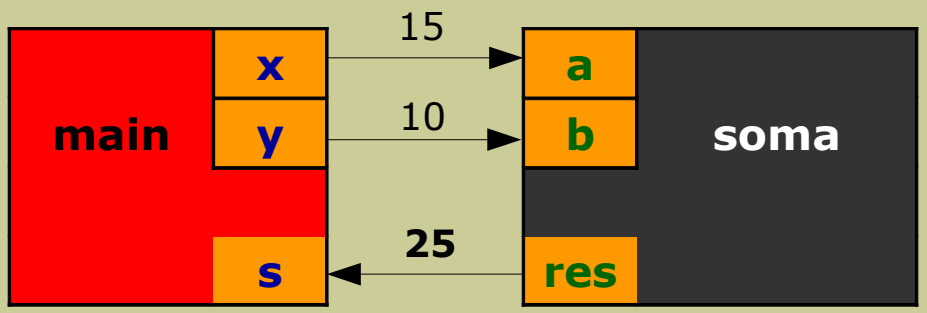
Situação atual sobre comunicação entre subprogramas

Argumentos de um subprograma

- Forma de passar valores para o subprograma (**entrada de dados**)
- Associação de 1 para 1 (1 argumento formal para a 1 argumento efetivo, ambos do mesmo tipo de dados)

Devolução/retorno de valor

- Forma de passar um valor do subprograma para quem o chamou (**saída de dados**)
- Um subprograma **devolve** apenas **um** valor (resultado) ou **nada**
- Uso da instrução **return** para devolver o valor



Resumindo

- Passagem de valores **para** um subprograma (entrada de dados): **0 ou mais**
- Passagem de valores **de** um subprograma (saída de dados): **0 ou 1**

Possível problema

- Será que não é possível a um subprograma passar mais do que um valor?

