Algoritmos e Programação

Linguagem C

Procedimentos

e

Funções

Eliane Pozzebon

Procedimentos e Funções

- Procedimentos – são estruturas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado.
- Funções – são procedimentos que retornam um valor ao seu término.
- A Linguagem C não faz distinção.

Porque utilizar procedimentos e funções?

- Evitam que os blocos do programa fiquem grandes demais e mais difíceis de ler e entender.
- Ajudam a organizar o programa.
- Permitem reaproveitamento de códigos construídos anteriormente.
- Evitam repetição de trechos de códigos, minimizando erros e facilitando alterações.

Funções
:: Como declarar

```
<tipo> nome_da_função (<tipo> arg1, <tipo> arg2, ...,
                       <tipo> argN)
{
   <corpo da função>

   return valor_de_retorno;

}
```

Funções
:: Como declarar

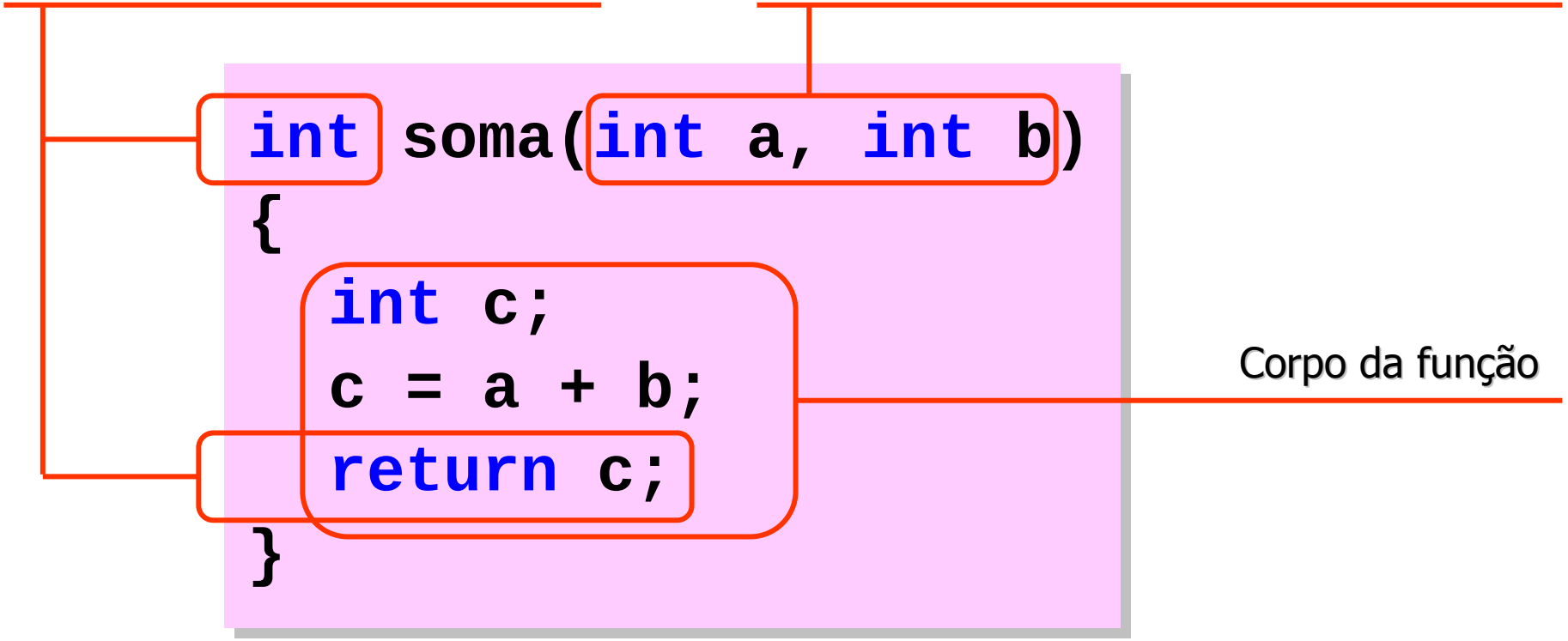
- Exemplo de uma função:

```
int soma(int a, int b)
{
   int c;
   c = a + b;
   return c;
}
```

Funções
:: Como declarar

Toda função deve ter um tipo (char, int, float), o qual indicará o tipo de seu valor de retorno (saída).

