Linguagem C: Ponteiros

Joice Otsuka

*Baseado no livro: Linguagem C Completa e Descomplicada, de André Backes

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 serve para armazenar endereços de memória (são valores inteiros sem sinal).
    - 2 bytes em sistemas de 16 bits, 4 bytes em sistemas de 32 bits e 8 bytes em sistemas de 64 bits

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.

```
//Declaração
tipo_ponteiro *nome_ponteiro;
ou
tipo_ponteiro* nome_ponteiro;
```

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

ou

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

Declaração

- Exemplos de declaração de variáveis e ponteiros

```c
int main(){
    //Declara um ponteiro para int
    int *p;
    //Declara um ponteiro para float
    float *x;
    //Declara um ponteiro para char
    char *y;
    //Declara um ponteiro para struct ponto
    struct ponto *p;
    //Declara uma variável do tipo int e um ponteiro para int
    int soma, *p2,;

    return 0;
}
```

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 (**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;`**

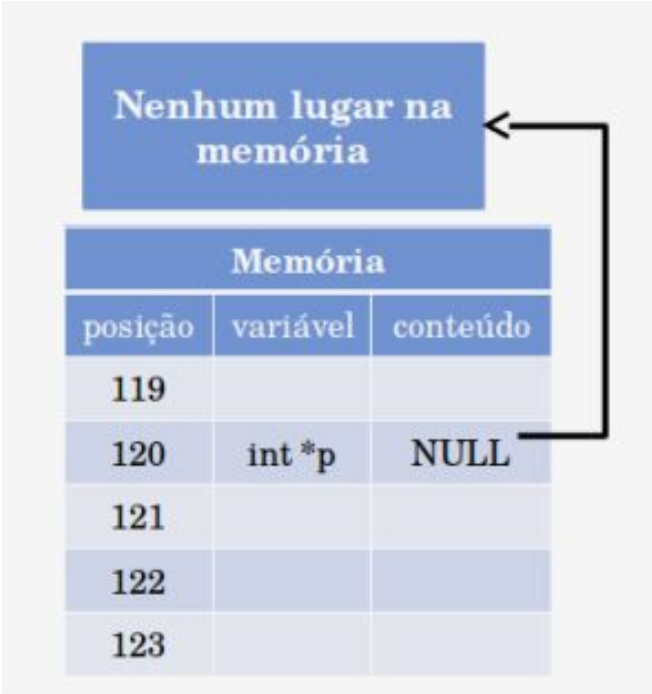
Memória		
posição	variável	conteúdo
119		
120	int *p	????
121		
122		
123		

Inicialização

- Um ponteiro pode ter o valor especial NULL que é o endereço de **nenhum lugar.**
- Exemplo
  - **int *p = NULL; // *p = 0;**

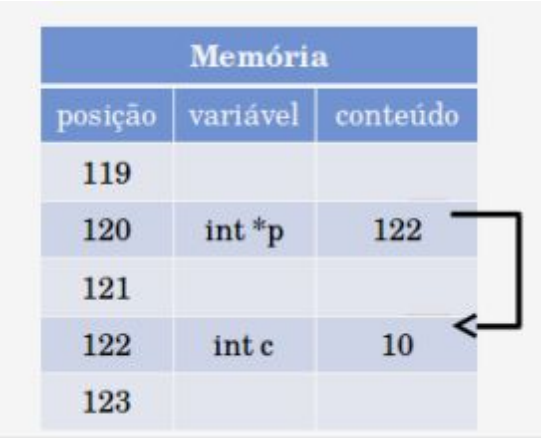
NULL é uma constante

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



Inicialização

- **Os ponteiros devem ser inicializados antes de serem usados.**
- Devemos apontar um ponteiro para um lugar conhecido
  - Podemos inicializá-lo com o endereço de uma variável que já exista no programa.