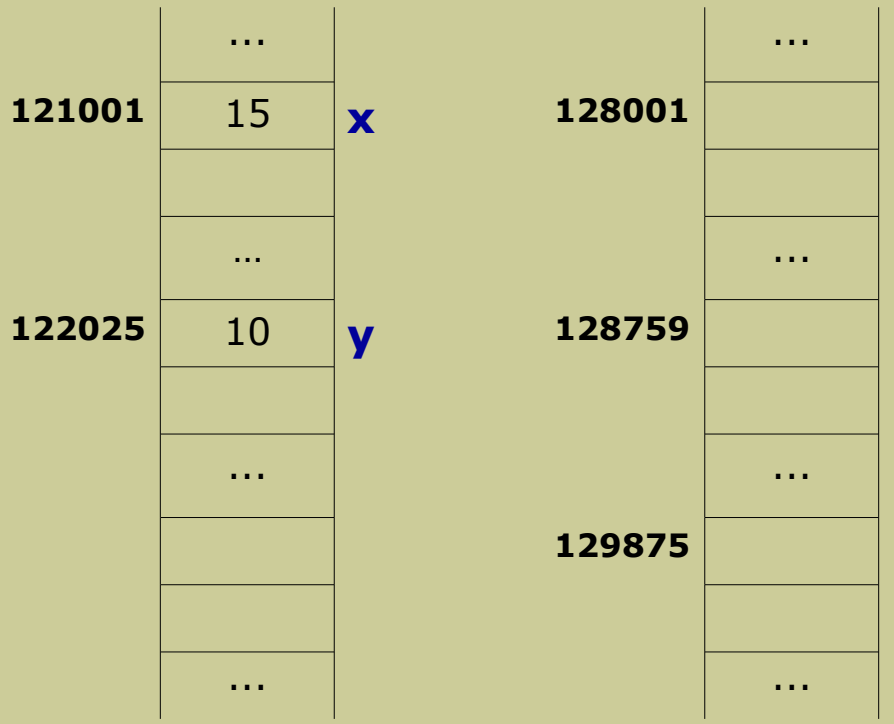
Solução

- Definir dados de entrada que sejam também dados de saída

Passagem de parâmetros/argumentos

Exemplo

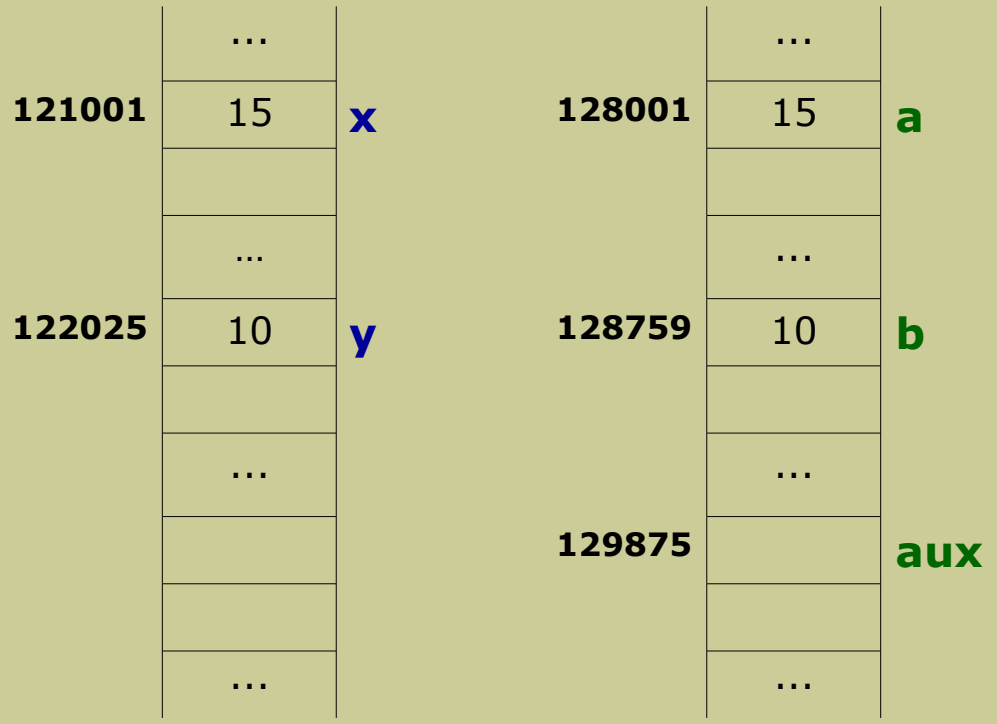
```
#include <stdio.h>
void trocar (int a, int b) {
  int  aux;
  aux = a;
  a = b;
  b = aux;
}
int main() {
  int  x, y;
  x = 15;
  y = 10;
  trocar(x, y);
  printf("x = %d e y = %d\n", x, y);
}
```



estrutura da memória antes da chamada do subprograma **trocar**

Exemplo

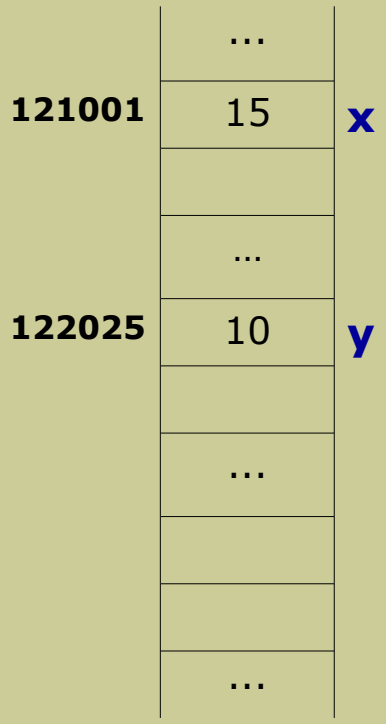
```
#include <stdio.h>
void trocar (int a, int b)
{
  int  aux;
  aux = a;
  a = b;
  b = aux;
}
int main()
{
  int  x, y;
  x = 15;
  y = 10;
  trocar(x, y);
  printf("x = %d e y = %d\n", x, y);
}
```



estrutura da memória com a chamada do subprograma **trocar**

Exemplo

```
#include <stdio.h>
void trocar (int a, int b)
{
  int  aux;
  aux = a;
  a = b;
  b = aux;
}
int main()
{
  int  x, y;
  x = 15;
  y = 10;
  trocar(x, y);
  printf("x = %d e y = %d\n", x, y);
}
```

