REVISÃO DE C (CONTINUAÇÃO)

DCE792 - AEDs II (Prática)

Atualizado em: 10 de agosto de 2024

Iago Carvalho

Departamento de Ciência da Computação

FUNÇÕES

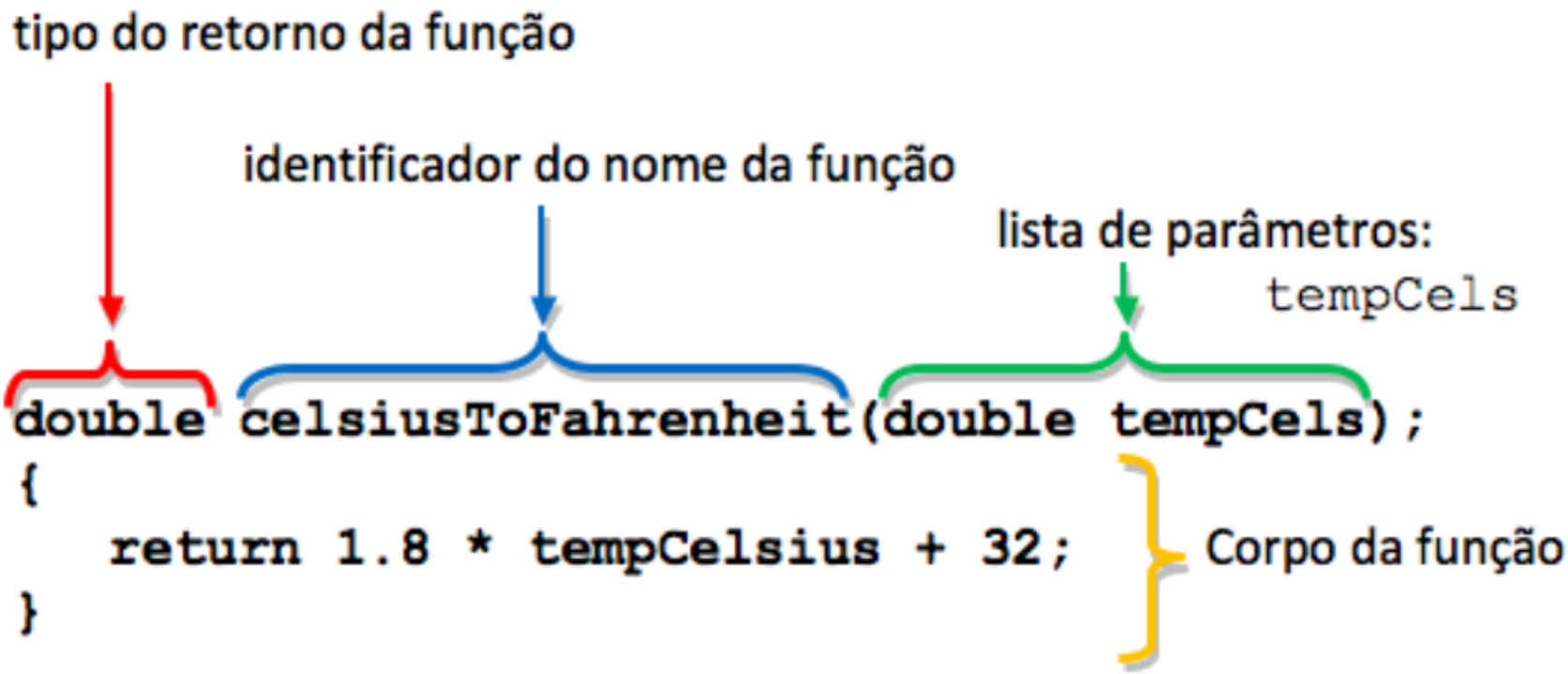
Uma função em C ou C++ pode ser descrita como

```
1 <tipo_retorno> <nome_função>(<lista_declaração_parâmetro>)
2 {
3     <corpo_função>
4 }
```

onde

- <tipo_retorno> é o tipo do valor que a função retorna
  - Caso a função não retorne nada, o tipo dela é void
- <nome_função> é o identificador que nomeia a função
- <lista_declaração_parâmetro> é uma lista, possivelmente vazia, de declarações separadas por vírgulas, dos parâmetros da função
- <corpo_função> descreve o comportamento da função

EXEMPLO DE UMA FUNÇÃO



O que esta função faz?

