Princípios de Desenvolvimento de Algoritmos
MAC122

Prof. Dr. Paulo Miranda
IME-USP

Ponteiros

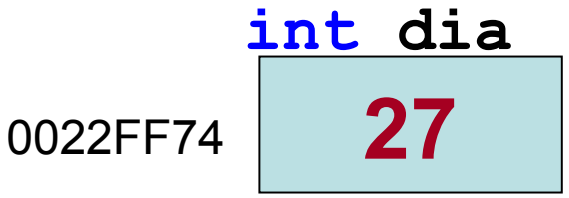
Ponteiros

- **Introdução:**
  - Uma variável é um espaço da memória principal reservado para armazenar dados.
  - Variáveis possuem:
    - **Nome:**
      - Identificador usado para acessar o conteúdo.
    - **Tipo:**
      - Determina a capacidade de armazenamento.

        Ex: int, char, float, ...
    - **Endereço:**
      - Posição na memória principal.

Ponteiros

- **Exemplo:**
  - **Nome:** `dia`
  - **Tipo:** `int`
  - **Endereço:** `0022FF74` `(hexadecimal)` **ou**
    `2293620` `(decimal)` **ou**
    `&dia` `(representação simbólica)`
  - **Conteúdo:** `27`



Ponteiros

- **Definição:**
  - Ponteiro é uma variável que armazena um endereço de memória, como por exemplo o endereço de uma outra variável.
- **Declaração (sintaxe):**

```
tipo *nome_do_ponteiro;
```

- **Exemplos:**

```
int *p;       /*declara ponteiro para um int.  */
char *tmp;    /*declara ponteiro para um char. */
float *pont; /*declara ponteiro para um float.*/
```

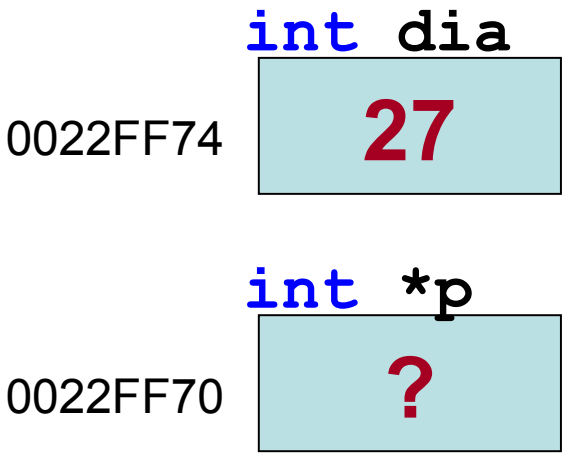
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
  int dia = 27;
  int *p;

  p = &dia;
  return 0;
}
```



- **As variáveis são declaradas.**
- **O ponteiro como toda variável também possui um endereço de memória (&p = 0022FF70).**

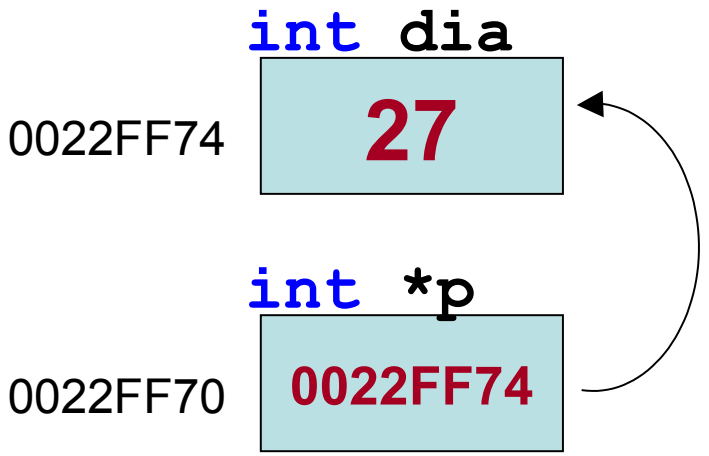
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   return 0;
}
```



- **O endereço de `dia` é atribuído para o ponteiro `p`.**
- **Dizemos que `p` aponta para a variável `dia` (graficamente representado por uma seta).**

Ponteiros

- **Por que usar ponteiros?**
  - Nos exemplos até agora, o acesso ao conteúdo das variáveis se dava através do nome delas.
  - Ponteiros nos fornecem um novo modo de acesso que explora o endereço das variáveis.
  - Para isso usamos o operador indireto (*), que nos permite ler e alterar o conteúdo das variáveis apontadas por um ponteiro.