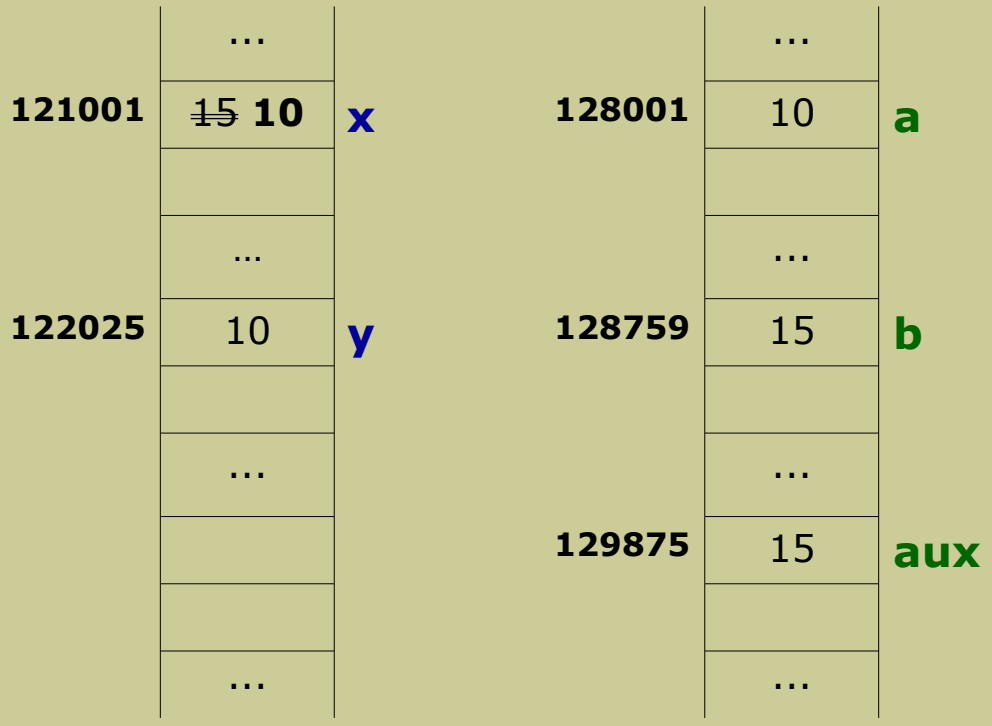


Estrutura da memória com a execução das instruções do subprograma **trocar**

Como passar os valores de **a** e **b** para **x** e **y**?

Exemplo

```
#include <stdio.h>
int trocar (int a, int b)
{
  int  aux;
  aux = a;
  a = b;
  b = aux;
  return a;
}
int main(){
  int  x, y;
  x = 15;
  y = 10;
  x = trocar(x, y);
  printf("x = %d e y = %d\n", x, y);
}
```



