Subprogramas

(Funções)

Subprograma

Definição

- Um subprograma é composto por um conjunto de comandos que, ao serem executados, realiza uma ou mais tarefas específicas
- Um subprograma constituí um módulo independente de quem o usa
- Um subprograma só pode ser usado/invocado/chamado
  - pelo programa principal (main), ou
  - por outro subprograma
- Um subprograma
  - recebe o comando do programa ou subprograma que o chama,
  - realiza as tarefas que lhe foram designadas, e por fim
  - devolve o comando ao programa ou subprograma que o chamou
- Um subprograma pode **devolver um elemento** de qualquer tipo de dados
- Na linguagem C, um subprograma designa-se habitualmente por **função**

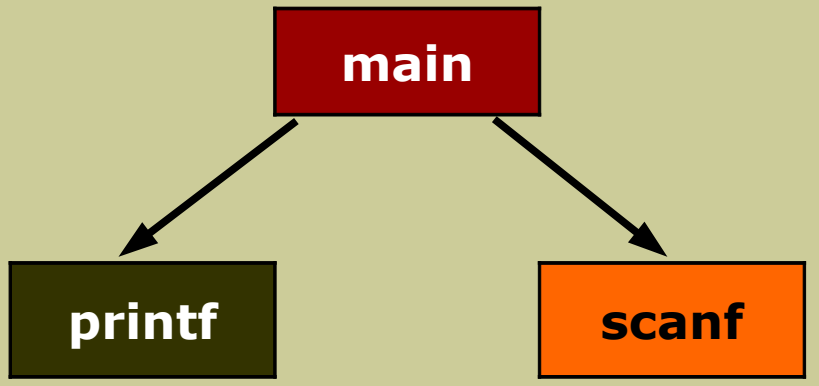
Relação hierárquica entre subprogramas

- Um programa ou subprograma pode utilizar subprogramas
  - já "implementados" (incluídos nas bibliotecas padrão), tais como:
    - subprogramas de entrada e de saída de dados (scanf, printf, ...)
    - funções matemáticas (sqrt, sin, cos, ...)
  - desenvolvidos e implementados especificamente para aquele programa ou subprograma

Relação hierárquica entre subprogramas

- A relação hierárquica entre subprogramas é estabelecida na escrita do programa ou subprograma que os chama

```
#include <stdio.h>
main(){
  int  x;
  float  y;
  printf("Escreva um valor inteiro: ");
  scanf("%d", &x);
  y = x + 5.4;
  printf("Valor de y = %f\n", y);
}
```



- os subprogramas **printf** e **scanf**
  - encontram-se definidos na biblioteca padrão "stdio.h"
  - são módulos independentes do programa que os chama

Vantagens e desvantagens de se usar subprogramas

- As **vantagens** são essencialmente as seguintes
  - evitar duplicação de código,
  - modular o programa (característico da programação estruturada),
  - reutilização de código já implementado (utilização de bibliotecas)
  - tornar o código-fonte mais claro
- A maior **desvantagem** é tornar o programa mais lento, pois a passagem do comando para os subprogramas e a sua devolução é uma tarefa que consume algum tempo

Sintaxe

- Um subprograma é composto por um **cabeçalho** e um **corpo**

```
tipo nome (argumentos)
```

cabeçalho

```
{
  declarações
  instruções
}
```

corpo

- O **corpo** é
  - um bloco de instruções ou instrução composta (bloco delimitado por **{ }**),
  - composto por um conjunto de **declarações** e um conjunto de **instruções**

Sintaxe - cabeçalho

```
tipo nome (argumentos)
```

- O cabeçalho especifica o que o subprograma faz
- O cabeçalho é composto por três entidades:
  - **nome** do subprograma (que deve ser sugestivo relativamente às tarefas que realiza)
  - **argumentos**, que correspondem aos dados de entrada do subprograma
    - cada argumento representa um dado de entrada
    - os argumentos são separados por vírgulas (**,**)
  - **tipo** de dados do resultado, que o subprograma pode devolver, que é de qualquer tipo
    - se não houver devolução de resultado, o tipo é **void**
- Um subprograma pode ser **declarado**
  - é definido apenas com o cabeçalho (sem corpo), mas termina com ponto e vírgula (;)
  - designa-se por **protótipo** e tem a seguinte sintaxe:

```
tipo nome (argumentos);
```

Sintaxe - corpo

```
{
  declarações
  instruções
}
```

- O **corpo** especifica **COMO** o subprograma faz, pois corresponde à implementação do algoritmo associado à realização das tarefas que lhe estão destinadas
- O **corpo** é composto pela zona de declarações e pela zona de instruções
- Na zona de **declarações** são declaradas todas as entidades necessárias (em especial as variáveis) à realização das tarefas contidas na zona das instruções
- As variáveis da zona de declarações e as que fazem parte dos argumentos, são variáveis **locais**, pois os espaços de memória que ocupam são libertados quando o subprograma termina (devolve o comando a quem o chamou)

Exemplo 1 (programa)

