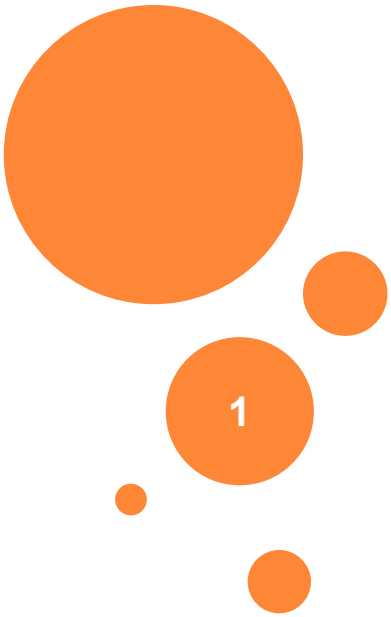


LINGUAGEM C: PONTEIROS

Prof. Humberto Razente

DEFINIÇÃO

- Variável
  - É um espaço reservado de memória usado para guardar um ***valor*** que pode ser modificado pelo programa;

- Ponteiro
  - É um espaço reservado de memória usado para guardar o ***endereço de memória*** de uma outra variável.
  - Um ponteiro é uma variável como qualquer outra do programa – sua diferença é que ela não armazena um valor inteiro, real, caractere ou booleano.
  - Ela serve para armazenar endereços de memória (são valores inteiros sem sinal).

Declaração

- Como qualquer variável, um ponteiro também possui um tipo

- É o ***asterisco*** (*) que informa ao compilador que aquela variável não vai guardar um valor mas sim um endereço para o tipo especificado.

```
int x;
float y;
struct ponto p;

int *x;
float *y;
struct ponto *p;
```

DECLARAÇÃO

- Na linguagem C, quando declaramos um ponteiro nós informamos ao compilador para que tipo de variável vamos apontá-lo

  - Um ponteiro **int*** aponta para um inteiro, isto é, **int**

  - Esse ponteiro guarda o endereço de memória onde se encontra armazenada uma variável do tipo **int**

INICIALIZAÇÃO

- Ponteiros apontam para uma posição de memória
  - **Cuidado**: Ponteiros não inicializados apontam para um lugar indefinido

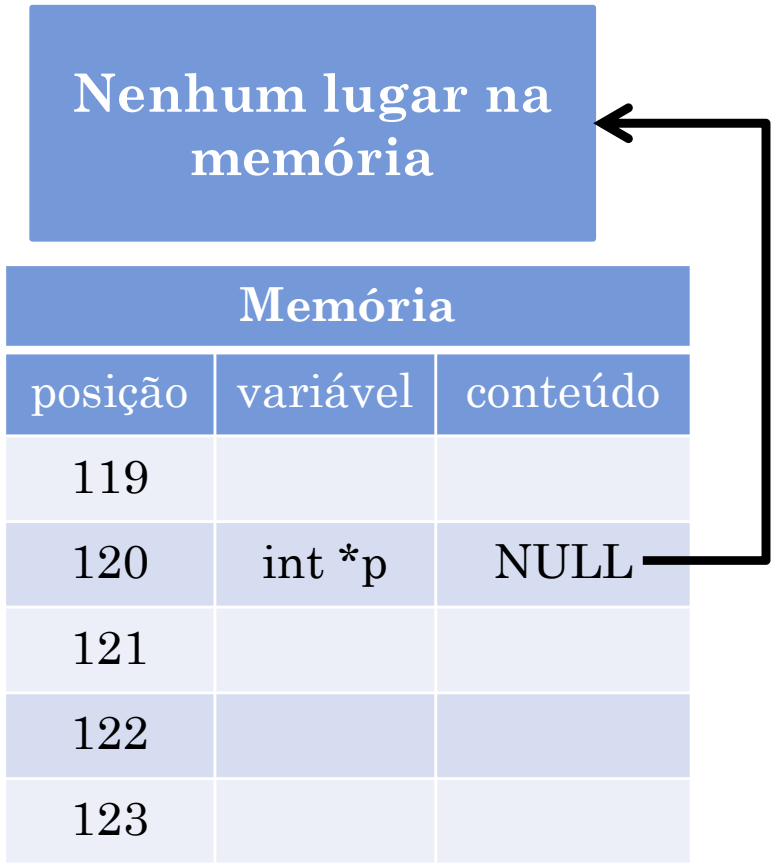
- Exemplo
  - `int *p;`

Memória		
posição	variável	conteúdo
119		
120	int *p	????
121		
122		
123		

INICIALIZAÇÃO

- Um ponteiro pode ter o valor especial NULL que é o endereço de nenhum lugar

- Exemplo
  - `int *p = NULL;`



INICIALIZAÇÃO

- O ponteiro armazena o endereço da variável para onde ele aponta
  - Para saber o endereço de memória de uma variável do nosso programa, usamos o operador **&**
  - Ao armazenar o endereço, o ponteiro estará apontando para aquela variável