IMPLEMENTAÇÃO DA FUNÇÃO

```
1  // definição da função
2  double celsiusToFahrenheit(double tempCels)
3  {
4      double f;
5      f = 1.8 * tempCels + 32;
6      return f;
7  }
```

```
1  // definição da função
2  double celsiusToFahrenheit(double tempCels)
3  {
4      return 1.8 * tempCels + 32;
5  }
```

COMO USAR UMA FUNÇÃO

```c
#include <stdio.h>

// protótipo da função
double celsiusToFahrenheit(double tempCels);

// método main (principal)
int main()
{
    double tempC, tempF;
    printf("Conversão Celsius para Fahrenheit\n");
    printf("(valor menor que -273.15 encerra o programa)\n\n");
    printf("Temperatura em Celsius: ");
    scanf("%lf", &tempC);

    if (tempC >= -273.15) {
        tempF = celsiusToFahrenheit(tempC);
        printf("%lf graus Celsius = %lf graus Fahrenheit.\n",
                tempC, tempF);
    }
   return 0;
}
```

PASSAGEM POR VALOR E POR PARÂMETRO

MODIFICANDO A VARIÁVEL DENTRO DA FUNÇÃO

Vocês podem nem ter notado, mas já fizeram isso anteriormente.

- Veja a utilização da função *scanf* abaixo

```
1  int main()
2  {
3      int x;
4      scanf("%d", &x); // passamos o endereço de memória de x: &x
5
6      if (x % 2 == 0)
7          printf("%d é um número par!\n", x);
8      else
9          printf("%d é um número ímpar!\n", x);
10 }
```

PASSAGEM POR VALOR

A função abaixo está correta?

```
1  void naoTroca(int a, int b)
2  {
3      int aux = a;
4      a = b;
5      b = aux;
6  }
```

PASSAGEM POR VALOR

```
1  void naoTroca(int a, int b)
2  {
3      int aux = a;
4      a = b;
5      b = aux;
6  }
```

Ela está errada!

- Os parâmetros são passados por valor
- Isto significa que somente cópias de textcolorbluea e textcolorblueb são passadas para a função
- Assim, a função não troca os valores de fato

Como arruma-la?

PASSAGEM POR PONTEIROS

Podemos fazer a passagem por ponteiros (em C e C++) ou passagem por referência (C++)

```
1  void troca(int *a, int *b)
2  {
3      int aux = *a;
4      *a = *b;
5      *b = aux;
6  }
```

Esta função recebe o endereço de memória de duas variáveis

- Troca o valor contido em ambos os endereços de memória

USO DE VALOR E PONTEIROS

```c
int main()
{
    int a = 1;
    int b = 2;
    naoTroca(a, b); // valores a e b são passados (e não há troca)
    printf("a = %d, b = %d", a, b); // a = 1, b = 2
}
```

```c
int main()
{
    int a = 1;
    int b = 2;
    troca(&a, &b); // endereço de memória de a e b são passados
    printf("a = %d, b = %d", a, b); // a = 2, b = 1
}
```

PASSAGEM POR PONTEIROS E POR REFERÊNCIA

Referência

```cpp
#include <iostream>
using namespace std;
void doubleIt(int&);//prototype com endereço de variavel
int main ()
{
  int num;
  cout << "Enter number: ";
  cin >> num;
  cin.get();
  doubleIt(num);   //chamo função, passando parametro num
  cout << "The number doubled in main is " << num << endl;
  cout<<"Digite enter para continuar..."<<endl;
  cin.get();
  return 0;
}
void doubleIt (int& x)
{
  cout << "The number to be doubled is " << x << endl;
  x *= 2;
  cout << "The number doubled in doubleIt is " << x << endl;
}
```

Ponteiros

```cpp
#include <iostream>
using namespace std;
void doubleIt(int*);  //parametro por endereço
int main ()
{
  int num;
  cout << "Enter number: ";
  cin >> num;
  cin.get();
  doubleIt(&num);//passei parametro como endereço
  cout << "The number doubled in main is " << num << endl;
  cout<<"Digite enter para continuar..."<<endl;
  cin.get();
  return 0;
}
void doubleIt (int* x)
{
  cout << "The number to be doubled is " << *x << endl;
  *x *= 2;
  cout << "The number doubled in doubleIt is " << *x << endl;
}
```

PONTEIROS

PONTEIROS

Um ponteiro (apontador ou pointer) é um tipo especial de variável que armazena um endereço de memória