- Elaborar um programa para desenhar um quadrado com lado e caráter dados pelo utilizador.

```
#include <stdio.h>
main(){
  char  ch;
  int  k, j, lado;
  printf("Qual o caráter?\n");
  scanf("%c", &ch);
  printf("Qual o lado?\n");
  scanf("%d", &lado);
  for (k = 1; k <= lado; k = k + 1){
     for (j = 1; j <= lado; j = j + 1)
        printf("%c", ch);
     printf("\n");
  }
}
```

Exemplo 1 (subprograma + programa)

- Subprograma para desenhar um quadrado com o caráter c, com lado N
  - nome: **desenharQuadrado**
  - argumentos (dados de entrada): um caráter (**char c**) e um valor inteiro (**int N**)
  - tipo de dados do resultado: **void** (não há devolução de resultado)

void desenharQuadrado (char c, int N)

```
{
  int  k, j;
  for (k = 1; k <= N; k = k + 1){
      for (j = 1; j <= N; j = j + 1)
          printf("%c", c);
      printf("\n");
  }
}
```

Exemplo 1 (subprograma + programa)

- Elaborar um programa para desenhar um quadrado com lado e caráter dados pelo utilizador, usando o subprograma **desenharQuadrado**

```
#include <stdio.h>
void desenharQuadrado (char c, int N);
main(){
   char  ch;
   int  lado;
   printf("Qual o caráter?\n");
   scanf("%c", &ch);
   printf("Qual o lado?\n");
   scanf("%d", &lado);
   desenharQuadrado(ch, lado);
}
```

Exemplo 2 (subprograma + programa)

- Subprograma para calcular o fatorial de um número inteiro positivo
  - nome: **fatorial**
  - argumentos (dados de entrada): um valor inteiro positivo (**int N**)
  - tipo de dados do resultado que é devolvido: **int**

```
int fatorial (int N)
```

```
{
  int  k, resultado;
  resultado = 1;
  for (k = 1; k <= N; k = k + 1)
      resultado = resultado * k;
  return resultado;
}
```

- Observação: por definição, fatorial(0) = 1

Exemplo 2 (subprograma + programa)

- Elaborar um programa para calcular o fatorial de um número inteiro positivo, usando o subprograma **fatorial**

```
#include <stdio.h>
int fatorial (int N);
main(){
  int  num, fat;
  do{
     printf("Inserir um numero inteiro positivo:\n");
     scanf("%d", &num);
  }while (num < 0);
  fat = fatorial(num);
  printf("Fatorial(%d) = %d\n", num, fat);
}
```

A instrução return

- Um subprograma devolve o comando para quem o chamou quando é executada:
  - a instrução **return**, ou
  - a última instrução do subprograma
- Quando a devolução do comando é feita com a execução da instrução **return**, o subprograma
  - termina o seu processamento, e
  - devolve o comando para o programa ou subprograma que o chamou
- A instrução **return** pode ter associado um resultado
  - se **não tem**, então nada mais faz
  - se **tem**, então também envia este resultado para quem o chamou
- Sintaxe

```
return  valor ;
```

em que

- **valor** é o resultado a enviar para quem chamou o subprograma

Exemplo 1

- Subprograma para desenhar um quadrado com o caráter c, com lado N

  (não há devolução de resultado)

```
void desenharQuadrado (char c, int N)
{
  int  k, j;
  for (k = 1; k <= N; k = k + 1){
     for (j = 1; j <= N; j = j + 1)
        printf("%c", c);
     printf("\n");
  }
}
```

```
void desenharQuadrado (char c, int N)
{
  int  k, j;
  for (k = 1; k <= N; k = k + 1){
     for (j = 1; j <= N; j = j + 1)
        printf("%c", c);
     printf("\n");
  }
  return ;
}
```

Exemplo 2

- Subprograma para calcular (e **devolver**) o fatorial de um número inteiro positivo

```
int fatorial (int N)
{
  int  k, resultado;
  resultado = 1;
  for (k = 1; k <= N; k = k + 1)
      resultado = resultado * k;
  return resultado;
}
```

Argumentos/Parâmetros

- A passagem de valores para um subprograma é feito através dos argumentos (ou parâmetros)
- Os argumentos de um subprograma são definidos no seu cabeçalho

tipo nome (argumentos)

- Cada argumento é definido como uma declaração de uma variável
- Uma lista de argumentos é definida com uma lista de declarações de variável

Argumentos/Parâmetros

- Sintaxe (um argumento)

```
tipo  nome
```

- Sintaxe (vários argumentos)

```
tipo  nome1, tipo nome2, ...
```

em que se define um tipo para cada argumento

  - mesmo que vários argumentos sejam do mesmo tipo

- Exemplo:
  - errado: int soma (int a, b)
  - correto: int soma (int a, int b)

Utilização/Invocação/Chamada de subprogramas

- Na chamada de um subprograma os argumentos são posicionais
  - deve-se utilizar o número exato de argumentos e a posição correta
- Por exemplo

```
void desenharQuadrado (char c, int N)
...
```