Os argumentos (ou parâmetros) indicam o tipo e quais valores são esperados para serem manipulados pela função (entrada).



Funções
:: Como declarar

- Uma função pode não ter argumentos, basta não informá-los. Exemplo:

```
int random()
{
   srand(time(NULL));
   return (rand() % 100);
}
```

Funções
:: Como declarar

- A expressão contida no comando **`return`** é chamado de valor de retorno da função.
- Esse comando é sempre o último a ser executado por uma função. Nada após ele será executado.
- As funções só podem ser declaradas fora de outras funções. Lembre-se que o corpo do programa principal (**`main()`**) é uma função!

Funções
:: Invocando

- Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor a uma variável:

```
resultado = soma(x,y);
```

- Na verdade, o resultado da chamada de uma função é uma expressão, que pode ser usada em qualquer lugar que aceite uma expressão:

```
printf("Soma: %d\n", soma(a,b) );
```

Funções:: Invocando

- Função que calcula a soma dos valores de x e y:

```
int x, y, resultado;
int soma(int a, int b){
    return (a + b);
}

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

Funções
:: Invocando

- As variáveis x e y no exemplo anterior são chamadas de parâmetros reais.
- Conforme exemplo anterior, os argumentos não possuem necessariamente os mesmos nomes que os parâmetros que a função espera.
- Seus valores são apenas copiados para a função chamada, sem ser afetados pelas alterações nos parâmetros dentro da função.

O tipo `void`

- É utilizado em procedimentos.
- É um tipo que representa o “nada”, ou seja:
  - uma variável desse tipo armazena conteúdo indeterminado,
  - uma função desse tipo retorna um conteúdo indeterminado.
- Indica que uma função não retorna nenhum valor, ou seja, é um procedimento.

Procedimentos
:: Como declarar

```
void nome_do_procedimento (<tipo> parâmetro1,
            <tipo> parâmetro2, ..., <tipo> parâmetroN)
{
   <corpo do procedimento>
}
```

Procedimentos
:: Como declarar

- Exemplo de procedimento:

```
void imprime_dobro(int x)
{
   printf("Dobro de x: %d", 2*x);
}
```

Procedimentos :: Invocando

- Para invocarmos um procedimento, devemos utilizá-lo como qualquer outro comando:

```
procedimento(parâmetros);
```

- Compare a diferença de invocação de uma função:

```
resultado = função(parâmetros);
```

Procedimentos :: Invocando

```
int x, y, resultado;

void soma()
{
   resultado = x + y;
}

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

A função `main()`

- É uma função especial invocada automaticamente pelo sistema operacional (OS) ao iniciar o programa.
- Quando utilizado, o comando **`return`** informa ao OS se o programa funcionou corretamente ou não.
- O padrão é que um programa retorne:
  - = zero – caso tenha funcionado corretamente ,
  - ≠ zero – caso contrário.

Declaração de funções e procedimentos Sem defini-los

- Funções e procedimentos podem ser definidos antes ou depois da função **`main()`**.
- Em ambos os casos, as funções e/ou procedimentos devem ser declarados antes da função **`main()`**.
- O conjunto de comandos que constituem uma função ou procedimento pode ser definido em qualquer lugar do programa.

Declaração de funções e procedimentos sem defini-los

- Para declarar uma função sem defini-la (especificar seu código), substituímos as chaves e seu conteúdo por ponto-e-vírgula.

```
<tipo> nome (<tipo> parâmetro1, <tipo>
  parâmetro2, ..., <tipo> parâmetroN);
```

Variáveis locais e globais

- Uma variável é chamada local quando é declarada dentro de uma função. Nesse caso:
  - Ela existe apenas dentro da função que a contém.
  - Após o término da execução da função, ela deixa de existir.