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

```
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

- Sendo que um ponteiro armazena um endereço de memória, como saber o valor guardado nesse endereço?
  - basta usar o operador ***asterisco*** “*” na frente do nome do ponteiro

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;
}
```

```c
int main()
{
    int i;
    float f;
    void *endereco; //ponteiro sem tipo

    endereco = &i;
    printf("A variável i inicia em %p\n",endereco);

    endereco = &f;
    printf("A variável f inicia em %p\n",endereco);

    endereco = &endereco;
    printf("A variável endereço inicia em %p\n",endereco);

    endereco = &main; // recebe o endereço da função main
    printf("A função main inicia em %p\n",endereco);

    endereco = &printf;
    printf("A função printf inicia no endereço %p\n",endereco);

    endereco = NULL;
    printf("Ponteiro nulo %p\n",endereco);

    return 0;
}
```

Imprimindo alguns endereços

```c
#include <stdio.h>

int main()
{
    int int1,int2;
    int *p;

    int1 = 28;
    int2 = 12;
    p = &int1;
    printf("int1:%d p:%p *p:%d\n", int1,p,*p);

    p = &int2;
    printf("int2:%d p:%p *p:%d\n", int2,p,*p);

    return 0;
}
```

Acesso ao conteúdo de um endereço

```c
#include <stdio.h>

int main()
{
    int int1;
    int *p;

    int1 = 28;
    p = &int1;
    printf("int1:%d p:%p *p:%d\n", int1,p,*p);

    *p=30;
    printf("int1:%d p:%p *p:%d\n", int1,p,*p);

    return 0;
}
```

Usos do * em C

(a) Multiplicação: **x = y*z;**

(b) Multiplicação-atribuição: **x *= y;** Equivalente a: **x = x*y;**

(c) Comentário: **/* seu comentário */**

(d) Declaração de ponteiro: **int *p;** or **int* p;** Leia: p é um ponteiro para um inteiro.

(e) Ponteiros compostos: **int **p;** or **int** p;** Leia: p é um ponteiro para um ponteiro para um inteiro.

(Também int ***p; e assim por diante…)

(f) De-referência: **x = *p;** Leia: Atribua a x, o valor armazenado na posição de memória apontada por p.

Fonte: https://www.hermetic.ch/cfunlib/ast_amp2.htm

Exercícios

4.6-6 Escreva um programa em **C** para ler duas variáveis do tipo **int** e dobrar o valor da menor delas (em caso de igualdade, dobrar o valor da primeira variável). As comparações e modificações devem ser feitas exclusivamente usando ponteiros. Os valores devem ser apresentados no final.

```c
#include <stdio.h>

int main()
{
    int int1,int2;
    int *p1, *p2;

    scanf("%d",&int1);
    scanf("%d",&int2);
    p1 = &int1;
    p2 = &int2;
    printf("int1:%d int2:%d\n", *p1,*p2);
    if(*p1 <= *p2)
        *p1*=2;
    else
        *p2*=2;

    printf("int1:%d int2:%d\n", *p1,*p2);

    return 0;
}
```

Exercícios

4.6-8 Observe atentamente o Programa **4.6-4** e tente prever os valores que serão escritos caso a leitura seja feita para o valor **10**. Depois implemente e execute o código. Sua previsão foi correta? Se não, entenda o que aconteceu.

Programa 4.6-4 Uso de múltiplos ponteiros (dois, neste caso).

```
/*
   Input: um valor inteiro
   Output: tres valores inteiros
*/
#include <stdio.h>

int main(){
    int i, *p1, *p2;

    printf("i = ");
    scanf("%d", &i);

    p1 = &i;
    p2 = p1;
    *p2 = *p1 * 2;

    printf("%d, %d e %d\n", i, *p1, *p2);

    return 0;
}
```


Arquivo: material/variosponteiros.c.