  - um programa ou subprograma ao invocar/chamar este subprograma
    - deve ser com 2 argumentos (e não 1 nem 3) e
    - o primeiro deve ser um caráter e o segundo um inteiro

Utilização/Invocação/Chamada de subprogramas

- Exemplos de chamadas incorretas

```
void desenharQuadrado (char c, int N);          (carater, inteiro)
main(){
  int  a, b;
  char c, d;
  ...
  desenharQuadrado(a, b);                       (inteiro, inteiro)
  desenharQuadrado(a, c);                       (inteiro, carater)
  desenharQuadrado(a);                          (inteiro)
}
```

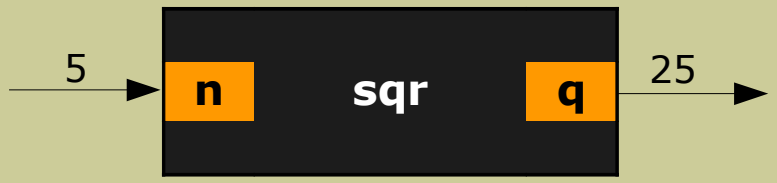
Utilização/Invocação/Chamada de subprogramas

- Exemplos de chamadas corretas

```
void desenharQuadrado (char c, int N);          (carater, inteiro)
main(){
  int  a, b;
  char c, d;
  ...
  desenharQuadrado(c, b);                       (carater, inteiro)
  desenharQuadrado('*', 4);                     (carater, inteiro)
  desenharQuadrado(d, a+b);                     (carater, inteiro)
}
```

Características de um subprograma

- Identificação
  - tem nome/identificador único
- Utilização/Invocação/Chamada
  - é feita a partir do programa principal ou de outro subprograma
- Unicidade
  - deve realizar uma única tarefa bem definida.
- Funcionamento
  - funciona como uma caixa preta



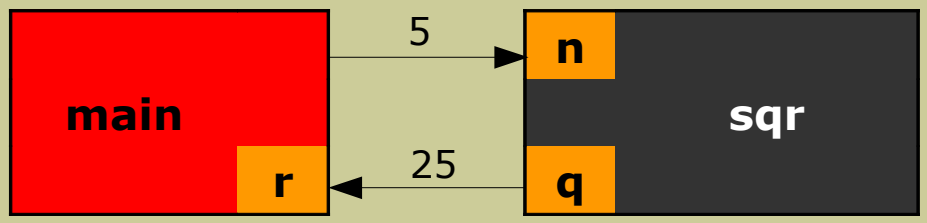
- Generalidade/Reutilização
  - a codificação de um subprograma deve ser **genérica** e **independente** de qualquer programa ou projeto, para que possa ser reutilizada por outros programas ou projetos

Comunicação entre subprogramas

- Utilização/Invocação/Chamada
  - é feita a partir do programa principal ou de outro subprograma
  - o programa principal ou subprograma invocador é suspenso
  - o subprograma invocado é executado
  - o subprograma invocador retoma a sua execução
- Execução
  - após ser invocado, um subprograma é executado:
    - entrada de dados ou passagem de argumentos/parâmetros
    - executa bloco de instruções
    - saída de dados