- Um ponteiro pode ser declarado utilizando o caractere especial *

```
1  int *pi;    // pi é um ponteiro do tipo int
2  char *pc;   // pc é um ponteiro do tipo char
3  float *pf;  // pf é um ponteiro do tipo float
4  double *pd; // pd é um ponteiro do tipo double
```

```
1  int *p1, *p2, *p3;
```

PARA ONDE O PONTEIRO APONTA

O **conteúdo** da memória apontada por um ponteiro se refere ao valor armazenado no endereço de memória para o qual o ponteiro aponta

Pode-se acessar ou alterar este conteúdo (ou valor) utilizando o mesmo caractere especial *

```
1  int main()
2  {
3      int x = 10, y = 100;
4      int *px = &x;
5      *px = *px + 1; // conteúdo de px recebe o conteúdo de px mais 1
6      printf("x = %d", x);
7      printf("y = %d", y);
8      return 0;
9  }
```

TIPOS DE PONTEIROS

Observe os ponteiros abaixo

```
1  int main()
2  {
3      char *c;
4      int *i;
5      float *f;
6      double *d;
7      return 0;
8  }
```

- Todos requerem o mesmo tamanho
- Armazenam um endereço de memória, independente do tipo
- Tipo é checado pelo compilador

Endereço	Valor	
00010000	??	c
00010001		
00010002		
00010003		
00010004	??	i
00010005		
00010006		
00010007		
00010008	??	f
00010009		
0001000A		
0001000B		
0001000C	??	d
0001000D		
0001000E		
0001000F		

PONTEIROS

```
1  void naoTroca(int x, int y)
2  {
3      int aux;
4      aux = x;
5      x = y;
6      y = aux;
7  }
8
9  void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1  void naoTroca(int x, int y)
2  {
3      int aux;
4      aux = x;
5      x = y;
6      y = aux;
7  }
8
9  void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1  void naoTroca(int x, int y)
2  {
3      int aux;
4      aux = x;
5      x = y;
6      y = aux;
7  }
8
9  void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	main
0x1012			
0x1016		x	naoTroca
0x1020		y	naoTroca
0x1024			
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	main
0x1012			
0x1016	100	x	naoTroca
0x1020	200	y	naoTroca
0x1024			
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016	100	x	naoTroca
0x1020	200	y	
0x1024		aux	
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016	100	x	naoTroca
0x1020	200	y	
0x1024	100	aux	
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	main
0x1012			
0x1016	200	x	naoTroca
0x1020	200	y	naoTroca
0x1024	100	aux	naoTroca
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016	200	x	naoTroca
0x1020	100	y	
0x1024	100	aux	
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	main
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032		px	troca
0x1036		py	troca
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	main
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032	0x1004	px	troca
0x1036	0x1008	py	troca
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032	0x1004	px	troca
0x1036	0x1008	py	
0x1040		aux	
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1  void naoTroca(int x, int y)
2  {
3      int aux;
4      aux = x;
5      x = y;
6      y = aux;
7  }
8
9  void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	100	x	main
0x1008	200	y	
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032	0x1004	px	troca
0x1036	0x1008	py	
0x1040	100	aux	
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	200	x	main
0x1008	200	y	main
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032	0x1004	px	troca
0x1036	0x1008	py	troca
0x1040	100	aux	troca
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1   void naoTroca(int x, int y)
2   {
3       int aux;
4       aux = x;
5       x = y;
6       y = aux;
7   }
8
9   void troca(int *px, int *py)
10  {
11      int aux;
12      aux = *px;
13      *px = *py;
14      *py = aux;
15  }
16
17  int main()
18  {
19      int x = 100, y = 200;
20      naoTroca(x, y);
21      printf("x=%d, y=%d\n", x, y);
22      troca(&x, &y);
23      printf("x=%d, y=%d\n", x, y);
24
25      return 0;
26  }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	200	x	main
0x1008	100	y	main
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032	0x1004	px	troca
0x1036	0x1008	py	troca
0x1040	100	aux	troca
0x1044			
0x1048			
0x1052			
0x1056			

PONTEIROS