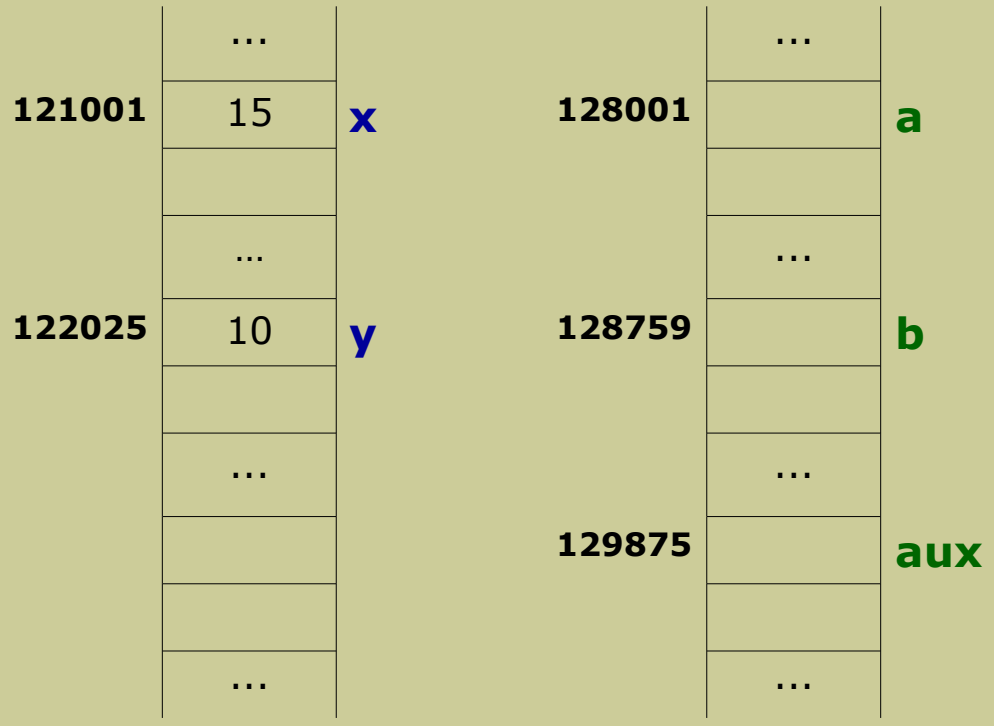
Como passar os valores de **a** e **b** para **x** e **y**?

Solução 1:

- usar o **return** só passa um deles
- mas isto não resolve

Exemplo

```
#include <stdio.h>
void trocar (int a, int b)
{
  int  aux;
  aux = a;
  a = b;
  b = aux;
}
int main()
{
  int  x, y;
  x = 15;
  y = 10;
  trocar(x, y);
  printf("x = %d e y = %d\n", x, y);
}
```



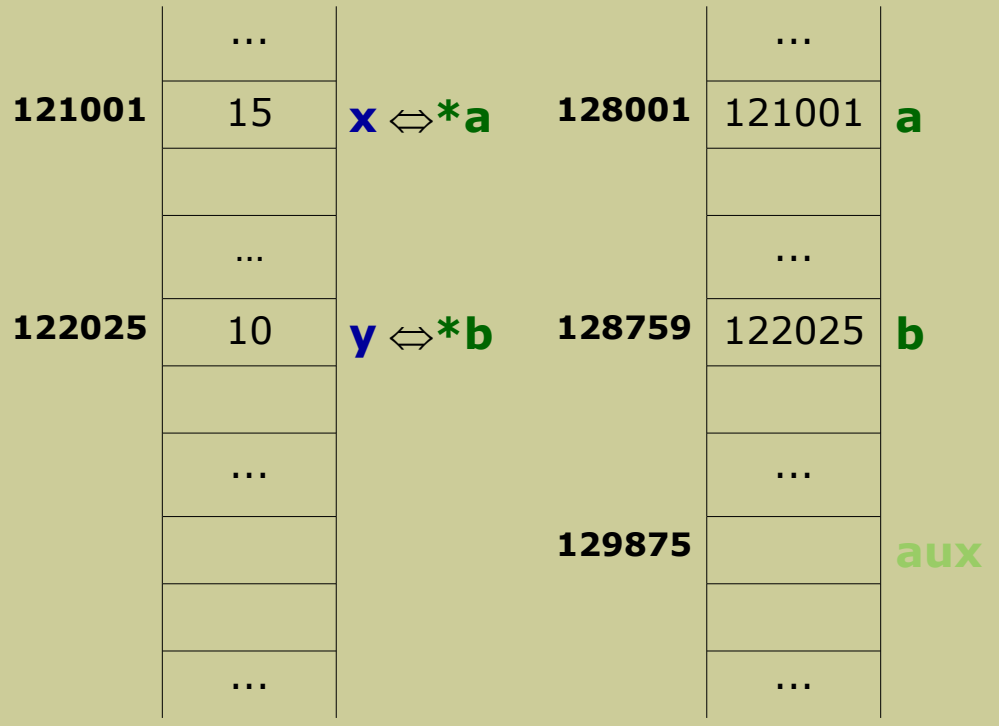
Como passar os valores de **a** e **b** para **x** e **y**?