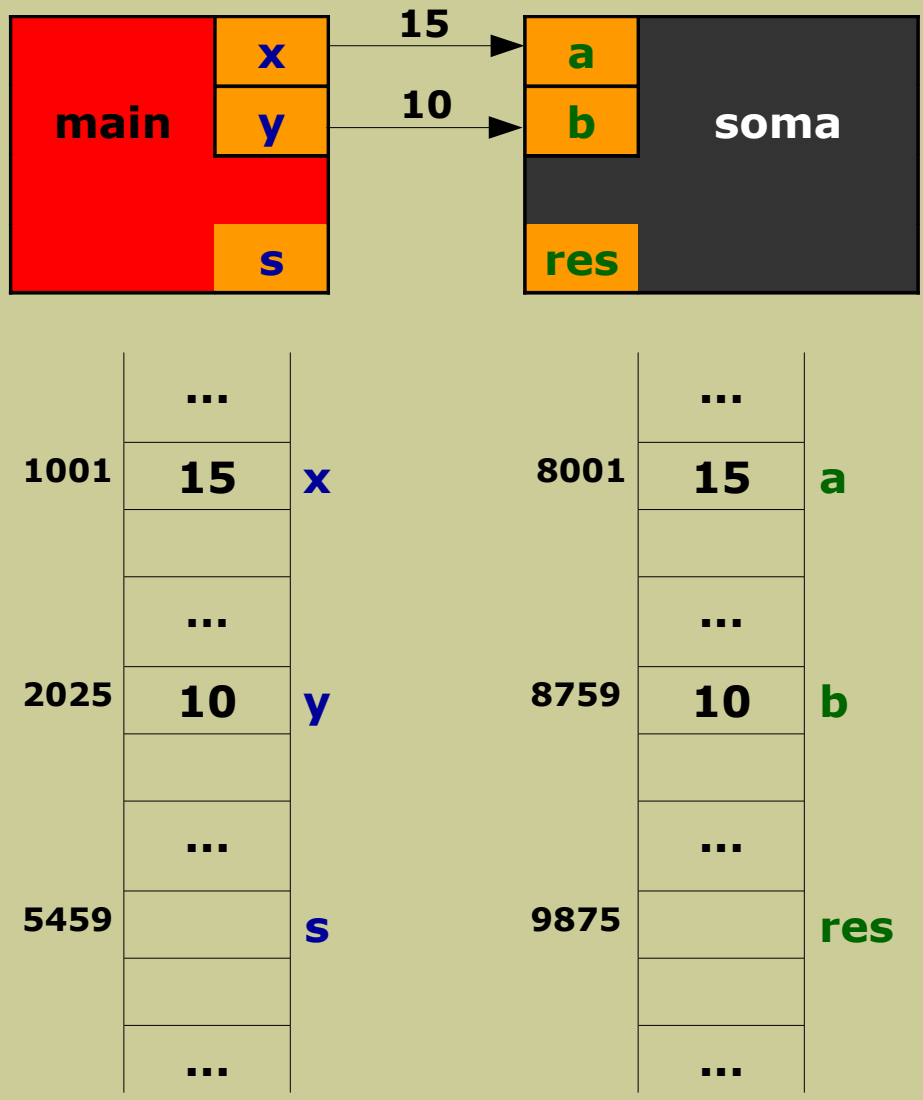
Comunicação entre subprogramas

- Passagem de argumentos/parâmetros
  - é uma operação de entrada de dados
  - associação de 1 para 1:
    - cada argumento efetivo (ou concreto) a
    - um argumento formal (que é uma variável)
- Retorno/Devolução de valor
  - é uma operação de saída de dados (resultados)



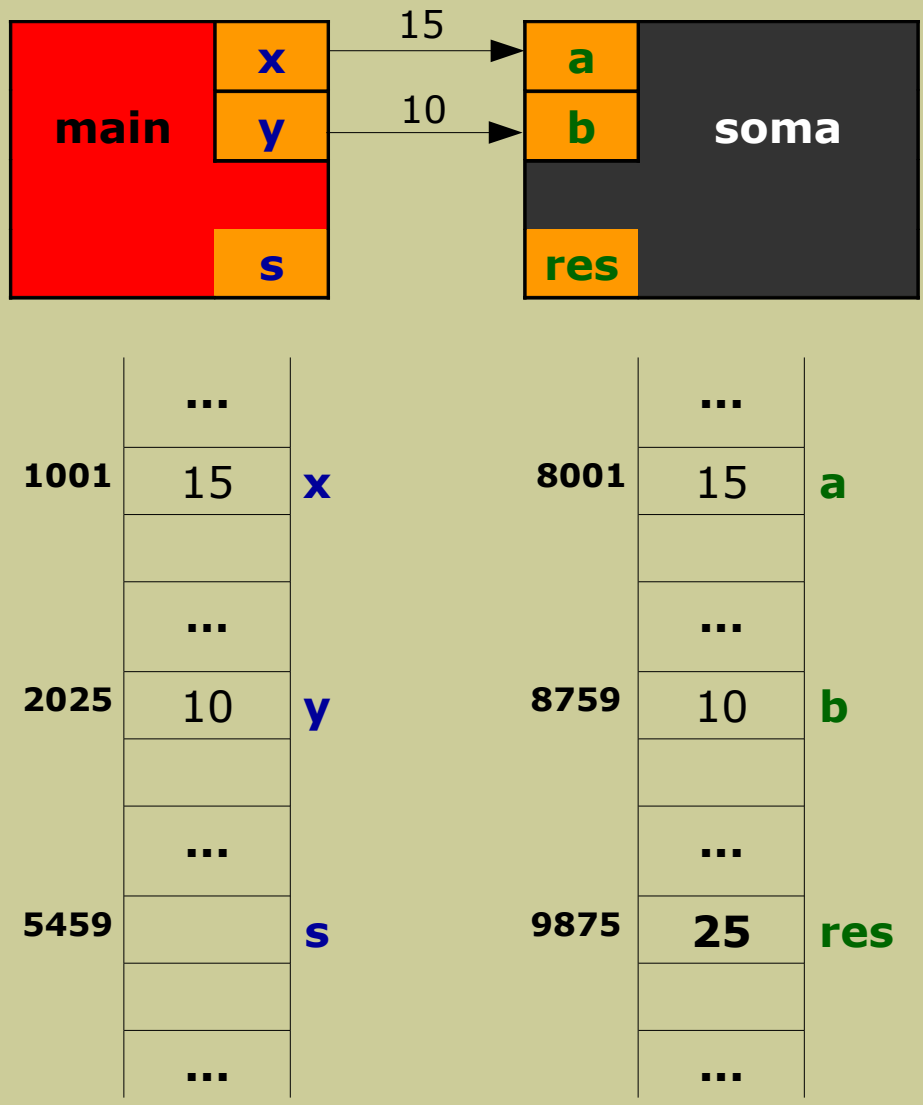
Variáveis locais – exemplo 1

```
#include <stdio.h>
int soma (int a, int b){
   int  res;
   res = a + b;
   return res;
}
int main(){
   int  x, y, s;
   x = 15;
   y = 10;
   s = soma(x, y);
   printf("Soma = %d\n", s);
}
```