- Uma variável é chamada global quando é declarada fora de qualquer função. Nesse caso:
  - Pode ser acessada em qualquer parte do programa.
  - Ela existe durante toda a execução do programa.

Variáveis locais e globais

- Boa prática de programação:
  - Deve-se evitar o uso de variáveis globais.
  - As funções devem modificar apenas as suas variáveis locais e as variáveis passadas a elas como parâmetros.

Escopo de variáveis

- O escopo de uma variável determina de que partes do código ela pode ser acessada.

- A regra de escopo em C é bem simples:
  - As variáveis globais são visíveis por todas as funções.
  - As variáveis locais são visíveis apenas na função onde foram declaradas.

Escopo de variáveis

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local “esconde” a variável global.

```
int nota;
void funcao()
{
   int nota;
   // Neste ponto, nota eh variavel local.
}
```

Variáveis automáticas × estáticas

- Variáveis declaradas dentro de uma função (locais) têm existência apenas enquanto a função está sendo executada, deixando de existir quando a função termina sua tarefa.

- Tal mecanismo é chamado pilha de execução:
  1. A pilha de execução está vazia;
  2. Ao iniciar a execução de um bloco {}, as variáveis são empilhadas à medida em que são criadas;
  3. Ao final de um bloco {}, essas variáveis são desempilhadas, liberando espaço na memória.

Variáveis automáticas × estáticas

- Exemplo:
  - Considere um procedimento que simplifica uma fração
  - A fração é guardada em um vetor de dois elementos:
    - posição 0: guarda o numerador
    - posição 1: guarda o denominador
  - A fração é simplificada dividindo-se cada elemento pelo máximo divisor comum

Variáveis automáticas × estáticas

```
void simplifica(int f[]) {
   int num, den;
   num = f[0];
   den = f[1];
   while (den)     // Equivale a (den != 0)
   {
      int resto;
      resto = num % den;
      num   = den;
      den   = resto;
   }
   f[0] = f[0]/num;
   f[1] = f[1]/num;
}
```

Variáveis automáticas × estáticas :: Situação da pilha de execução

Antes do laço while:

Dentro do laço while:

Após o laço while:

Depois do procedimento:

Variáveis automáticas × estáticas

- As variáveis locais também são conhecidas como variáveis automáticas, por serem criadas e destruídas sempre que a função for executada.
- Se houver uma atribuição de valor a uma variável dentro de uma função, ela será realizada todas as vezes em que essa função for chamada.

Variáveis automáticas × estáticas

- Para que uma função não destrua uma variável local ao final da execução, é preciso declará-la como estática.
- Para uma variável estática, a atribuição de valor inicial acontece somente uma vez, quando a variável é criada.
- Ao contrário das variáveis globais, as variáveis estáticas só podem ser usadas dentro da função que as declara.

Parâmetros

- Parâmetros ou argumentos são os valores recebidos e/ou retornados por uma função.

- Podem ser divididos em duas categorias:
  - Formais: correspondem aos parâmetros utilizados na definição da função.
  - Reais: correspondem aos parâmetros da função chamadora utilizados para chamar a função.