Exercícios

Faça as duas listas de exercícios sobre endereços de memória e ponteiros no AVA

Utilização

- De modo geral, um ponteiro só pode receber o endereço de memória de uma variável do mesmo tipo do ponteiro
  - Isso ocorre porque variáveis de diferentes tipos ocupam espaços de memória de tamanhos diferentes
  - Na verdade, nós podemos atribuir a um ponteiro de inteiro (**int ***) o endereço de uma variável do tipo **float**. No entanto, o compilador assume que qualquer endereço que esse ponteiro armazene obrigatoriamente apontará para uma variável do tipo **int**
  - Isso gera problemas na interpretação dos valores

Utilização

```
int main(){
    int *p, *p1, x = 10;
    float y = 20.0;
    p = &x;
    printf("Conteudo apontado por p: %d \n",*p);

    p1 = p;
    printf("Conteudo apontado por p1: %d \n",*p1);

    p = &y;
    printf("Conteudo apontado por p: %d \n",*p);
    printf("Conteudo apontado por p: %f \n",*p);
    printf("Conteudo apontado por p: %f \n",*((float*)p));

    return 0;
}
```

Warnings do compilador

Conversão explícita (type cast) para o tipo float*

```
Conteudo apontado por p: 10
Conteudo apontado por p1: 10
Conteudo apontado por p: 1101004800
Conteudo apontado por p: 0.000000
Conteudo apontado por p: 20.000000
```

Operações com ponteiros

- Atribuição
  - Ex1: p1 aponta para o mesmo lugar que p;

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

  - Ex2: a variável apontada por p1 recebe o mesmo conteúdo da variável apontada por p;

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

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

Operações com ponteiros

- Apenas duas operações aritméticas podem ser utilizadas com o endereço armazenado pelo ponteiro: adição e subtração
- podemos apenas somar e subtrair valores INTEIROS
  - p++;
    - soma +1 no endereço armazenado no ponteiro.
  - p--;
    - subtrai 1 no endereço armazenado no ponteiro.
  - p = p+15; (p += 15)
    - soma +15 no endereço armazenado no ponteiro.

Operações com ponteiros

- As operações de adição e subtração no endereço dependem do tipo de dado que o ponteiro aponta.
  - Considere um ponteiro para inteiro, int *
  - O tipo int ocupa um espaço de 4 bytes na memória
  - Assim, nas operações de adição e subtração são adicionados/subtraídos 4 bytes por incremento/decremento, pois esse é o tamanho de um inteiro na memória

Memória		
posição	variável	conteúdo
119		
120	int a	10
121		
122		
123		
124	int b	20
125		
126		
127		
128	char c	‘k’
129	char d	‘s’
130		

Operações com ponteiros

- Operações ilegais com ponteiros
  - Dividir ou multiplicar ponteiros;
  - Somar o endereço de dois ponteiros;
  - Não se pode adicionar ou subtrair valores dos tipos **float** ou **double** de ponteiros.

Operações com ponteiros

- Já sobre seu conteúdo apontado, valem todas as operações
  - (*p)++;
    - incrementa o conteúdo da variável apontada pelo ponteiro p;
  - *p = (*p) * 10;
    - multiplica o conteúdo da variável apontada pelo ponteiro p por 10;

```
int *p;
int c = 10;

p = &c;

(*p)++;
*p = (*p) * 10;
```

(a) Qual valor armazenado no endereço apontado por p?

(b) Qual valor armazenado na variável c?

Operações com ponteiros

- Operações relacionais
  - **==** e **!=** para saber se dois ponteiros são iguais ou diferentes.
  - **>**, **<**, **>=** e **<=** para saber qual ponteiro aponta para uma posição mais alta na memória.

```c
int main(){
    int *p, *p1, x, y;
    p = &x;
    p1 = &y;
    if(p == p1)
        printf("Ponteiros iguais\n");
    else
        printf("Ponteiros diferentes\n");

    return 0;
}
```