Solução 2:

O programa principal em vez de enviar os valores de **x** e **y**, por que não enviar os próprios **x** e **y**? **É possível? Como fazer?**

Exemplo

```
#include <stdio.h>
void trocar (int a, int b)
void trocar (int *a, int *b)
{
  int  aux;
  aux = a;
  a = b;
  b = aux;
}
int main(){
  int  x, y;
  x = 15;
  y = 10;
  trocar(x, y);  trocar(&x, &y);
  printf("x = %d e y = %d\n", x, y);
}
```

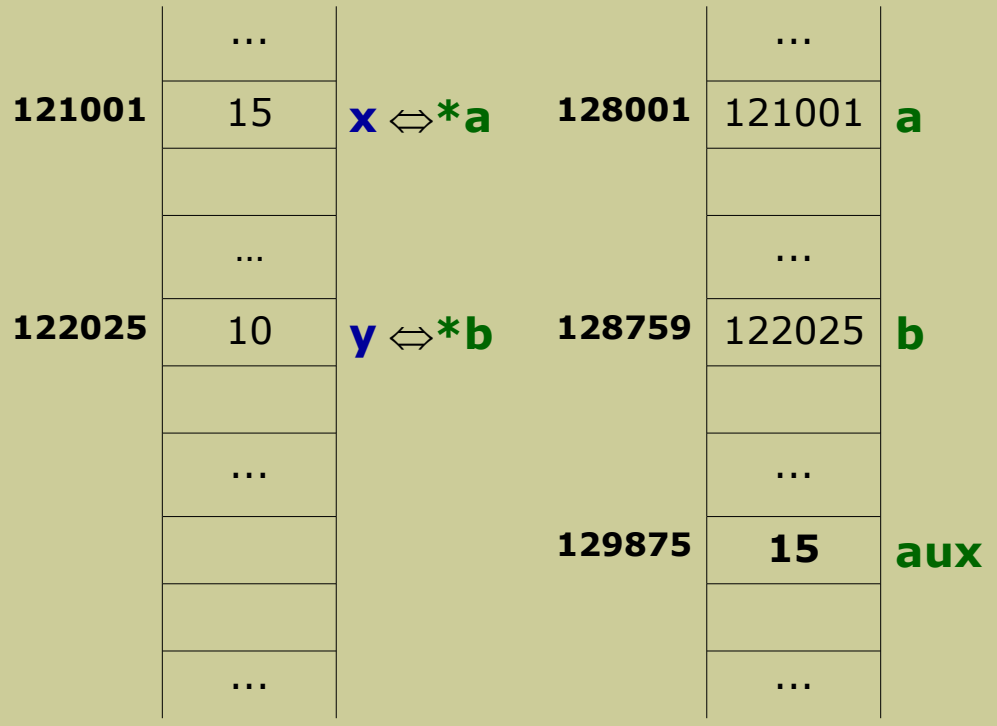


Solução 2:

O programa principal em vez de enviar os valores de **x** e **y**, por que não enviar os próprios **x** e **y**? Enviar os seus endereços de memória, através do operador **&**.