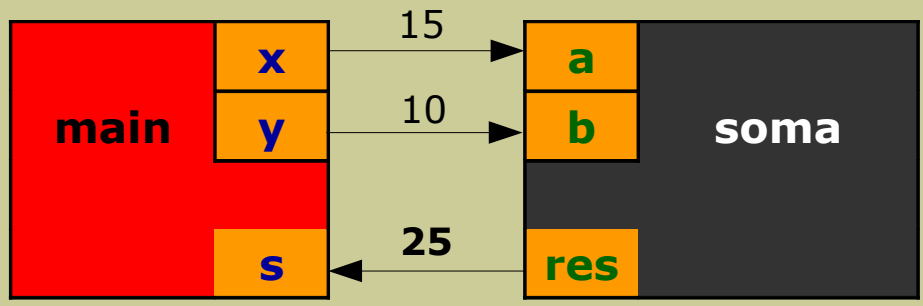
Variáveis locais – exemplo 1

```
#include <stdio.h>
int soma (int a, int b){
   int  res;
   res = a + b;
   return res;
}
int main(){
   int  x, y, s;
   x = 15;
   y = 10;
   s = soma(x, y);
   printf("Soma = %d\n", s);
}
```



Variáveis locais – exemplo 1

```
#include <stdio.h>
int soma (int a, int b){
   int  res;
   res = a + b;
   return res;
}
int main(){
   int  x, y, s;
   x = 15;
   y = 10;
   s = soma(x, y);
   printf("Soma = %d\n", s);
}
```

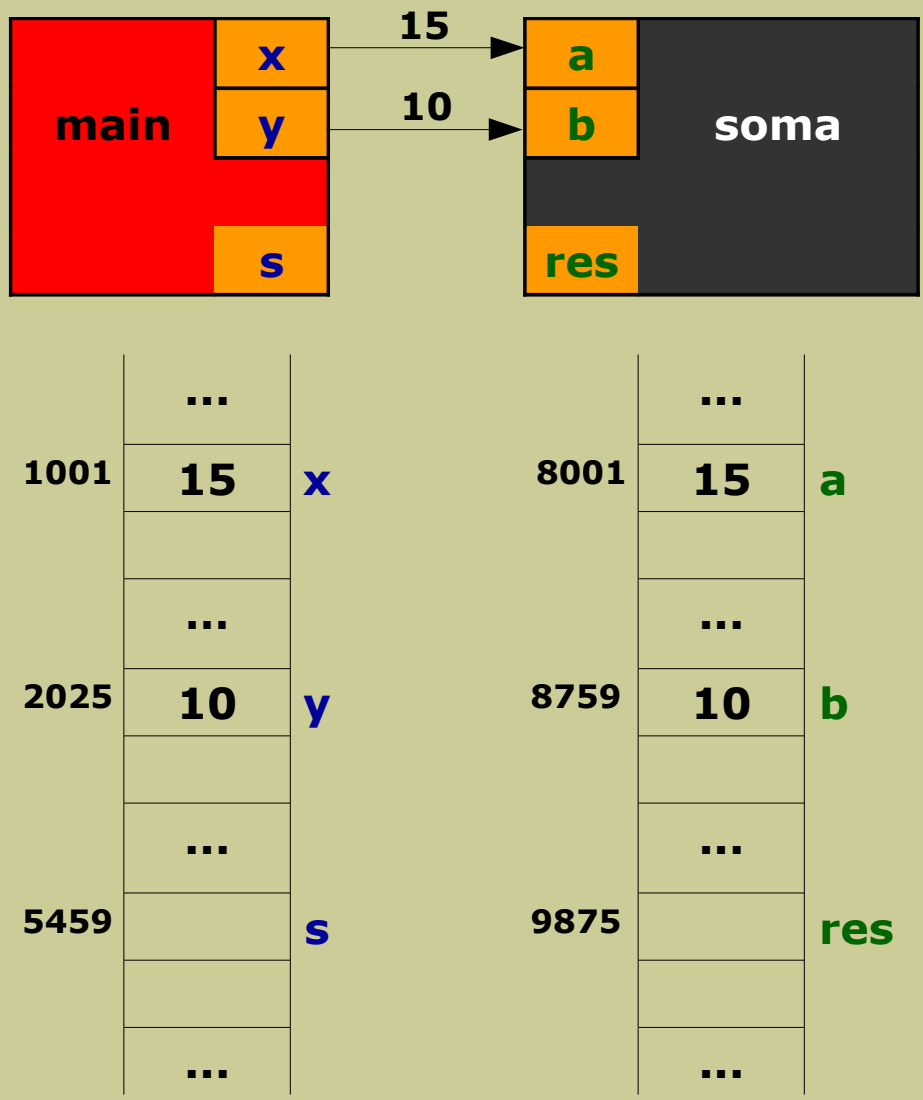


	...	
1001	15	x
	...	
2025	10	y
	...	
5459	25	s
	...	