Ponteiros e Arrays

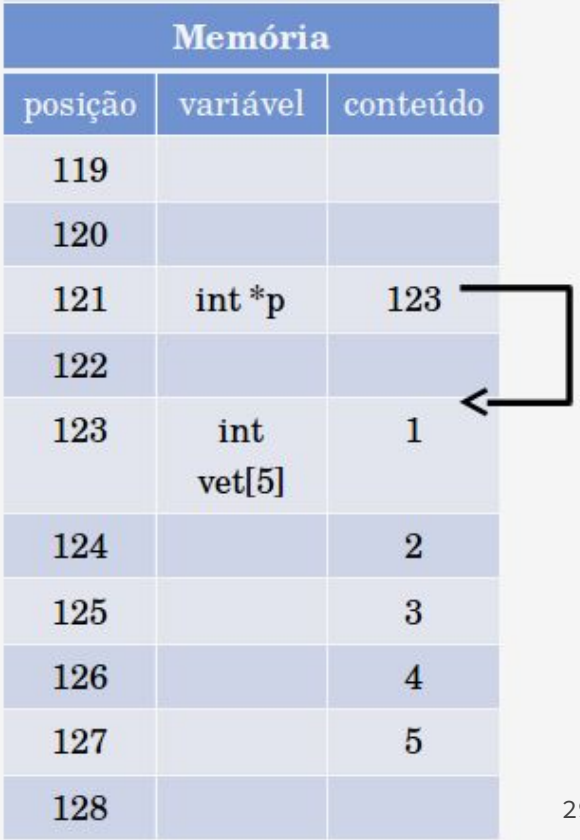
- Ponteiros e arrays possuem uma ligação muito forte.
  - Arrays são agrupamentos de dados do mesmo tipo na memória.
  - Quando declaramos um array, informamos ao computador para reservar uma certa quantidade de memória a fim de armazenar os elementos do array de forma sequencial.
  - Como resultado 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;
```



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)**.

```c
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

```
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

```
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;
}
```

Exercícios

4.6-13 Qual a saída esperada para o **Programa 4.6-6**?. Tente fazer sua previsão e depois implemente e verifique se acertou.

Programa 4.6-6 Exemplos de uso de ponteiros com vetores.

```
/*
   Output: dados de um vetor manipulado com
     ponteiros
*/
#include <stdio.h>

int main(){
   int vet[] = {1, 2, 3, 4, 5, 6, 7, 8};

   /* Antes */
   printf("original:   [ ");
   for(int i = 0; i < 8; i++)
      printf("%2d ", vet[i]);
   printf("]\n");

   /* Modificacoes */
   vet[4] = 40;

   int *pont = vet;
   *pont = 99;
   *(pont + 1) = 33;
   pont[6] = 10 * pont[7];

   /* Depois */
   printf("modificado: [ ");
   for(int i = 0; i < 8; i++)
      printf("%2d ", vet[i]);
   printf("]\n");
}
```

Arquivo: material/ponteirovetor.c.

Exercícios

4.6-14 Suponha que um vetor **vet** de **floats** não possui nenhum elemento igual a zero. Considerando-se que todas as instruções abaixo manipulem posições válidas desse vetor, determine quantas se tornam nulas depois da execução. Assuma que **p** seja um ponteiro para **float**.

