UFABC

Funções

Prof. Fabrício Olivetti de França

Resposta do Exercício 4

```
while ( trocou ) {
        trocou = 0;
        for ( i=0; i<N; i++ ) {
                for ( j=0; j<N; j++ ) {
                        for ( k=0; k<N; k++ ) {
                                for ( p=0; p<v2[j]; p++ ) {
                                        pot *= v1[i];
                                        if ( pot == v3[k] ) {
                                                trocou = 1;
                                                v2[j] = p;
                                        }
                                }
                        }
                }
        }
}
```

UFABC

Resposta do Exercício 4

```
while ( trocou ) {
        trocou = 0;
        for ( i=0; i<N; i++ ) {
                for ( j=0; j<N; j++ ) {
                        for ( k=0; k<N; k++ ) {
                                for ( p=0; p<v2[j]; p++ ) {
                                        pot *= v1[i];
                                        if ( pot == v3[k] ) {
                                                trocou = 1;
                                                v2[j] = p;
                                        }
                                }
                        }
                }
        }
}
```

UFABC

BLOCOS (DES)ESTRUTURADOS

Em algoritmos mais complexos, mesmo com a ajuda de blocos de instruções, o código pode se tornar confuso e ilegível.

Um segundo passo da estruturação do código é a **modularização**, ou seja, dividir os blocos de instruções em **funções** próprias que realizem determinada tarefa.

UFABC

BLOCOS (DES)ESTRUTURADOS

Em algoritmos mais complexos, mesmo com a ajuda de blocos de instruções, o código pode se tornar confuso e ilegível.

O motivo para múltiplos níveis de indentação é que o código está tentando realizar múltiplas funções dentro de um mesmo bloco.

UFABC

BLOCOS (DES)ESTRUTURADOS

Um segundo passo da estruturação do código é a **modularização**, ou seja, dividir os blocos de instruções em **funções** próprias que realizam apenas uma única tarefa.

UFABC

Funções

Funções tem o objetivo de segmentar o código-fonte em pequenas tarefas individuais.

Elas devem ser pequenas, claras e realizar apenas uma única tarefa.

UFABC

Modularizando seu programa

Para criar uma solução algorítmica de seu problema é interessante criar um código **declarativo**.

A função principal deve conter, preferencialmente, apenas uma sequência de funções que descrevem a solução.

UFABC

Modularizando seu programa

Escreva um programa que receba como entrada várias sequência de dois números inteiros separados por espaço.

O primeiro número é o numerador e o segundo é o denominador.

O programa deve imprimir cada fração simplificada e terminar quando o denominador for zero.

UFABC

simplificaFracao.c

```
while( validar_entrada(x,y) ) {
        divisor = simplificar(x, y);
        printf(“%d %d\n”, x/divisor, y/divisor)
        scanf(“%d %d\n”, &x, &y);
}
```

UFABC

simplificaFracao.c

Primeiro eu pensei no que eu preciso ter para resolver o problema, agora eu penso em como resolver por partes.

O que deve ser feito em **validar_entrada**?

UFABC

simplificaFracao.c

```
int validar_entrada(x, y)
{
        if( y != 0 ) {
                return 1;
        }else{
                return 0;
        }
}
```

UFABC

simplificaFracao.c

ALTERNATIVA:

```
int validar_entrada(x, y)
{
        return y!=0;
}
```

UFABC

simplificaFracao.c

ALTERNATIVA:

```
int validar_entrada(x, y)
{
        return y;
}
```

UFABC

simplificaFracao.c

```
int simplificar(x, y)
{
        ???
}
```

UFABC

simplificaFracao.c

Para simplificar uma fração, basta calcular o máximo divisor comum entre os números e, em seguida, dividir esses números pelo mdc.

UFABC

simplificaFracao.c

```
int simplificar(x, y)
{
        int divisor;
        divisor = mdc(x,y);
        return divisor;
}
```

UFABC

simplificaFracao.c

A função **mdc** é simplesmente nosso algoritmo de Euclides :-)

UFABC

Funções em C

Na linguagem C as funções tem o seguinte formato:

```
tipo_retorno nome_função( lista_parâmetros )
{
        /* instruções */
}
```

UFABC

Funções em C

Função que recebe um parâmetro inteiro e um caractere e retorna um inteiro:

```
int função( int x, char y )
{
        /* instruções */
       return w;
}
```

UFABC

Funções em C

Toda função que retorna alguma coisa deve terminar com o comando **return**.