	...	
8001	15	a
	...	
8759	10	b
	...	
9875	25	res
	...	

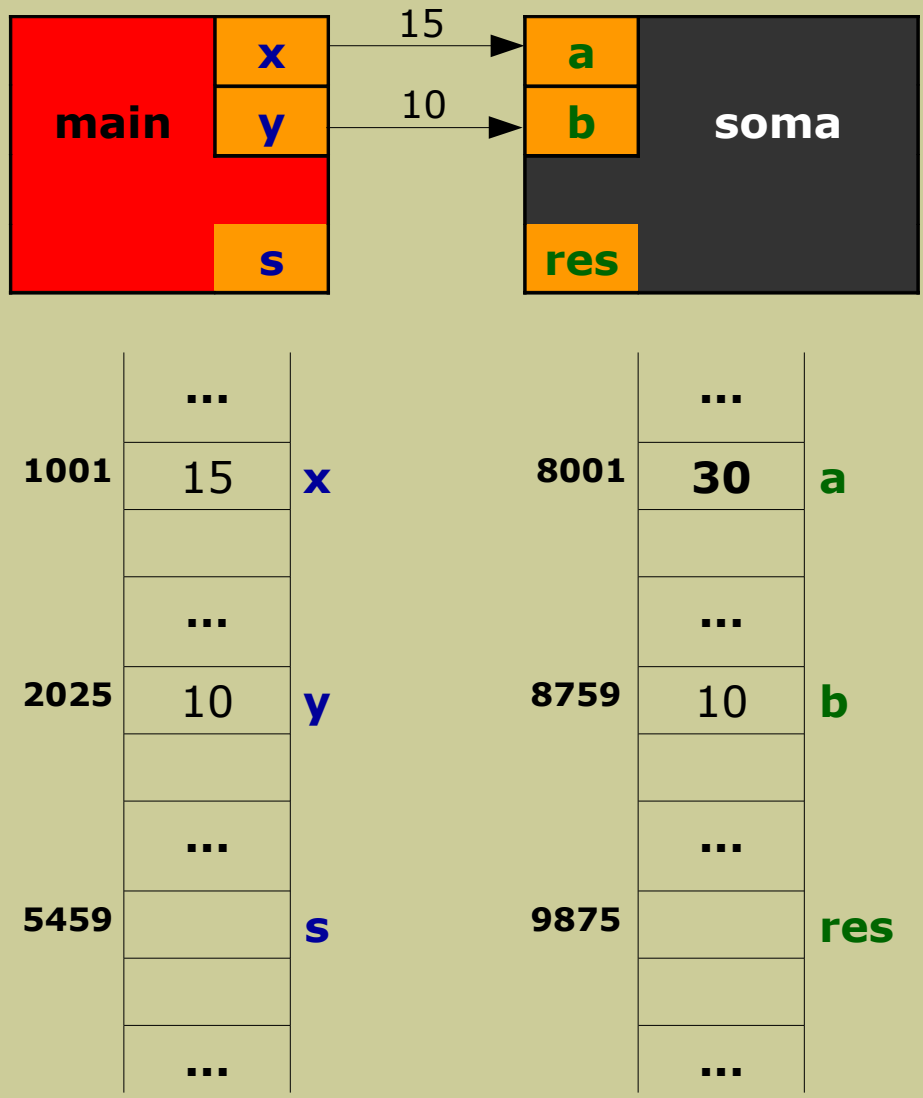
Variáveis locais – exemplo 2

```
#include <stdio.h>
int soma (int a, int b){
   int  res;
   a = 2 * a;
   res = a + b;
   return res;
}
int main(){
   int  x, y, s;
   x = 15;
   y = 10;
   s = soma(x, y);
   printf("Soma = %d\n", s);
}
```



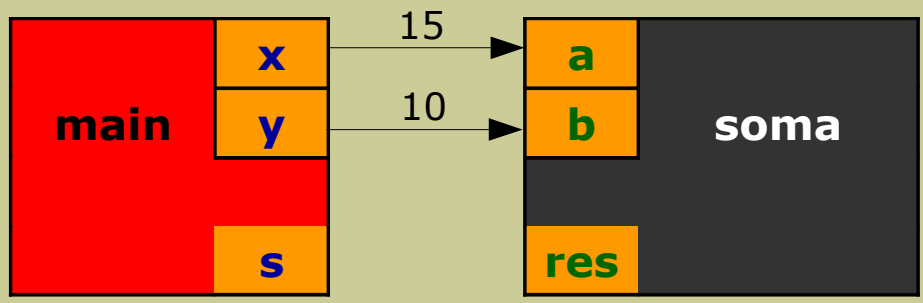
Variáveis locais – exemplo 2

```
#include <stdio.h>
int soma (int a, int b){
   int  res;
   a = 2 * a;
   res = a + b;
   return res;
}
int main(){
   int  x, y, s;
   x = 15;
   y = 10;
   s = soma(x, y);
   printf("Soma = %d\n", s);
}
```