```
1  void naoTroca(int x, int y)
2  {
3      int aux;
4      aux = x;
5      x = y;
6      y = aux;
7  }
8
9  void troca(int *px, int *py)
10 {
11     int aux;
12     aux = *px;
13     *px = *py;
14     *py = aux;
15 }
16
17 int main()
18 {
19     int x = 100, y = 200;
20     naoTroca(x, y);
21     printf("x=%d, y=%d\n", x, y);
22     troca(&x, &y);
23     printf("x=%d, y=%d\n", x, y);
24
25     return 0;
26 }
```

Endereço	Conteúdo	Nome	
0x1000			
0x1004	200	x	main
0x1008	100	y	
0x1012			
0x1016			
0x1020			
0x1024			
0x1028			
0x1032			
0x1036			
0x1040			
0x1044			
0x1048			
0x1052			
0x1056			

ALOCAÇÃO DINÂMICA DE MEMÓRIA

ALOCAÇÃO DINÂMICA

Utilizada quando não se sabe de antemão quanto espaço de memória será necessário

- Por exemplo, quando não é possível determinar o tamanho de um vetor de antemão

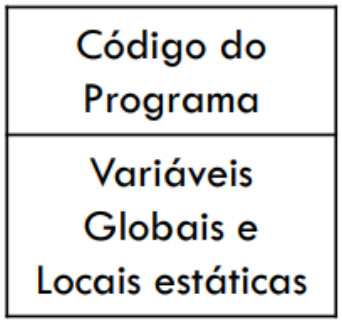
Alocação dinâmica possui algumas propriedades interessantes

- Espaço de memória é requisitado em tempo de execução
- Espaço permanece alocado até que seja explicitamente liberado
- Depois de liberado, espaço pode ser disponibilizado para outros usos e não pode mais ser acessado
- Espaço alocado e não liberado explicitamente é liberado ao final da execução do programa

ALOCAÇÃO ESTÁTICA X DINÂMICA

Alocação Estática

Memória Principal:



Alocação Dinâmica

Memória Principal:

BIBLIOTECA PARA ALOCAÇÃO DINÂMICA

Alocação dinâmica é realizada pela biblioteca stdlib.h

As funções mais importantes são

- *malloc*: Alocação dinâmica de memória
- *calloc*: Alocação dinâmica de memória
- *free*: Liberação de memória previamente alocada
- *sizeof*: Retorna o tamanho de um tipo

MALLOC E *CALLOC*

Ambas as funções são utilizadas para alocar memória dinamicamente. Entretanto, elas possuim algumas diferenças

	malloc	calloc
Significado	memory alloc	contiguous alloc
Argumentos	Um único argumento	Dois argumentos
Velocidade	Rápido	Não tão rápido assim
Conteúdo	Lixo de memória	Inicializado com zero
Memória	Bloco de memória único de um tamanho fixo	Múltiplos blocos de memória iguais

USO DE *MALLOC* E *CALLOC*

malloc(tamanho do bloco de memória)

- *malloc*(n * sizeof(int))

calloc(número de blocos, tamanho de cada bloco)

- *calloc*(n, sizeof(int))

EXEMPLO DE USO

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
   int i, n;
   int *a;

   printf("Number of elements to be entered:");
   scanf("%d",&n);

   a = (int*)calloc(n, sizeof(int));
   printf("Enter %d numbers:\n",n);
   for( i=0 ; i < n ; i++ ) {
      scanf("%d",&a[i]);
   }

   printf("The numbers entered are: ");
   for( i=0 ; i < n ; i++ ) {
      printf("%d ",a[i]);
   }
   free( a );

   return(0);
}
```

DIRETIVAS DE COMPILAÇÃO E BIBLIOTECAS

DIRETIVAS *INCLUDE* E *DEFINE*

#include

- Utilizada para incluir outro arquivo em nosso código fonte
- Útil para inserir bibliotecas ou fragmentar nosso código
- Pre-processamento do compilador substitui a diretiva pelo código que ela faz referência

#define

- Em sua forma mais simples, define constantes simbólicas com nomes mais apropriados
- Quando um identificador é associado a um #define, todas as suas ocorrências no código-fonte são substituídas pelo valor da constante
- Note que #define também pode ser utilizado para criar diretivas mais elaboradas, inclusive aceitando argumentos, chamadas *Macros*.

INCLUDE E DEFINE

```c
// incluindo a biblioteca stdio
#include <stdio.h>

// definindo o valor de PI
#define PI 3.141592

// definindo o que é um 'beep'
// (obs: há formas mais elaboradas de fazer um 'beep')
#define BEEP "\x07"

int main()
{
    printf("pi = %d\n", PI);
    printf(BEEP);
    return 0;
}
```