Memória		
posição	variável	conteúdo
119		
120	int *p	122
121		
122	int c	10
123		


```
int main(){
    //Declara uma variável int contendo o valor 10
    int c = 10;
    //Declara um ponteiro para int
    int *p;
    //Atribui ao ponteiro o endereço da variável int
    p = &c ;

    return 0;
}
```

UTILIZAÇÃO

- Para acessar o **valor** guardado dentro de uma posição na memória apontada por um ponteiro, basta usar o operador ***asterisco*** "*" na frente do nome do ponteiro

```
int main(){
    //Declara uma variável int contendo o valor 10
    int c = 10;
    //Declara um ponteiro para int
    int *p;
    //Atribui ao ponteiro o endereço da variável int
    p = &c;
    printf("Conteudo apontado por p: %d \n",*p);// 10
    //Atribui um novo valor à posição de memória apontada por p
    *p = 12;
    printf("Conteudo apontado por p: %d \n",*p);// 12
    printf("Conteudo de count: %d \n",c);// 12

    return 0;
}
```

UTILIZAÇÃO

- ***p** :conteúdo da posição de memória apontado por **p**
- **&c**: o endereço na memória onde está armazenada a variável **c**

```
int main(){
    //Declara uma variável int contendo o valor 10
    int c = 10;
    //Declara um ponteiro para int
    int *p;
    //Atribui ao ponteiro o endereço da variável int
    p = &c;
    printf("Conteudo apontado por p: %d \n",*p);// 10
    //Atribui um novo valor à posição de memória apontada por p
    *p = 12;
    printf("Conteudo apontado por p: %d \n",*p);// 12
    printf("Conteudo de count: %d \n",c);// 12

    return 0;
}
```

Operações com ponteiros

- Atribuição
  - p1 aponta para o mesmo lugar que p

```
int *p, *p1;
int c = 10;
p = &c;
p1 = p;//equivale a p1 = &c;
```

  - A seguir, a variável apontada por p1 recebe o conteúdo da variável apontada por p

```
int *p, *p1;
int c = 10, d = 20;
p = &c;
p1 = &d;

*p1 = *p;//equivale a d = c;
```

Ponteiros Genéricos

- Normalmente, um ponteiro aponta para um tipo específico de dado

  - Um ponteiro genérico é um ponteiro que pode apontar para qualquer tipo de dado.

- Declaração

```
void *nome_ponteiro;
```

Ponteiros Genéricos

- Exemplos

```
int main(){
    void *pp;
    int *p1, p2 = 10;
    p1 = &p2;
    //recebe o endereço de um inteiro
    pp = &p2;
    printf("Endereco em pp: %p \n",pp);
    //recebe o endereço de um ponteiro para inteiro
    pp = &p1;
    printf("Endereco em pp: %p \n",pp);
    //recebe o endereço guardado em p1 (endereço de p2)
    pp = p1;
    printf("Endereco em pp: %p \n",pp);

    return 0;
}
```

Ponteiros Genéricos

- Para acessar o **conteúdo** de um ponteiro genérico é preciso antes convertê-lo para o tipo de ponteiro com o qual se deseja trabalhar
  - Isso é feito via ***type cast***

```
int main(){
    void *pp;
    int p2 = 10;
    // ponteiro genérico recebe o endereço de um
    // inteiro
    pp = &p2;
    //enta acessar o conteúdo do ponteiro genérico
    printf("Conteudo: %d\n",*pp); //ERRO
    // converte o ponteiro genérico pp para (int *)
    // antes de acessar seu conteúdo.
    printf("Conteudo: %d\n",*(int*)pp); //CORRETO

    return 0;
}
```

Ponteiros e Arrays

- Ponteiros e arrays possuem uma ligação muito forte

  - Arrays são agrupamentos de dados do mesmo tipo na memória

  - Quando declaramos um array, informamos ao computador para reservar uma certa quantidade de memória a fim de armazenar os elementos do array de forma sequencial

  - Como resultado dessa operação, o computador nos devolve um ponteiro que aponta para o começo dessa sequência de bytes na memória

Ponteiros e Arrays

- O nome do array (sem índice) é apenas um ponteiro que aponta para o primeiro elemento do array

```
int vet[5] = {1,2,3,4,5};
int *p;

p = vet;
```

Memória		
posição	variável	conteúdo
119		
120		
121	int *p	123
122		
123	int vet[5]	1
127		2
131		3
135		4
139		5
143		

Ponteiros e Arrays

- Os colchetes **[ ]** substituem o uso conjunto de operações aritméticas e de acesso ao conteúdo (operador "*") no acesso ao conteúdo de uma posição de um array ou ponteiro.
  - O valor entre colchetes é o deslocamento a partir da posição inicial do array.
  - Nesse caso, **p[2]** equivale a ***(p+2)**.

```
int main (){
    int vet[5] = {1,2,3,4,5};
    int *p;
    p = vet;

    printf("Terceiro elemento: %d ou %d",p[2],*(p+2));

    return 0;
}
```