Variáveis locais – exemplo 2

```
#include <stdio.h>
int soma (int a, int b){
   int  res;
   a = 2 * a;
   res = a + b;
   return res;
}
int main(){
   int  x, y, s;
   x = 15;
   y = 10;
   s = soma(x, y);
   printf("Soma = %d\n", s);
}
```

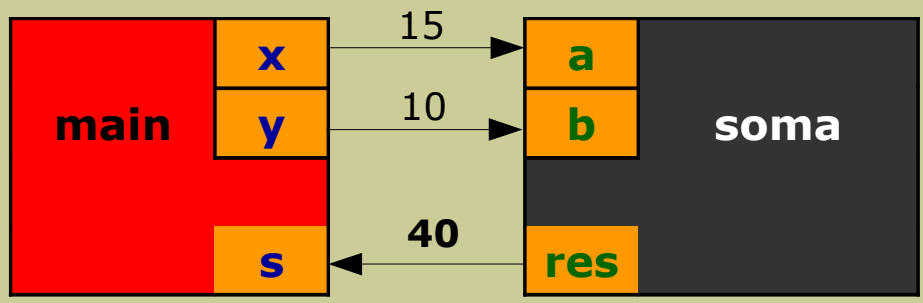


	...	
1001	15	x
	...	
2025	10	y
	...	
5459		s
	...	

	...	
8001	30	a
	...	
8759	10	b
	...	
9875	**40**	res
	...	

Variáveis locais – exemplo 2

```
#include <stdio.h>
int soma (int a, int b){
   int  res;
   a = 2 * a;
   res = a + b;
   return res;
}
int main(){
   int  x, y, s;
   x = 15;
   y = 10;
   s = soma(x, y);
   printf("Soma = %d\n", s);
}
```



	...	
1001	15	x
	...	
2025	10	y
	...	
5459	40	s
	...	

	...	
8001	30	a
	...	
8759	10	b
	...	
9875	40	res
	...	

Variáveis locais

- Uma variável local só existe na zona de memória associada ao subprograma onde está declarada
- Após a execução do subprograma, a memória ocupada pelas suas variáveis (locais) é libertada
- As variáveis declaradas no programa principal são também locais (ao programa principal), pelo que têm as mesmas características das declaradas nos subprogramas

Outros exemplos

Exemplo 3

- Enunciado
  - implementar o subprograma (função) **maior** que devolve o maior de dois números inteiros passados como parâmetros
  - construir um programa em C que determine o maior de três números inteiros, usando a função anterior.

Exemplo 3

- Subprograma (função)
  - maiorDe2
- Argumentos
  - dois valores inteiros, N1 e N2
- Resultado (retorno)
  - um valor inteiro correspondente ao maior valor entre N1 e N2, maior

Exemplo 3

- Algoritmo (subprograma) em pseudo-código

```
algoritmo maiorDe2
  parâmetros de entrada: N1 e N2 (inteiros)
  parâmetros de saída: maior (inteiro)
  se (N1 > N2) então
     maior ← N1
  senão
     maior ← N2
  fim_se
  devolver maior
fim_algoritmo
```

Exemplo 3

- Subprograma em C

```
int  maiorDe2  (int N1, int N2){
   int  maior;
   if (N1 > N2)
       maior = N1;
   else
       maior = N2;
   return  maior;
}
```

Exemplo 3

- Algoritmo (programa principal) em pseudo-código

```
algoritmo programaPrincipal
   escrever: "Inserir três números inteiros: "
   ler: A, B, C
   maior ← maiorDe2(A, B)
   maior ← maiorDe2(maior, C)
   escrever: "Maior = ", maior
fim_algoritmo
```

Exemplo 3

- Programa principal em C

```
#include <stdio.h>
int  maiorDe2 (int N1, int N2);
int main(){
   int   A, B, C, maior;
   printf("Insira três números inteiros: ");
   scanf("%d%d%d", &A, &B, &C);
   maior = maiorDe2(A, B);
   maior = maiorDe2(maior, C);
   printf("O maior entre %d, %d e %d é : %d\n", A, B, C, maior);
   return 1;
}
```

Exemplo 4

- Enunciado
  - implementar o subprograma (função) **maiorVarios** que determine (e devolva) o maior número entre N números reais (N ≥ 2) introduzidos pelo utilizador (um de cada vez)
  - construir um programa em C para determinar o maior número inserido pelo utilizador, entre N (N ≥ 2) números reais - usar a função **maiorVarios**

Exemplo 4

- Subprograma (função)
  - maiorVarios
- Argumentos
  - um número inteiro N (N ≥ 2)
- Resultado (retorno)
  - o maior valor dos N valores reais inseridos pelo utilizador, maior