Parâmetros

Parâmetros formais

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

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

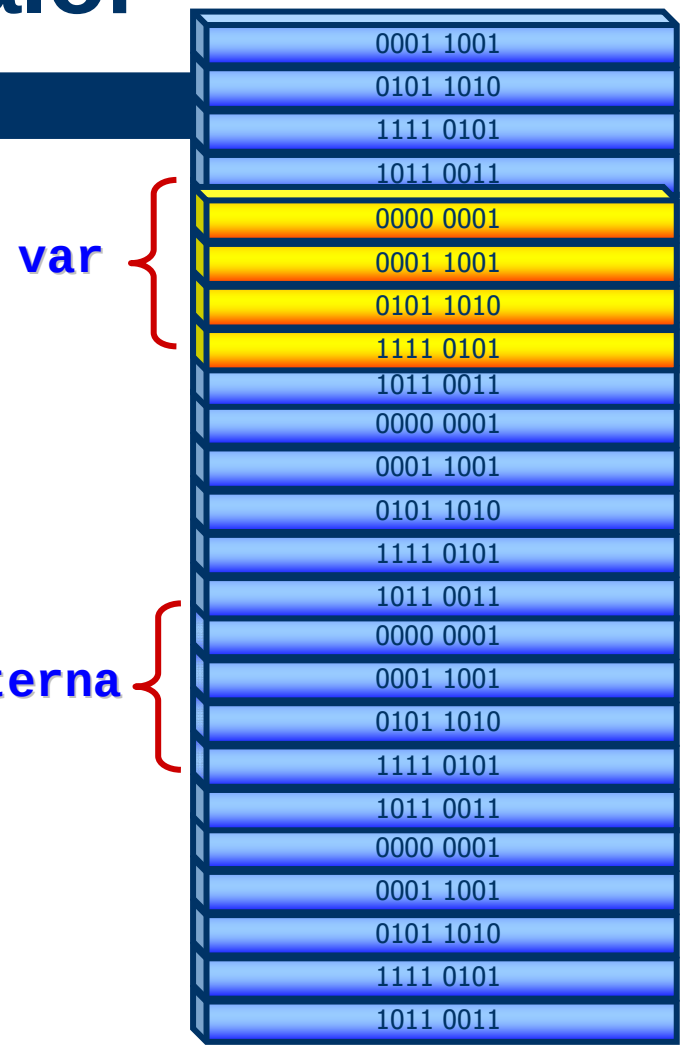
Parâmetros reais

Passagem de Parâmetros

- É o mecanismo de informar sobre quais valores o processamento definido na função deve ser realizado.
- Os parâmetros são passados para uma função de acordo com a sua posição.
- Os parâmetros formais de uma função se comportam como variáveis locais (criados na entrada e destruídos na saída)
- Existem duas categorias:
  - Por valor
  - Por referência

Passagem de Parâmetros :: Passagem por valor

- Os valores das variáveis externas (função chamadora) são copiados para as variáveis internas da função chamada.
- Alteração no valor das variáveis terá efeito local à função chamada.