Exemplo

```
#include <stdio.h>
void trocar (int a, int b)
void trocar (int *a, int *b)
{
  int  aux;
  aux = a;  aux = *a;
  a = b;
  b = aux;
}
int main(){
  int  x, y;
  x = 15;
  y = 10;
  trocar(x, y);  trocar(&x, &y);
  printf("x = %d e y = %d\n", x, y);
}
```

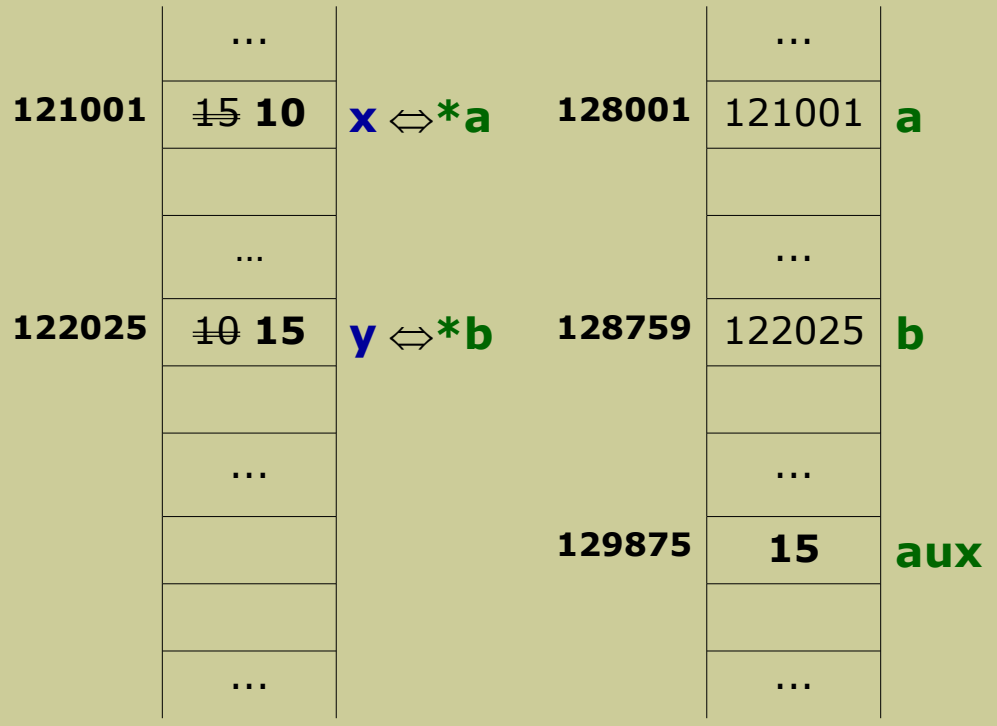


Solução 2:

O programa principal em vez de enviar os valores de **x** e **y**, por que não enviar os próprios **x** e **y**? Enviar os seus endereços de memória, através do operador **&**.

Exemplo

```
#include <stdio.h>
void trocar (int a, int b)
void trocar (int *a, int *b)
{
  int  aux;
  aux = a;   aux = *a;
  a = b;     *a = *b;
  b = aux;   *b = aux;
}
int main() {
  int  x, y;
  x = 15;
  y = 10;
  trocar(x, y);  trocar(&x, &y);
  printf("x = %d e y = %d\n", x, y);
}
```



Solução 2:

O programa principal em vez de enviar os valores de **x** e **y**, por que não enviar os próprios **x** e **y**? Enviar os seus endereços de memória, através do operador **&**.

