PROGRAMAÇÃO DE COMPUTADORES V - TCC- 00.323

# Modulo 11:Endereços e ponteiros

Aura - Erick
aconci@ic.uff.br, erickr@id.uff.br

# Roteiro

- Endereços
- operador **&**
- ponteiros
- operador ***** **=** *valor* do objeto apontado (***p** )
- o valor especial **NULL** .
- tipos de ponteiros
- Usos
- Trabalho 7 !!!

# Motivação

- Os conceitos de **endereço** e **ponteiro** são fundamentais em qualquer linguagem de programação, embora fiquem ocultos em algumas linguagens.

- Em C, esses conceitos são explícitos.

- O conceito de ponteiro não é tão usual; é preciso fazer algum esforço para dominá-lo.

# Endereços

A memória RAM de qualquer computador é uma sequência de bytes.

Cada byte armazena de 1 a 256 possíveis valores.

Os bytes são numerados **sequencialmente na memória.**

**O número de um byte é o seu *endereço* (*address*).**

## Cada **tipo** na memória do computador ocupa um certo número de **bytes consecutivos**.

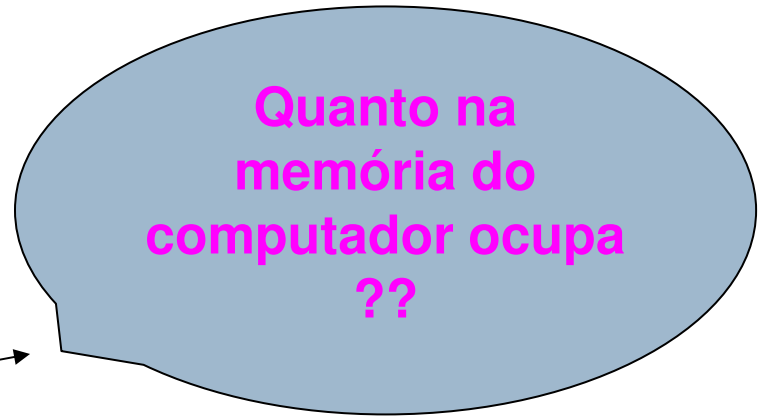
- Um **char** ocupa 1 byte.
- Um **int** ocupa 4 bytes em **alguns** computadores e 8 em **outros** ( o número exato é dado pela expressão **sizeof (int)** ).
- Um **double** ocupa **usualmente** 8 bytes ( o número exato é dado pela expressão **sizeof** (**double**) ).

Cada objeto na memória tem um *endereço*.

Na maioria dos computadores, o **endereço** de um objeto é o **endereço do seu primeiro byte**.

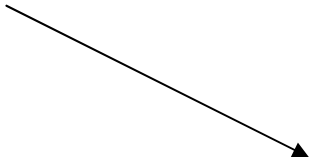
Por exemplo, depois das declarações seguidas:

```
char c;
int i;

struct
{
   int x, y;
 }
ponto;

int v[4];
```



# Os endereços das variáveis poderiam ser

| variáveis | endereço |
|---|---|
| **c** | **89421** |
| **i** | **89422** |
| **ponto** | **89426** |
| **v[0]** | **89434** |
| **v[1]** | **89438** |
| **v[2]** | **89442** |



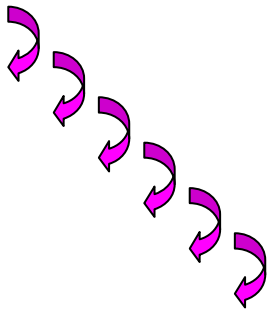
Esses os endereços são fictícios

Mas **porque a distância entre os números os endereços das variáveis é diferente**?

O que a **distância significa**?

Distancia entre os endereços das variáveis
corresponde ao espaço ocupado para armazena-la:

| | |
|---|---|
| c | 89421 |
| i | 89422 |
| ponto | 89426 |
| v[0] | 89434 |
| v[1] | 89438 |
| v[2] | 89442 |



O endereço de uma variável é dado pelo operador **&**.

(Não confunda esse uso de "**&**" com o operador lógico ***and*** ,
que em C se escreve "**&&**".)

Se **i** é uma variável então **&i** é o seu endereço.

No exemplo acima, **&i** vale **89422** e **&v[3]** vale **89446**

(Mas **porque** ??? )

# Endereços (relembrando....)

A memória RAM de qualquer computador é uma sequência de bytes.
Cada byte armazena de 1 a 256 possíveis valores.
Os bytes são numerados sequencialmente.
O número de um byte é o seu *endereço* (= *address*).