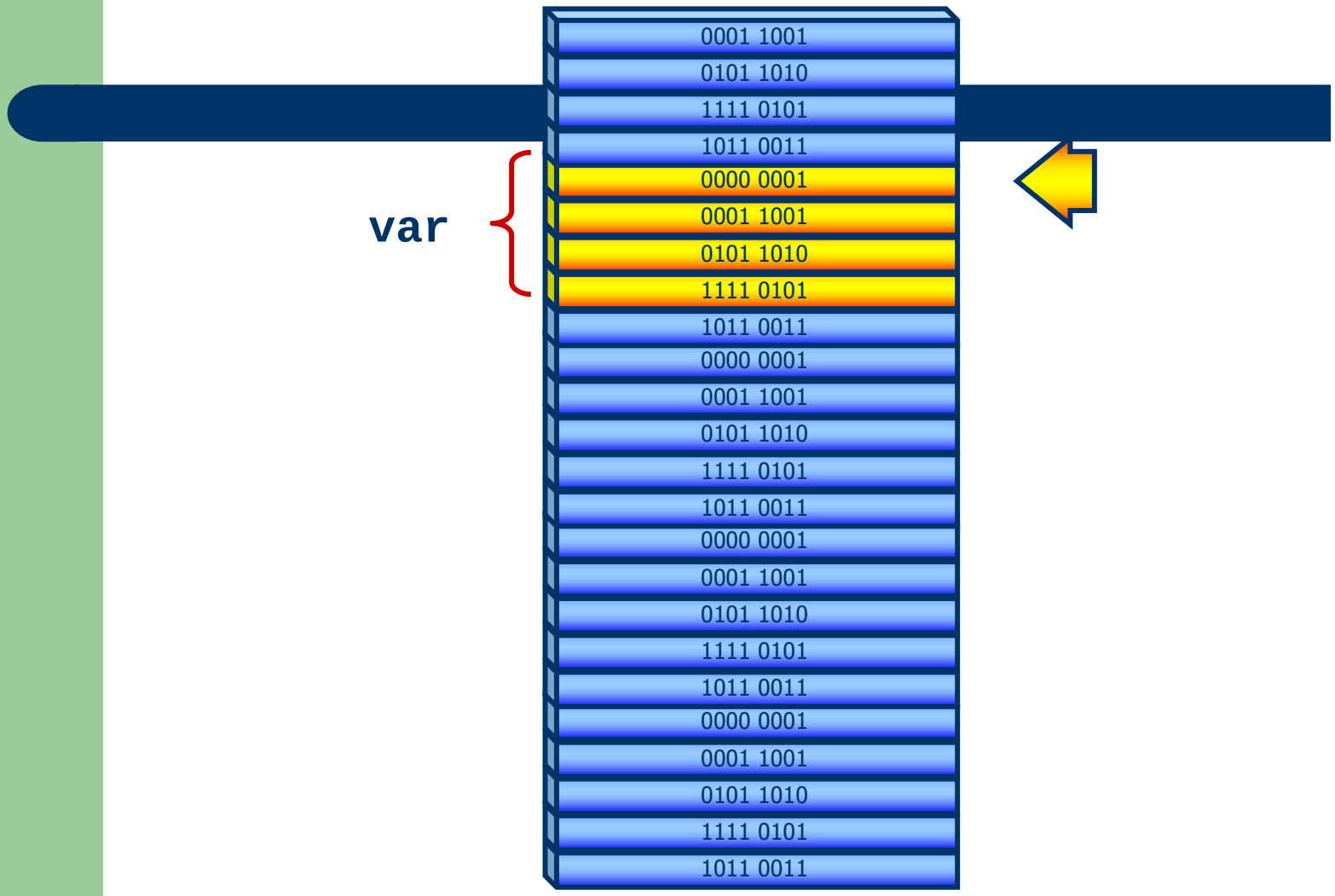
Passagem de Parâmetros :: Passagem por referência

- Os valores das variáveis externas não são passados para a função, mas sim os seus endereços.
- Ocorre alteração no valor das variáveis externas.
- Usamos os caracteres:

  & - indica o endereço da variável

  ***** - indica o conteúdo do apontador

Passagem de Parâmetros
:: Passagem por referência



Passagem de Parâmetros

- Uma desvantagem da passagem por valor é que, se um item de dados grande está sendo passado, copiar esses dados pode consumir um tempo de processamento considerável.

Passagem de Vetores

- Vetores têm um comportamento diferente quando usados como parâmetros ou valores de retorno de funções.
- O compilador interpreta o nome de um vetor como o endereço do primeiro elemento do vetor.
- Dessa forma, os vetores são sempre passados por referência, sem usar uma notação especial.

Passagem de Vetores :: Exemplo

```
<tipo> funcao(int vet[], ...)
{
      ...
}
```

Passagem de Vetores

- Ao passar um vetor como parâmetro, se ele for alterado dentro da função, as alterações ocorrerão no próprio vetor e não em uma cópia.
- Ao retornar um vetor como valor de retorno, não é feita uma cópia deste vetor.
- Assim, o vetor retornado pode desaparecer, se ele foi declarado no corpo da função.
- Ao passar um vetor como parâmetro, não é necessário fornecer o seu tamanho na declaração da função.
- Porém, é importante lembrar que o vetor tem um tamanho que deve ser considerado pela função durante a manipulação dos dados.

Passagem de Vetores

- Quando o vetor é multidimensional, a possibilidade de não informar o tamanho na declaração da função se restringe apenas à primeira dimensão.

```
void show_matriz(int mat[][10], int n_linhas)
{
    ...
}
```

Const

- Para indicar que um parâmetro de função não deve ser alterado, pode-se utilizar o qualificador **`const`**.
- Seu uso não inibe a alteração do parâmetro.
- Porém, se existir no corpo da função uma alteração de um parâmetro declarado como constante, o compilador gerará uma mensagem de advertência.