Modelo de comunicação entre subprogramas

- Em princípio, todas as linguagens suportam o seguinte modelo de comunicação entre subprogramas
  - número de dados de entrada: M ($M \geq 0$)
  - número de dados de saída: N ($N \geq 0$)

Tipos de argumentos

- Nas várias linguagens de programação pode-se encontrar 3 tipos de argumentos
  - Entrada
  - Saída
  - Entrada e Saída
- Os dados de entrada para um subprograma
  - são designados por argumentos efetivos (ou concretos), e
  - pertencem ao domínio do programa ou subprograma invocador (o que chama)
- As variáveis associadas aos argumentos de um subprograma que recebem dados de entrada ou que devolvem resultados (dados de saída)
  - são designadas por argumentos formais, e
  - pertencem ao domínio do subprograma invocado (o que é chamado)

Tipos de argumentos

- A linguagem C só tem argumentos de entrada
  - há linguagens que admitem mais tipos de argumentos (por exemplo: Pascal)
- No entanto, na linguagem C,
  - é possível emular argumentos de saída e argumentos de entrada/saída, através de **variáveis apontadoras** que os referenciam

Métodos de passagem de argumentos

- Passagem por valor
  - é o único mecanismo de passagem de argumentos na linguagem C, o que significa que os argumentos são todos de entrada (de valores)
  - os argumentos efetivos pertencem ao domínio do subprograma invocador
  - os argumentos formais pertencem ao domínio do subprograma invocado
  - ideia base:
    - qualquer alteração no valor de um argumento formal não provoca alteração no valor do argumento efetivo correspondente

Métodos de passagem de argumentos

- Passagem por referência
  - ideia base:
    - passar a própria variável em vez do seu valor
      - isto implica alterar o valor de um argumento efetivo usando o argumento formal
  - não existe na linguagem C um mecanismo formal/sintático de passagem de argumentos por referência
  - o que se faz é simular este mecanismo através de passagem de argumentos por valor de um endereço de memória (através de ponteiros)

Passagem de um argumento por valor

- Funcionamento
  - o subprograma invocador **S** chama o subprograma **s** passando-lhe um VALOR
  - o VALOR pode ser uma **constante** ou o **conteúdo de uma variável** do subprograma **S** (argumento efetivo)
  - o subprograma invocado **s** recebe o VALOR e armazena-o numa variável do seu domínio (argumento formal)
- É um mecanismo unidireccional de comunicação de dados entre subprogramas
  - os argumentos formais do subprograma invocado recebem valores, mas não devolvem quaisquer valores
  - a devolução só pode ser feita usando a instrução **return**, e não através de um argumento formal
  - resumindo:
    - a passagem de argumentos por valor é um mecanismo unidireccional de **S** para **s**
    - a utilização da instrução **return** é um mecanismo unidireccional de **s** para **S**

Passagem de um argumento por referência

- Em vez de passar o valor de uma variável, passa-se o valor do seu endereço de memória (através do operador **&**), que é inalterável
- Ao passar-se um endereço de memória,
  - então há que o receber dentro do subprograma através de uma variável do tipo apontadora (um ponteiro)
- Se dentro do subprograma se usa uma variável apontadora, então pode-se alterar a variável por ela apontada
  - isto é, uma variável do domínio do subprograma invocador (normalmente é o programa principal – main)

Passagem de arrays para subprogramas

Chamada do subprograma

- Quando se passa um **array** como argumento, o subprograma
  - **não recebe** o array na totalidade,
  - **recebe** apenas o endereço do 1º elemento do array
    - passa-se o valor de **V** que é igual a **&V[0]**

Construção do subprograma

- Ao passar-se um endereço para o subprograma, a variável que o recebe terá de ser do tipo apontador para o tipo de dados dos elementos do array
- Logo, no cabeçalho do subprograma que recebe um array como argumento, aparece normalmente um apontador a receber o respetivo argumento

Exemplo

```
#include <stdio.h>
int soma (int *A, int N)
{
  int  k, S = 0;
  for (k = 0; k <= N-1; k++)
      S = S + A[k];
  return S;
}
int main()
{
  int   sm, V[5] = { 5, 10, 15, 20, 25 };
  float   media;
  sm = soma(V, 5);
  media = (float) sm / 5;
  printf("Media = %f\n", media);
}
```

NOTA:

- o argumento A do subprograma **soma** é um apontador para um inteiro, que receberá o endereço do **array** V, aquando da chamada do subprograma **soma** pelo programa principal (**main**).