BIBLIOTECA MATEMÁTICA

A math.h é a principal biblioteca matemática da linguagem

- Para utilizar esta biblioteca, deve-se incluila utilizando #include <math.h>

Ela oferece, dentre outras coisas, funções para

- Arredondamento
- Potência
- Trigonometria
- Exponenciação e logaritmos

ARREDONDAMENTO

Função	Descrição	Exemplo
`double ceil(x)`	arredonda `x` para cima	`ceil(9.1)` → 10.0
`double floor(x)`	arredonda `x` para baixo	`floor(9.8)` → 9.0
`double round(x)`	arredonda `x`	`round(9.5)` → 10.0 `round(9.4)` → 9.0
`double trunc(x)`	retorna a parte inteira de `x`	`trunc(9.8)` → 9.0

POTÊNCIA

Função	Descrição	Exemplo
`double pow(x, y)`	x elevado a y: x^y	`pow(3, 2)` $\rightarrow$ 9.0
`double sqrt(x)`	raiz quadrada de `x`: $\sqrt{x}$	`sqrt(25)` $\rightarrow$ 5.0
`double cbrt(x)`	raiz cúbica de `x`: $\sqrt[3]{x}$	`cbrt(27)` $\rightarrow$ 3.0

TRIGONOMETRIA

Função	Descrição	Exemplo
`double cos(x)`*	retorna o cosseno `x`	`cos(1.047)` → 0.5
`double sin(x)`*	retorna o seno `x`	`sin(1.571)` → 1.0
`double tan(x)`*	retorna a tangente `x`	`tan(0.785)` → 1.0
`double acos(x)`**	retorna o arco cosseno	`acos(0.5)` → 1.047
`double asin(x)`**	retorna o arco seno	`asin(1.0)` → 1.571
`double atan(x)`**	retorna o arco tangente	`atan(1.0)` → 0.785

* valores em radianos

** valores de x entre $[-1, 1]$

EXPONENCIAÇÃO E LOGARITMOS

Função	Descrição	Exemplo
`double exp(x)`	retorna exponencial de `x`: e^x	`exp(5)` $\rightarrow$ 148.4
`double log(x)`	logaritmo natural de `x`: $\ln(x)$	`log(5.5)` $\rightarrow$ 1.7
`double log10(x)`	logaritmo de `x`: $\log(x)$	`log10(1000)` $\rightarrow$ 3.0

OUTRAS FUNÇÕES DA *STDLIB.H*

Esta biblioteca também fornece métodos para conversão de tipos de variáveis

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i = atoi("-10"); // atoi converte string -> int
    float f = atof("10.5"); // atof converte string -> float
    double d = strtod("10.5",  NULL); // strtod converte string -> double

    system("clear"); // executa o comando clear no terminal

    srand(0); // seleciona a semente para geração de nros aleatórios
    int r = rand(); // r recebe um nro aleatório
    printf("Número aleatório: %d\n\n", r);

    printf("i = %d, f = %f, d = %lf\n", i, f, d);
    printf("Valor absoluto de i: %d\n\n", abs(i));

    exit(0); // função que finaliza o programa imediatamente
    return 0;
}
```

STRUCTS

TIPOS PERSONALIZADAS

Existem diversos tipos previamente definidos em C

- *int*, *float*, *char*, ...

É possível criar novos tipos de dados utilizando a palavra reservada struct

- Informa ao compilador o nome, o tamanho em *bytes* e a maneira como ela deve ser armazenada e recuperada da memória

Entretanto, o uso mais útil do struct é para definir estruturas de dados heterogêneas

- Estrutura que agrupa itens de dados de diferentes tipos

ESTRUTURA ALUNO

```c
#include <stdio.h>

struct Aluno {
    int nMat;       // número de matrícula
    float nota[3]; // três notas
    float media;    // média aritmética
};                  // fim da definição da estrutura (com ;)

int main()
{
    struct Aluno bart;   // declara a variável do tipo 'struct Aluno'
    bart.nMat = 1521001;
    bart.nota[0] = 8.5;
    bart.nota[1] = 9.5;
    bart.nota[2] = 10.0;
    bart.media = ( bart.nota[0]+ bart.nota[1] + bart.nota[2] ) / 3.0;
    printf("Matrícula: %d\n", bart.nMat);
    printf("Média    : %.1f\n", bart.media);
    return 0;
}
```

ESTRUTURA ALUNO

A instrução struct Aluno bart; declara uma variável bart do tipo struct Aluno

- Memória é reservada para os membros: 4 *bytes* para nMat, 12 *bytes* para a matriz nota (3 *floats*) e 4 *bytes* para media

Os membros da estrutura são armazenados em sequência na memória

O operador ponto (.) conecta o nome de uma variável de estrutura a um membro dela

As declarações de uma variável simples e de uma variável de estrutura seguem o mesmo formato

