Algoritmos de Programação 2

Professor: José Eurípedes Ferreira de Jesus Filho
jeferreirajf@gmail.com

Universidade Federal de Jataí – UFJ

Agenda

- Introdução.
- Variáveis em memória.
- Ponteiros.
- Ponteiros para registros.
- Exercícios.

Introdução

- Ponteiros rotineiramente se tornam o terror de todo programador.
- Contudo, ponteiros são extremamente necessários para um bom programador.
- É mais SIMPLES do que você imagina.
- Basta você entender que ponteiros são variáveis como quaisquer outra. O segredo é o que está dentro dele!

Variáveis na memória

- Ao declarar uma variável em C, o sistema operacional deve reservar o espaço necessário para esta variável.
- O próprio SO é encarregado de vincular o identificador que o programador atribuiu a variável com o respectivo endereço de memória dela.

```
int x = -1;
```

Identificador	Tipo	Endereço
x	int	0x04

Tabela SO

Memória

Endereço	Memória
0x00	
0x04	-1
0x08	
0x12	
0x16	
...	
0x99	

- Um inteiro em C ocupa 16 bits na memória.

Variáveis na memória

- Portanto, uma variável possui seu identificador, seu endereço na memória e seu valor, de fato.

```
int x = -1;
```

Identificador	Tipo	Endereço
x	int	0x04

Tabela SO

Memória

Endereço	Memória
0x00	
0x04	-1
0x08	
0x12	
0x16	
...	
0x99	

Ponteiros

- Um ponteiro também é uma variável!
- Em C, um ponteiro nada mais é do que um tipo especial de variável inteira!

```
int x = -1;
int *xPont;
```

Identificador	Tipo	Endereço
x	int	0x04
xPont	*int	0x16

Tabela SO

Memória

Endereço	Memória
0x00	
0x04	**-1**
0x08	
0x12	
0x16	**(lixo)**
0x20	
0x24	

Ponteiros

- A grande diferença é que ponteiros armazenam endereços de memória.

```
int x = -1;
int *xPont = &x;
```

Identificador	Tipo	Endereço
x	int	0x04
xPont	*int	0x16

Tabela SO

Memória

Endereço	Memória
0x00	
0x04	**-1**
0x08	
0x12	
0x16	**0x04**
0x20	
0x24	

Ponteiros

- Com o seguinte código, teremos o seguinte resultado:

```
int x = -1;
int *xPont = &x;
printf(“%p\n”, xPont);
printf(“%d\n”, (*xPont));
```

>> 0x04

>> -1

Identificador	Tipo	Endereço
x	int	0x04
xPont	*int	0x16

Tabela SO

Memória

Endereço	Memória
0x00	
0x04	**-1**
0x08	
0x12	
0x16	**0x04**
0x20	
0x24	

Ponteiros

- Assim, sabemos agora que para acessar o conteúdo no qual um ponteiro “aponta”, devemos usar (*). (Existem variações!)

```
int x = -1;
int *xPont = &x;
printf(“%p\n”, xPont);
printf(“%d\n”, (*xPont));
```

>> 0x04
>> -1

Identificador	Tipo	Endereço
x	int	0x04
xPont	*int	0x16

Tabela SO

Memória

Endereço	Memória
0x00	
0x04	-1
0x08	
0x12	
0x16	0x04
0x20	
0x24	

Ponteiros

- É importante colocar o tipo de dado para o qual o ponteiro irá apontar.

```
float x = -1;
float *xPont = &x;
printf(“%p\n”, xPont);
printf(“%f”, (*xPont));
```

>> 0x00
>> -1

Identificador	Tipo	Endereço
x	float	0x00
xPont	*float	0x32

Tabela SO

Memória

Endereço	Memória
0x00	-1
0x04	
0x08	
0x12	
0x16	(lixo)
0x20	
0x24	

Ponteiros

- Isso geraria um erro. Por que isso acontece?

```
float x = -1;
int *xPont = &x;
printf(“%f\n”, x);
printf(“%d”, (*xPont));
```

```
>> -1
>> -1082130432
```

Identificador	Tipo	Endereço
x	float	0x00
xPont	*int	0x16

Tabela SO

Memória

Endereço	Memória
0x00	-1
0x04	
0x08	
0x12	
0x16	(lixo)
0x20	
0x24	

Ponteiros

- Sumarizando até agora:
  - ✓ Ponteiros também são variáveis.
  - ✓ Armazenam o endereço de memória.
  - ✓ Para acessar a memória cujo ponteiro armazena o endereço, devemos usar o operador unário *.
  - ✓ Temos que declarar corretamente os tipos do ponteiro.
  - ✓ Para pegar o endereço de uma variável utilizamos o operador unário &.
- Cuidado nas expressões! * também é o símbolo da multiplicação!

Ponteiros para registros

- Podemos declarar um ponteiro que aponta para um registro!
- Para acessar os elementos da **`struct`** através do ponteiro, devemos utilizar **`->`** ao invés de ponto.

Exemplo!

Exercícios

- O que são ponteiros?

Exercícios

- Qual(is) a(s) diferença(s) entre um ponteiro para um inteiro e um inteiro?

Exercícios

- Existem riscos ao utilizar ponteiros? Quais?

Exercícios

- Escreva um programa que declare e inicialize uma variável inteira e um ponteiro para essa variável. Em seguida, multiplique o valor da variável por 2 usando o ponteiro e imprima o novo valor no terminal.

Exercícios

- Escreva uma função em C chamada **`swap`** que troque os valores de duas variáveis inteiras usando ponteiros. Em seguida, escreva um programa principal que teste essa função.

Exercícios

- Escreva um programa em C que declare uma estrutura chamada **`s_person`** com campos para `nome`, `e-mail` e `idade`. Crie um tipo baseado nessa estrutura chamado `Person`. Em seguida, crie uma variável deste tipo. Declare também um ponteiro para essa variável e atribua valores aos campos usando o ponteiro. Imprima os valores da estrutura usando o ponteiro.

Próxima aula

- Alocação estática e alocação dinâmica.