Esse comando instrui o computador a terminar a função e devolver o valor calculado para quem chamou essa função.

UFABC

Funções em C

Ex.:

```
float função( … )
{
        return 10.2;
}
…
    x = função( … ); /* x possui o valor 10.2 */
```

UFABC

Funções em C

A instrução **return** pode aparecer em qualquer momento dentro da função.

Na linguagem C as funções estão limitadas a retornarem apenas um único valor.

UFABC

Funções em C

As funções que não retornam nenhum valor são declaradas como void e são denominadas de **procedimentos**:

```
void funcao( ... )
{
...
}
```

UFABC

Procedimentos vs Funções

```
x = soma(2, 3);

printf("%d\n", x);
```

UFABC

Procedimentos vs Funções

```
x = soma(2, 3);      /* função */

printf("%d\n", x);  /* procedimento */
```

UFABC

Funções em C

As funções devem ser declaradas no início do código, logo após as diretivas #include:

tipo função(tipos);

UFABC

Escopo de Variáveis

```
int funcao( int x )
{
    int y;
}
int main( ){
    int z;
    z = funcao( 10 );
}
```

UFABC

Escopo de Variáveis

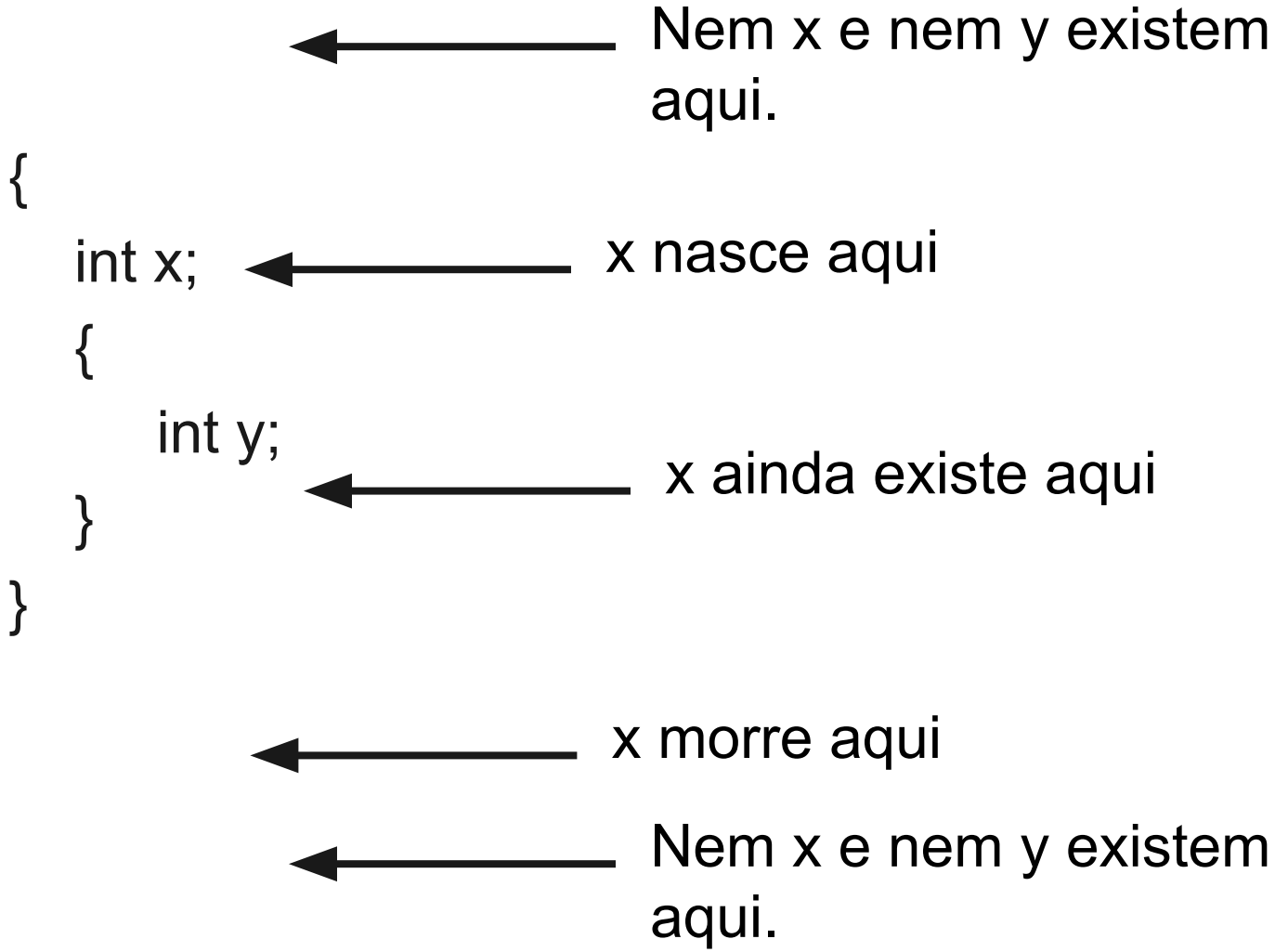
```
int funcao( int x )
{
    int y;
}
int main( ){
    int z;
    z = funcao( 10 );
}
```