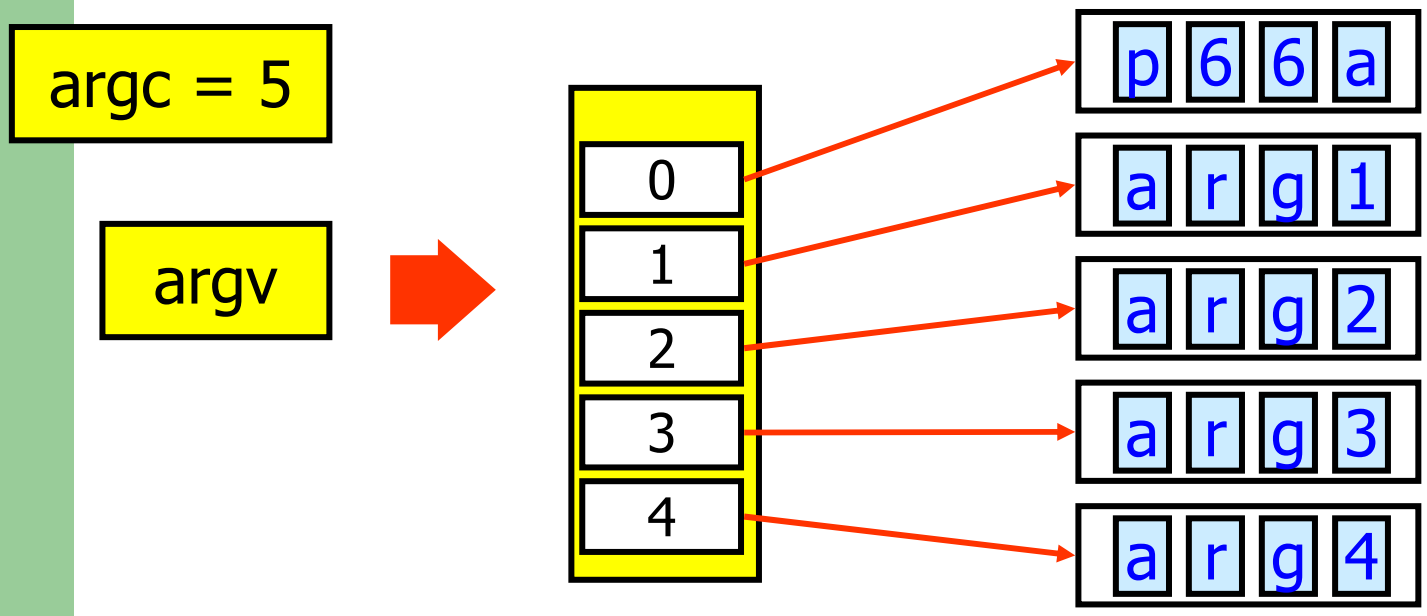
Parâmetros da função principal :: `argv`, `argc`

- Parâmetros
  - Argc (Número de argumentos recebidos)
    - É um número inteiro
    - Possui valor maior ou igual a 1(um)
  - Argv (Guarda os argumentos recebidos)
    - É um vetor de ponteiros para strings
    - O primeiro argumento, argv[0], é sempre o nome do programa

```
int main(int argc, char **argv)
```

Parâmetros da função principal :: `argv`, `argc`

- ```
  ./p66a arg1 arg2 arg3 arg4
  ```


  ```

Funções
:: Recursividade

- Um objeto é dito recursivo se pode ser

"Para fazer iogurte, você precisa de leite e de um pouco de iogurte."

"Para entender recursividade, você primeiro tem de entender recursividade."

Funções
:: Recursividade

- A recursão é uma forma interessante de resolver problemas, pois o divide em problemas menores de mesma natureza.

- Um processo recursivo consiste de duas partes:
  - O caso trivial, cuja solução é conhecida.
  - Um método geral que reduz o problema a um ou mais problemas menores de mesma natureza.

Exemplos....