Ponteiros e Arrays

- Nesse exemplo

```
int vet[5] = {1,2,3,4,5};
int *p;

p = vet;
```

- Temos que:
  - ***p** é equivalente a **vet[0]**
  - **vet[índice]** é equivalente a ***(p+índice)**
  - **vet** é equivalente a **&vet[0]**
  - **&vet[índice]** é equivalente a **(vet + índice)**

Ponteiros e Arrays

Usando array

```c
int main(){
    int vet[5]= {1,2,3,4,5};
    int *p = vet;
    int i;
    for (i = 0;i < 5;i++)
        printf("%d\n",p[i]);

    return 0;
}
```

Usando ponteiro

```c
int main(){
    int vet[5]= {1,2,3,4,5};
    int *p = vet;
    int i;
    for (i = 0;i < 5;i++)
        printf("%d\n",*(p+i));

    return 0;
}
```

Ponteiro para ponteiro

- A linguagem C permite criar ponteiros com diferentes níveis de apontamento

  - É possível criar um ponteiro que aponte para outro ponteiro, criando assim vários níveis de apontamento

  - Assim, um ponteiro poderá apontar para outro ponteiro, que, por sua vez, aponta para outro ponteiro, que aponta para um terceiro ponteiro e assim por diante.

PONTEIRO PARA PONTEIRO

- Um ponteiro para um ponteiro é como se você anotasse o endereço de um papel que tem o endereço da casa do seu amigo.

- Podemos declarar um ponteiro para um ponteiro com a seguinte notação
  - **`tipo_ponteiro **nome_ponteiro;`**

- Acesso ao conteúdo
  - **`**nome_ponteiro`** é o conteúdo final da variável apontada;
  - **`*nome_ponteiro`** é o conteúdo do ponteiro intermediário.

Ponteiro para ponteiro

- Exercício: Diga o que é impresso em cada printf abaixo

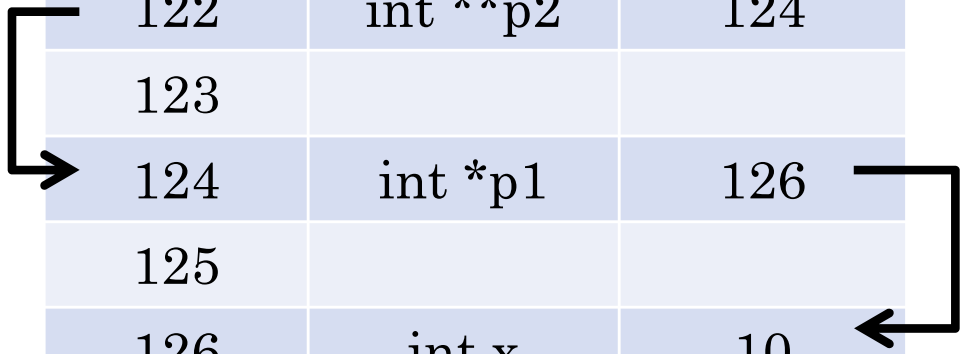
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int x = 10;
    int *p1 = &x;
    int **p2 = &p1;

    // Endereço de p2
    printf("Endereco de p2: %d \n",p2);
    // Conteudo do endereço (endereço de p1)
    printf("Conteudo em *p2: %d \n",*p2);
    // Conteudo do endereço do endereço, ou
    // seja, ovalor da variável apontada por p1
    printf("Conteudo em **p2: %d \n",**p2);

    return 0;
}
```

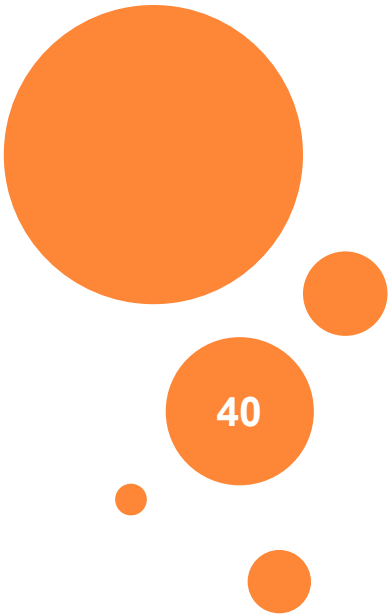
Memória		
posição	variável	conteúdo
119		
120		
121		
122	int **p2	124
123		
124	int *p1	126
125		
126	int x	10
127		



Material Complementar

- Vídeo Aulas
  - Aula 55: Ponteiros pt.1 – Conceito
  - Aula 56: Ponteiros pt.2 – Operações
  - Aula 57: Ponteiros pt.3 – Ponteiro Genério
  - Aula 58: Ponteiros pt.4 – Ponteiros e Arrays
  - Aula 59: Ponteiros pt.5 – Ponteiro para Ponteiro

  - https://programacaodescomplicada.wordpress.com/indice/linguagem-c/



LINGUAGEM C: PONTEIROS

Contém slides originais gentilmente disponibilizados pelo Prof. André R. Backes (UFU)