y só existe dentro desse bloco.

z só existe dentro desse bloco.

UFABC

Escopo de Variáveis



UFABC

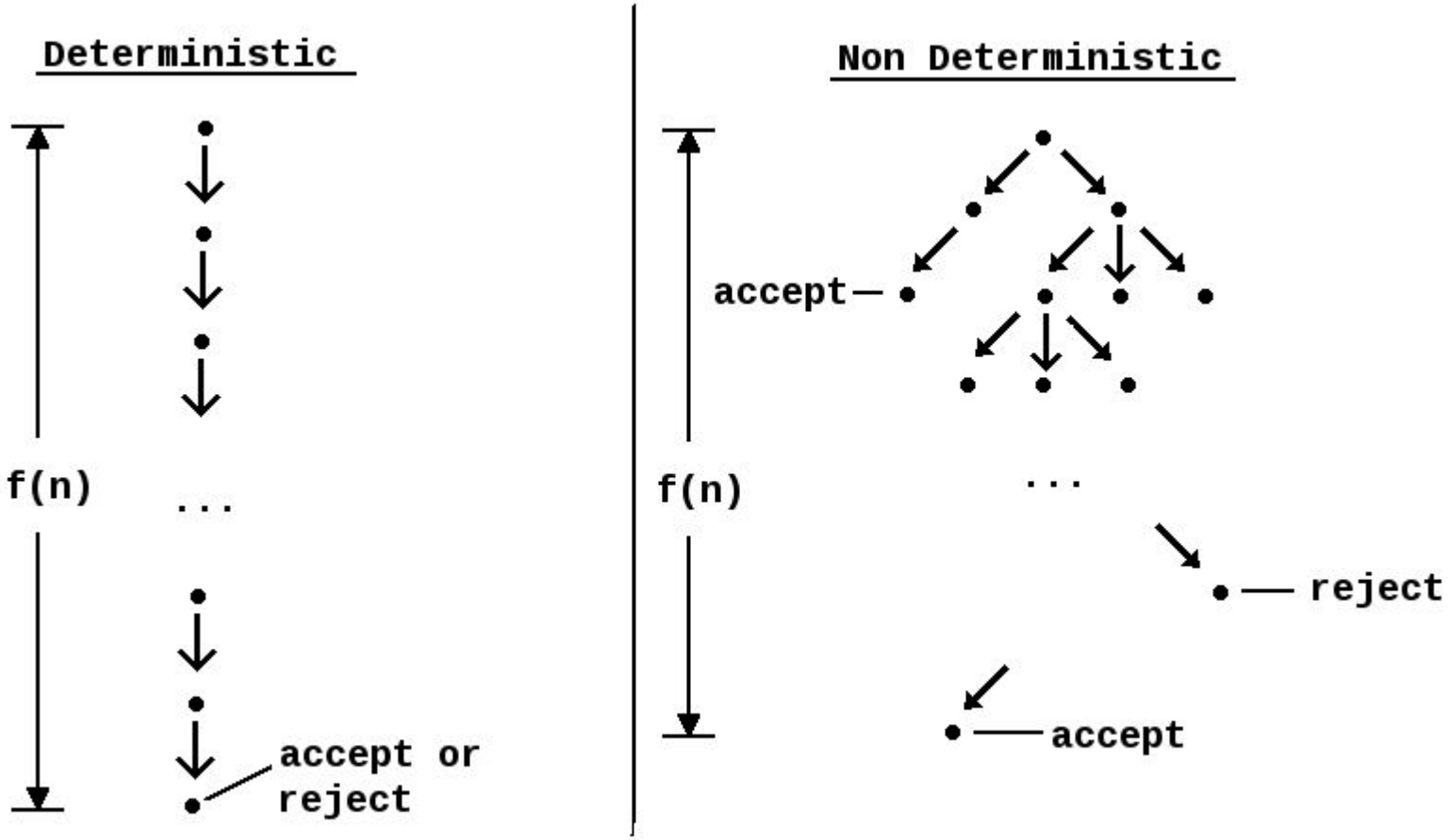
Algoritmos determinísticos vs não-determinísticos.