```
1 struct Aluno bart;
2 struct Aluno *ponteiroAluno;
3 int home;
```

DEFINIÇÃO DE UMA ESTRUTURA

A definição de uma estrutura **não** cria uma variável

- Ela literalmente cria um tipo de dados novo
- Informa ao compilador o nome, o tamanho e como os daods devem ser armazenados em memória
- Não reserva memória

Em resumo, é um tipo de dados definido pelo programador

- Tipo adicional não presente anteriormente na linguagem

INICIALIZAÇÃO DE UMA ESTRUTURA

```c
struct Data {
    int  dia;
    char mes[10];
    int  ano;
};

struct Data natal = { 25, "Dezembro", 2016 };
struct Data niver = { 20, "Outubro",  1986 };
```

As variáveis são inicializadas juntamente com suas declarações

- Os valores atribuidos a cada variável devem ser inseridos **exatamente na mesma ordem** que foram definidos na estrutura
- Os valores tem que ser separados por vírgula e estar contido entre chaves

ATRIBUIÇÃO DE ESTRUTURAS

Uma variável estrutura pode ser atribuída à outra variável do mesmo tipo por meio de uma atribuição simples

```
1  struct Data natal = { 25, "Dezembro", 2016 };
2
3  struct Data natalDesteAno;
4  natalDesteAno = natal;
```

COMANDO *TYPEDEF*

O comando typedef é muito útil quando estamos trabalhando com estruturas

- Define um apelido (*alias*) para um tipo
- Simplificam a utilização de estruturas heterogêneas
- Permite **omitir** a palavra struct ao declarar variáveis

```c
typedef struct { // não precisamos definir o nome aqui
    int  dia;
    char mes[10];
    int  ano;
} Data;          // 'apelido' (novo nome) para a estrutura: Data
```

```c
Data natal = { 25, "Dezembro", 2016 };

Data natalDesteAno;
natalDesteAno = natal;
```

OPERAÇÕES COM ESTRUTURAS HETEROGÊNEAS

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

int main()
{
    Venda A = {20, 110.0};
    Venda B = {3, 258.0};
    Venda total;

    // soma membro a membro
    total.pecas = A.pecas + B.pecas;
    total.preco = A.preco + B.preco;
}
```

Erro comum

```c
// ERRO!
total = A + B;
```

ESTRUTURAS ANINHADAS

```c
typedef struct {
    int  dia;
    char mes[10];
    int  ano;
} Data;

typedef struct {
    int   pecas;
    float preco;
    Data  diaVenda;
} Venda;

int main()
{
    // exemplo de declaração
    Venda v = {20, 110.0, {7, "Novembro", 2015} };

    // exemplo de uso:
    printf("Ano da venda: %d", v.diaVenda.ano);

    return 0;
}
```

UTILIZAÇÃO DE ESTRUTURAS EM FUNÇÕES

Uma estrutura pode ser passada como argumento para funções

- Passagem por valor
- Passagem por ponteiros
- Passagem por referência (C++)

Funciona da mesma maneira que a passagem de variáveis

- Passagem por valor envia uma cópia da estrutura
  - Modificações de valores dentro da função não são refletidos na estrutura original
- Passage por ponteiros ou referência envia o endereço de memória inicial da estrutura
  - Permite a modificação dos valores dos itens da estrutura

PASSAGEM DE ESTRUTURAS POR VALOR

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

void imprimeTotal(Venda v1, Venda v2)
{
    Venda total = {0, 0.0};
    total.pecas = v1.pecas + v2.pecas;
    total.preco = v1.preco + v2.preco;
    printf("Nro peças:   %d\n", total.pecas);
    printf("Preço total: %.2f\n", total.preco);
}

int main()
{
    Venda v1 = {1, 20}, v2 = {3, 10};
    imprimeTotal(v1, v2);
    return 0;
}
```

PASSAGEM DE ESTRUTURAS POR PONTEIROS

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

void imprimeTotal(Venda *v1, Venda *v2)
{
    Venda total = {0, 0.0};
    total.pecas = (*v1).pecas + (*v2).pecas;
    total.preco = (*v1).preco + (*v2).preco;
    printf("Nro peças:   %d\n", total.pecas);
    printf("Preço total: %.2f\n", total.preco);
}

int main()
{
    Venda v1 = {1, 20}, v2 = {3, 10};
    imprimeTotal(&v1, &v2);
    return 0;
}
```

PRÓXIMA AULA:

NOÇÕES DE COMPLEXIDADE DE ALGORITMOS