No exemplo anterior **cada endereço na memória é distante do endereço da posição na memória seguinte** de modo a que devem caber neles os **tipos a ser depositados**, isto é os objetos

**int i** e **v[3]** ;

A gente já usava **esse operador**

Exemplo:

O segundo argumento da função de biblioteca
**scanf** abaixo
é o endereço da posição na memória onde devem ser
depositados os objetos lidos do dispositivo
padrão de entrada:

```
int i;
scanf ( "%d", &i ) ;
```

# Ponteiros

Um *ponteiro* (= apontador = *pointer*) é um **tipo especial de variável que armazena endereços**.

Um ponteiro pode ter o valor especial
**NULL**
que **não é endereço de lugar algum**.

A **constante NULL** está definida na interface
**Stdlib.h**
stdlib.h

e seu **valor é 0** , na maioria dos computadores.

## "p aponta para i"

Se **um ponteiro p** armazena **o endereço de uma variável i**, podemos dizer "**p aponta para i**"

ou

"**p é o endereço de i**".

Em termos um pouco mais abstratos, diz-se que

**p é uma *referência* à variável i** .

Se um ponteiro **p** tem valor diferente de **NULL** então

***p** é **o *valor* do objeto apontado** por **p** .

Não confunda esse uso de "*" com o operador de multiplicação!

Como **i** é uma variável e **p** é **&i** então dizer " ***p** "
é o mesmo que dizer " **i** ".

Como ? ? ? ? ? ?  Pode deixar que repetimos.....

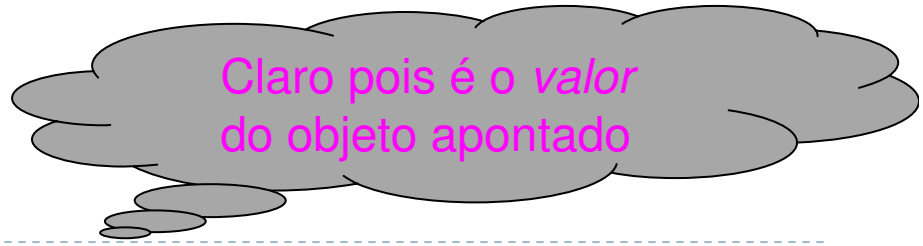
Se um ponteiro p armazena o endereço de uma variável i, podemos dizer "p aponta para i" ou "p é o endereço de i".

Diz-se que p é uma *referência* à variável i.

Se um ponteiro p então ***p**
é o *valor* do objeto apontado por **p**.

Por exemplo, se **i** é uma variável e **p** é o *valor* do objeto apontado , que é &i

então dizer " ***p** " é o mesmo que dizer " **i** ".



## “p aponta para i”

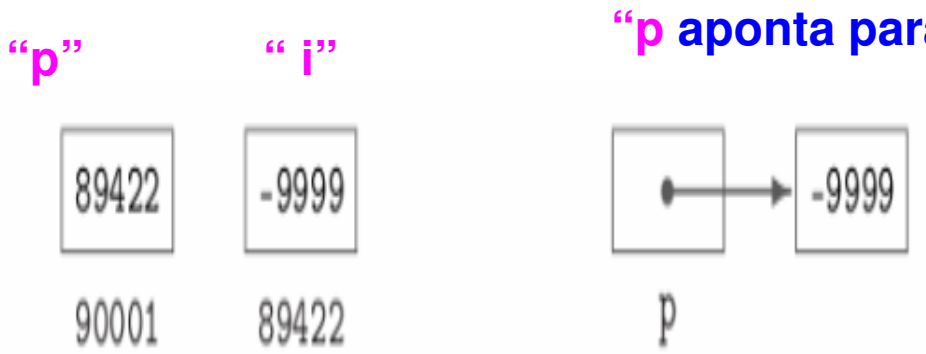



Figura esquerda: um ponteiro p, armazenado no endereço 90001, contém o endereço de um inteiro. Figura direita: representação esquemática da situação.

## Tipos de ponteiros:

ponteiros para caracteres,
ponteiros para inteiros,
ponteiros para ponteiros para inteiros,
ponteiros para  registros (endereços) etc.