Idealmente um algoritmo retorna SEMPRE a mesma saída dada a mesma entrada.

Porém, alguns algoritmos, denominados não-determinísticos, podem variar sua saída a cada execução.

UFABC

Algoritmos determinísticos vs não-determinísticos.



UFABC

Algoritmos determinísticos vs não-determinísticos.

Ex.: algoritmos que utilizam elementos aleatórios, algoritmos que alteram variáveis globais ou estáticas.

UFABC

Algoritmos determinísticos vs não-determinísticos.

Exemplos:

```
hora_atual();
temperatura_atual();
assentos_disponiveis();
encontrar(x); /* com lista sendo global */
```

UFABC

Ingressos de cinema

Digamos que temos um sistema para venda de ingressos:

```
int vende_assento (int qtde)
{
  int assentos = 40; /* total de assentos */
  if (assentos < qtde) return -1;
  assentos -= qtde;
  return qtde;
}
```

UFABC

Ingressos de cinema

Sempre que chamar a função a variável terá o valor **40**.

```
int vende_assento (int qtde)
{
  int assentos = 40; /* total de assentos */
  if (assentos < qtde) return -1;
  assentos -= qtde;
  return qtde;
}
```

UFABC

Ingressos de cinema

A palavra-chave **static** indica que a variável será inicializada apenas na primeira chamada da função e será atualizada nas seguintes!

```
int vende_assento (int qtde)
{
  static int assentos = 40; /* total de assentos */
  if (assentos < qtde) return -1;
  assentos -= qtde;
  return qtde;
}
```

UFABC

Ingressos de cinema

```
int x = vende_assentos(2); /* x = 2, assentos = 38 */
int x = vende_assentos(4); /* x = 4, assentos = 34 */
int x = vende_assentos(2); /* x = 2, assentos = 32 */
…
int x = vende_assentos(2); /* x = 2, assentos = 1 */
int x = vende_assentos(2); /* x = -1, assentos = 1 */
```

UFABC

Problemas

O uso de variáveis de estado e o não-determinismo causado por ele pode dificulta a validação da corretude do algoritmo.

Pode introduzir **bugs** difíceis de serem detectados.

UFABC

Mas...

Esse tipo de algoritmo é necessário para encontrar soluções de alguns problemas importantes na computação.

Criptografia depende fortemente de algoritmos não-determinísticos.

UFABC

Convenção

Funções devem:

- Ocupar idealmente até duas páginas na tela de seu editor.
- Cumprir apenas um único papel/objetivo.
- Possuir no máximo 5 (quando muito 10) variáveis internas.

UFABC

Convenção

A declaração, diferente dos blocos de instruções, deve colocar o “abre chaves” na linha seguinte do nome da função.

O nome da função deve ser todo minúsculo e os espaços substituídos por “_”, o nome deve representar uma ação.

UFABC

Convenção

A função **main** deve retornar um inteiro, sendo **0** quando ela termina da forma esperada e um código representando o erro de execução.

UFABC