```
p = &vet[5];
*p = 0;
p[1] = 0;
p[-1] = 0;
p++;
p[0] = 0;
*(p + 1) = 0;
```

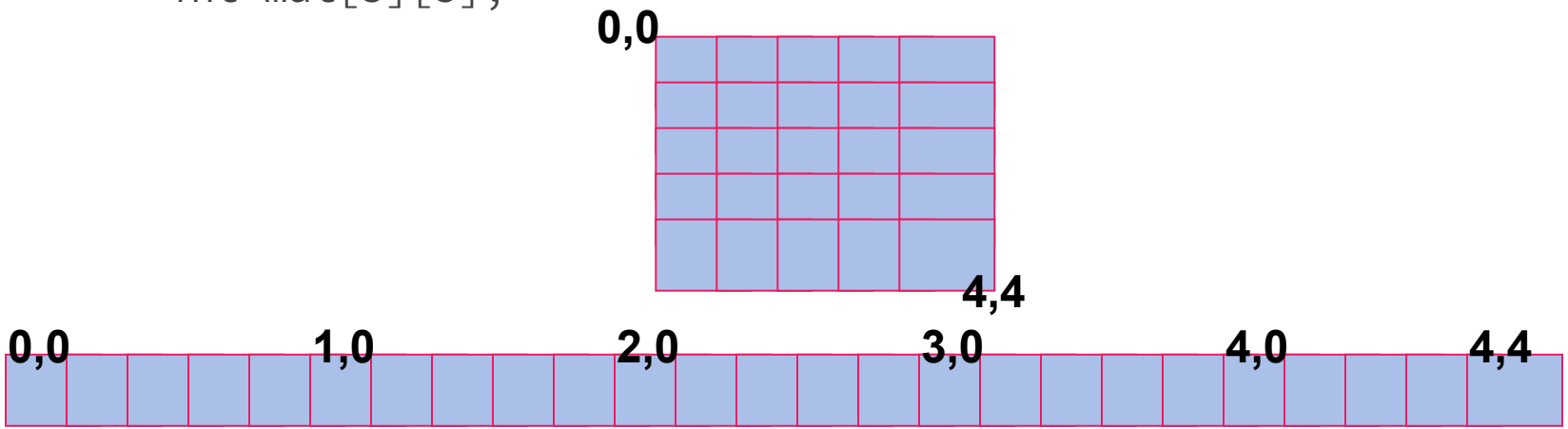
Exercícios

4.6-15 Assuma que **vet** seja um vetor de valores numéricos que contenha pelo menos um valor negativo. O que o contador **cont** na sequência de comandos seguinte determina, sendo **p** um ponteiro para o tipo base do vetor?

```
cont = 0;
p = vet;
while (*p >= 0) {
    cont++;
    p++;
}
```

Ponteiros e Arrays

- Arrays Multidimensionais
  - Apesar de terem mais de uma dimensão, na memória os dados são armazenados linearmente.
  - Ex.:
  - int mat[5][5];



Ponteiros e Arrays

- Pode-se então percorrer as várias dimensões do array como se existisse apenas uma dimensão. As dimensões mais à direita mudam mais rápido

Array

```c
int main() {
    int i, j;
    int mat[2][2] = {{1, 2}, {3, 4}};
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            printf("%d ", mat[i][j]);
        }
    }
    return 0;
}
```

Ponteiro

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

Exercícios

4.6-16 Suponha que uma matriz **mat** de inteiros com dimensões **15 x 12** e que **i** e **j** sejam valores inteiros. Considere que **p** seja um ponteiro para inteiro e que aponte para o primeiro elemento da matriz (primeira linha, primeira coluna).

Quando se escreve ***(p + 12*i + j)**, a qual elemento da matriz se está referenciando? Considere tanto **i** quanto **j** no intervalo **[0,11]**.

Ponteiros e Arrays

- Pode-se então percorrer as várias dimensões do array como se existisse apenas uma dimensão. As dimensões mais à direita mudam mais rápido

Array

```c
int main() {
    int i, j;
    int mat[2][2] = {{1, 2}, {3, 4}};
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            printf("%d ", mat[i][j]);
        }
    }
    return 0;
}
```

Ponteiro (a)

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

Ponteiros e Arrays

Ponteiro (a)

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

Ponteiro (b)

```c
int main() {
    int mat[2][2] = {{1, 2}, {3, 4}};
    int i, j, *p = &mat[0][0];
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            printf("%d ", *(p + (2 * i) + j));
        }
    }
    return 0;
}
```

Ponteiro para struct

- Existem duas abordagens para acessar o conteúdo de um ponteiro para uma struct
- Abordagem 1
  - Devemos acessar o conteúdo do ponteiro para struct para somente depois acessar os seus campos e modificá-los.
- Abordagem 2
  - Podemos usar o ***operador seta “->”***
  - **ponteiro->nome_campo**

```
struct ponto {
    int x, y;
};

struct ponto q;
struct ponto *p;

p = &q;

(*p).x = 10;      (abordagem 1)
p->y = 20;        (abordagem 2)
```

Exercícios

4.6-4 Escreva um programa em **C**, que declare uma variável do tipo registro (**struct**). Use o operador **&** para mostrar o endereço tanto da variável como um todo quanto dos seus campos.

Exercícios

4.6-4 Escreva um programa em **C**, que declare uma variável do tipo registro (**struct**). Use o operador **&** para mostrar o endereço tanto da variável como um todo quanto dos seus campos.