Exemplo 4

- Algoritmo (subprograma) em pseudo-código

```
algoritmo maiorVarios
  parâmetros de entrada: N (inteiro)
  parâmetros de saída: maior (real)
  ler: X
  maior ← X
  k ← 2
  enquanto (k ≤ N) fazer
     ler: X
     se (X > maior) então
        maior ← X
     fim_se
     k ← k + 1
  fim_enquanto
  devolver maior
fim_algoritmo
```

Exemplo 4

- Subprograma em C

```
float maiorN (int N){
   int  k;
   float  maior, X;
   printf("Inserir um numero real: ");
   scanf("%f", &X);
   maior = X;
   for (k = 2; k <= N; k = k + 1){
       printf("Inserir outro número real:");
       scanf("%f", &X);
       if (X > maior)
           maior = X;
   }
   return  maior;
}
```

Exemplo 4

- Algoritmo (programa principal) em pseudo-código

```
algoritmo programaPrincipal
  fazer
     escrever: "Quantos numeros quer inserir (≥ 2)?"
     ler: N
  enquanto (N < 2)
  maior ← maiorVarios(N)
  escrever: "O maior número inserido foi: ", maior
fim_algoritmo
```

Exemplo 4

- Programa principal em C

```
#include <stdio.h>
float maiorVarios (int N);
int main(){
  int  N;
  float  maior;
  do{
     printf("Quantos numeros quer inserir (≥ 2)?");
     scanf("%d", &N);
  }while (N < 2);
  maior = maiorVarios(N);
  printf("O maior número inserido foi : %f\n", maior);
}
```

Exemplo 5

- Enunciado
  - implementar o **subprograma** (função) que determine (e devolva) o preço final de um produto, dado o seu código e o seu preço base;
    - o preço final depende do seu preço base e do IVA a incidir sobre ele
    - o IVA de um produto está relacionado com o algarismo das unidades do seu código, da seguinte forma:
      - códigos terminados em 0, 1, 2 e 3, têm IVA de 6%
      - códigos terminados em 4 e 5, têm IVA de 13%
      - códigos terminados em 6, 7, 8 e 9, têm IVA de 23%
  - construir um **programa** para determinar o total a pagar por vários produtos adquiridos num supermercado

Exemplo 5

- Subprograma (função)
  - precoFinal
- Argumentos
  - código de produto (um número inteiro positivo)
  - o preço base do produto (um número real)
- Resultado (retorno)
  - o preço final do produto

Exemplo 5

- Algoritmo (subprograma) em pseudo-código

```
algoritmo precoFinal
  parâmetros de entrada: cod (inteiro) e pBase (real)
  parâmetros de saída: pFinal (real)
  se (restoDivisão(cod, 10) <= 3) então
     IVA ← 6
  senão
     se (restoDivisão(cod, 10) <= 5) então
        IVA ← 13
     senão
        IVA ← 23
     fim_se
  fim_se
  pFinal ← pBase + (pBase * IVA/100)
  devolver pFinal
fim_algoritmo
```

Exemplo 5

- Subprograma em C

```
float precoFinal (int cod, float pBase){
  int  IVA;
  float  pFinal;
  if (cod % 10 <= 3)        // cod termina em 0, 1, 2 ou 3
     IVA = 6;
  else
     if (cod % 10 <= 5)     // cod termina em 4 ou 5
        IVA = 13;
     else                   // cod termina em 6, 7, 8 ou 9
        IVA = 23;
  pFinal = pBase + (pBase * (float)IVA / 100);
  return  pFinal;
}
```

Exemplo 5

- Algoritmo (programa principal) em pseudo-código

```
algoritmo programaPrincipal
  totalPagar ← 0
  escrever: "Inserir o código do produto (inteiro negativo para terminar): "
  ler: cod
  enquanto (cod > 0) fazer
     escrever: "Inserir o preço base do produto: "
     ler: pBase
     pFinal ← precoFinal(cod, pBase)
     totalPagar ← totalPagar + pFinal
     escrever: "Inserir o código do produto (inteiro negativo para terminar): "
     ler: cod
  fim_enquanto
  escrever: "Preço total a pagar: ", totalPagar
fim_algoritmo
```

Exemplo 5

- Programa principal em C

```
#include <stdio.h>
float precoFinal (int cod, float pBase);
int main(){
   int  cod;
   float  pBase, pFinal, totalPagar;
   totalPagar = 0;
```

Exemplo 5

- Programa principal em C

```
. . .
   printf("Inserir o código do produto [inteiro negativo para terminar]: ");
   scanf("%d", &cod);
   while(cod > 0){
      printf("Inserir o preço base do produto: ");
      scanf("%f", &pBase);
      pFinal = precoFinal(cod, pBase);
      totalPagar = totalPagar + pFinal;
      printf("Inserir o código do produto [inteiro negativo para terminar]: ");
      scanf("%d", &cod);
   }
   printf("Preço total a pagar: %f\n", totalPagar);
}
```