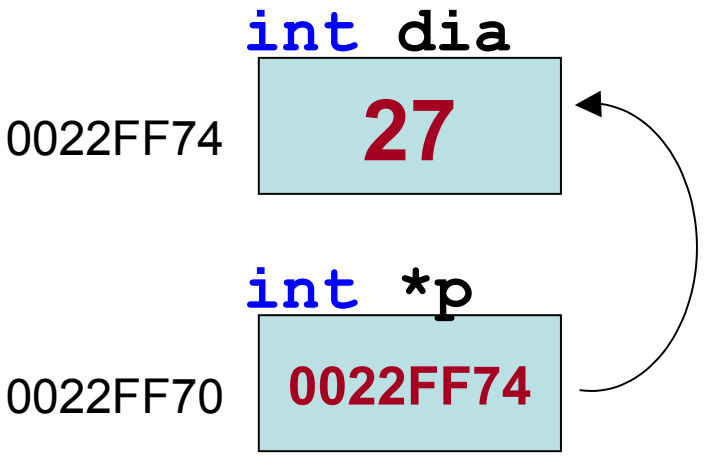
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   *p = 10;
   return 0;
}
```



- **O endereço de `dia` é <u>atribuído</u> para o ponteiro `p`.**
- **Dizemos que `p` <u>aponta</u> para a variável `dia` (graficamente representado por uma seta).**

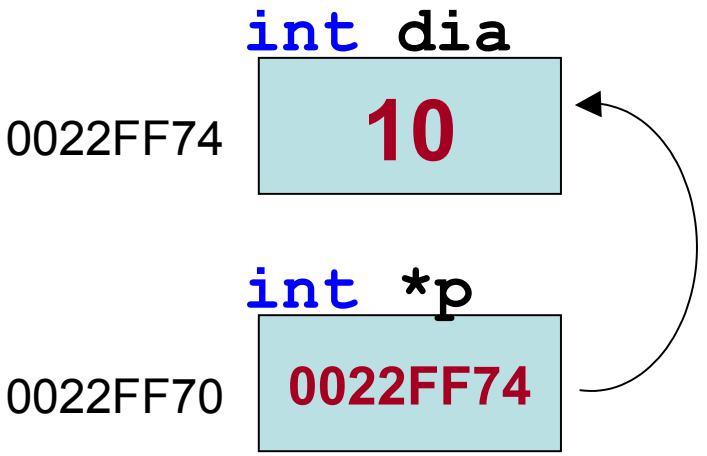
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   *p = 10;
   return 0;
}
```



- **O código `*p` é o conteúdo da variável apontada por `p`, ou seja o conteúdo de `dia`, que recebe o valor `10`.**

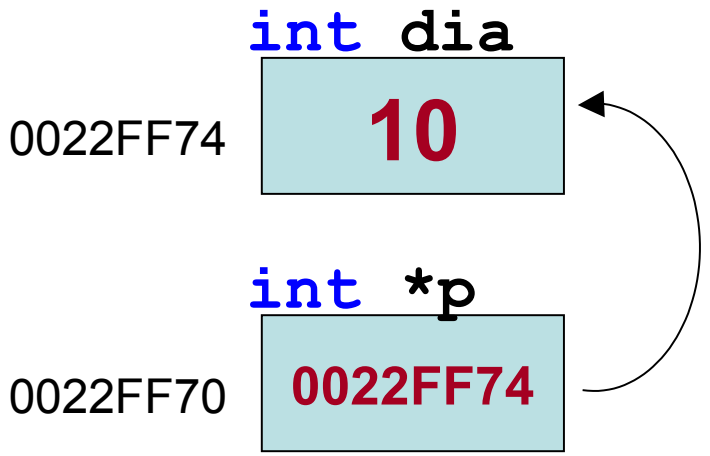
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   *p = 10;
   return 0;
}
```



- **A declaração `int *p;` indica que a variável `p` é um ponteiro para um inteiro e que `*p` é do tipo `int`.**

Ponteiros

- **Por que usar ponteiros?**
  - Variáveis simples e estruturas são passadas por valor para funções. Ou seja, é gerada uma cópia da variável e alterações na função não produzem qualquer efeito externo.
  - Com o uso de ponteiros é possível realizar a passagem dos valores por referência.

Ponteiros

- **Exemplo: Função que lê um valor inteiro.**

```c
#include <stdio.h>

void LeInteiro(int a){
  printf("Entre com a: ");
  scanf("%d",&a);
}

int main(){
  int a=0;
  LeInteiro(a);
  printf("a: %d\n",a);
  return 0;
}
```

- **O código acima irá imprimir 0 na saída padrão sempre. O problema é que o `scanf` altera apenas uma variável local da função que deixa de existir após a sua execução.**
- **A variável `a` da função principal permanece intacta.**

Ponteiros

- **Exemplo: Função que lê um valor inteiro.**

```c
#include <stdio.h>

void LeInteiro(int a){
  printf("Entre com a: ");
  scanf("%d",&a);
}

int main(){
  int a=0;
  LeInteiro(a);
  printf("a: %d\n",a);
  return 0;
}
```

```c
#include <stdio.h>
int LeInteiro(){
  int a;
  printf("Entre com a: ");
  scanf("%d",&a);
  return a;
}
int main(){
  int a=0;
  a = LeInteiro();
  printf("a: %d\n",a);
  return 0;
}
```

- **Uma possível solução é apresentada a direita. Porém funções só podem retornar um único valor e em casos onde é necessário alterar mais de uma variável o código não se aplica.**

Ponteiros

- **Exemplo: Função que lê um valor inteiro.**

```c
#include <stdio.h>

void LeInteiro(int a){
  printf("Entre com a: ");
  scanf("%d",&a);
}

int main(){
  int a=0;
  LeInteiro(a);
  printf("a: %d\n",a);
  return 0;
}
```

```c
#include <stdio.h>
void LeInteiro(int *p){
  int a;
  printf("Entre com a: ");
  scanf("%d",&a);
  *p = a;
}
int main(){
  int a=0;
  LeInteiro(&a);
  printf("a: %d\n",a);
  return 0;
}
```

- **Uma solução usando ponteiros é mostrada. O ponteiro `p` é inicializado com o endereço da variável `a` da função principal. Logo, `*p` é o próprio conteúdo de `a` da função principal.**
- **Dizemos que `a` foi passada por referência.**

Ponteiros

- **Exemplo Real: Função que troca os valores de duas variáveis.**

```c
#include <stdio.h>
void troca(int *a, int *b){
  int tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}
int main(){
  int a=25,b=12;
  troca(&a, &b);
  printf("a: %d, b: %d\n",a,b);
  return 0;
}
```

- **Vimos que operações de troca são importantes em algoritmos como o Bubble sort. A função acima recebe o endereço de duas variáveis e usando uma variável temporária procede com a troca dos valores. A saída do programa será "a: 12, b: 25".**