```c
struct Registro {
    int i;
    float f;
    char s[16];
    char c1;
    char c2;
};


int main() {
    struct Registro reg;
    void *endereco;

    endereco = &reg;
    printf("variavel reg inicia em %p\n", endereco);

    endereco = &(reg.i);
    printf("variavel reg.i inicia em %p\n", endereco);

    endereco = &(reg.f);
    printf("variavel reg.f inicia em %p\n", endereco);

    endereco = reg.s;
    printf("variavel reg.s inicia em %p\n", endereco);

    endereco = &(reg.c1);
    printf("variavel reg.c1 inicia em %p\n", endereco);

    endereco = &(reg.c2);
    printf("variavel reg.c2 inicia em %p\n", endereco);

    return 0;
}
```

Exercícios

4.6-9 Considere a seguinte declaração para um registro.

```
struct reg {
    int i;
    float f;
};
```

Suponha as seguintes declarações de variáveis

```
struct reg r;
struct reg *pont_r;
```

Há dúvidas de que ambas as partes abaixo são equivalentes?

```
// parte 1: acesso direto */
r.i = 10;
r.f = 1.26;
 // parte 2: acesso por ponteiro */
pont_r = &r;
pont_r->i = 10;
pont_r->f = 1.26;
```

Exercícios

4.6-10 Considere a seguinte declaração para um registro.

```
struct reg {
    char nome[101];
    char cpf[12];
};
```

Suponha as seguintes declarações de variáveis

```
struct reg r1, r2;
struct reg *pont_r;
```

O que o código seguinte faz? Ele é válido?

```
pont_r = &r1;
r2 = *pont_r;
```

Exercícios

4.6-12 Escreva um programa em **C** para ler dois registros contendo **RG** e **ano de nascimento** de duas pessoas, imprimindo os dados da mais velha (em caso de mesma idade, apresentar os dados da qual foi lida primeira).
Os dados de cada indivíduo devem ser mantidos em registros e as comparações e apresentações devem usar exclusivamente ponteiros. Use inteiros para **RG** e **ano de nascimento**.

Exercícios

Verifique o que ocorre quando o CNPJ da empresa é alterado. A alteração é refletida no registro do funcionário?

O que pode ser feito para refletir no registro f ?

```c
#include "stdio.h"
#include "string.h"

typedef struct {
    char CNPJ[15];
} Empresa;

typedef struct {
    char nome[20];
    Empresa empresa;
} Funcionario;

int main() {

    Empresa e;
    strcpy(e.CNPJ, "12345678901234");

    Funcionario f;
    strcpy(f.nome, "Fulano");
    f.empresa = e;

    printf("%s %s\n", f.nome, f.empresa.CNPJ);
    strcpy(e.CNPJ, "43210987654321");
    printf("%s %s\n", f.nome, f.empresa.CNPJ);

    return 0;
}
```

```c
#include "stdio.h"
#include "string.h"

typedef struct {
    char CNPJ[15];
} Empresa;

typedef struct {
    char nome[20];
    Empresa* empresa;
} Funcionario;

int main() {

    Empresa e;
    strcpy(e.CNPJ, "12345678901234");

    Funcionario f;
    strcpy(f.nome, "Fulano");
    f.empresa = &e;

    printf("%s %s\n", f.nome, f.empresa->CNPJ); // (*f.empresa).CNPJ
    strcpy(e.CNPJ, "43210987654321");
    printf("%s %s\n", f.nome, f.empresa->CNPJ);

    return 0;
}
```