O computador precisa saber de que tipo de ponteiro você está falando.
Para declarar um ponteiro p para um inteiro, diga **int *p ;**

Para declarar um ponteiro **p** para um **registro cel**, diga :
**struct cel *p;**

Um ponteiro **r** para um ponteiro que apontará para  um inteiro é declarado assim: **int **r;**

# Exemplos

Suponha que **a**, **b** e **c** são variáveis inteiras.
Eis um jeito "bobo" de fazer "**c = a+b**":

```
int *p ;        /* p é um ponteiro para um inteiro */

int *q ;        /* p é um ponteiro para um inteiro */

p = &a ;        /* o valor de p é o endereço de a , p aponta para a */

q = &b ;        /* o valor de q é o endereço de b , q aponta para b */

c = *p  +  *q;
```

# Exemplo

Suponha que **a**, **b** e **c** são variáveis inteiras.
Outro exemplo bobo, que faz o mesmo:

```
int *p ;        /* p  é um ponteiro para um inteiro */

int **r ;       /* r  é um ponteiro para um ponteiro de um inteiro */

p = &a ;        /* p  aponta para a */

r  = &p ;       /* r  aponta para p e *r aponta para a */

c = **r + b ;
```

# Aplicação

Suponha que precisamos de uma função que troque os valores de duas variáveis inteiras, digamos i e j.
É claro que a função

```
void troca (int i, int j) /* errado! */
{
   int temp;
   temp = i; i = j; j = temp;
}
```

# Aplicação cont.

Não produz o efeito desejado, pois recebe apenas os valores das variáveis e não as variáveis propriamente ditas.

A função **recebe "cópias" das variáveis** e troca os valores dessas cópias, enquanto as variáveis "originais" permanecem inalteradas.

Para obter o efeito desejado, é preciso passar à função os *endereços* das variáveis:

```
void troca (int *p, int *q)
{
   int temp;
   temp = *p; *p = *q; *q = temp;
}
```

Para aplicar a função às variáveis i e j basta dizer

```
troca (&i, &j);
```

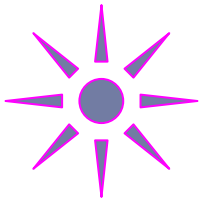
ou ainda

```
int *p, *q;

p = &i;
q = &j;

troca (p, q);
```

# Referencias

- https://pt.wikibooks.org/wiki/Programar_em_C
- https://pt.wikibooks.org/wiki/Programar_em_C/Fun%C3%A7%C3%B5es
- https://pt.wikipedia.org/wiki/Sistema_octal

- Veja mais na internet , e leia mais por ai ! ! !
- Troquem entre voces indicações de videos legais!!!



Há muita coisa legal deste assunto : ponteiros na internet ,
e por ai!!!

Me mande o endereço do video mais legal ! ! !

http://www.ime.usp.br/~pf/algoritmos/

E o próximo trabalho !!!
Nada ???
Então ! Tá !

# **7º** Trabalho - Entrega: 26 / 02 /2016

Escreva um programa que gere **algumas funções** para um **sistema** maior de um **controlador de estoques e promoções de um supermercado**, com os seguintes requisitos:

Mostre na tela as **instruções de funcionamento do programa**, como sendo seus Menus.

Todos os produtos devem ter **um código que deve ser um número inteiro**, mas representado por uma **enumeração** ou (enumerações) , como descrito no item 1, a seguir. Assim, para o usuário do programa (gerente, consumidor ou caixa) não aparece o código, mas o **nome do produto (**o que faz muito mais sentido e é mais fácil de guardar e entender) .

Cada **produto do supermercado** deve estar associado a pelo menos uma **estrutura** com pelo menos 12 elementos que serão descritos no item 2, abaixo. Essa deve estar em um **array com tantos** elementos quanto os itens **vendidos** no supermercado ou que estejam em seu **estoque**.

Todos os **campos da estrutura sempre estarão preenchidos,** alguns fornecidos pelo **controle de estoque** e outros **calculados,** como indicado nos **itens 3 a 5** abaixo.

# 1- **Os códigos dos produtos devem ter as seguintes características**

**Frutas e Legumes** iniciam por 10000 e podem ir até 20000.

**Maçâs devem ter no terceiro digito 1**, e os 2 últimos são os subtipos das maçãs, por exemplo:

**10110– maça gala nacional**

**10112 – maça gala chilena**

**10113 – maça gala argentina**

**10120 – maça fuji nacional**

**10130 – maça red nacional**

**10140 – maça verde nacional**

O mesmo ocorre para toda as demais **frutas e legumes**, cada um com uma campo para o tipo, e os 2 últimos para os **subtipos,** sendo o último para o pais de origem.

Por exemplo **Bananas, devem ter no terceiro digito 2** : **10210 - banana d´água**

**10220 – banana prata**

**10230 – banana da terra**

**10240 – banana ouro**

**10250 – banana maçã**

**Biscoitos e outros produtos de padaria** iniciam por 30000 e vão até 40000 (para esse você deve **inventar algum exemplo e incluir em seu programa**).

O mesmo ocorre para os demais possíveis produtos, sendo as seções definidas por x0000.

**Invente mais um exemplo diferente de algum outro produto** (desodorante, shampoo, detergentes, arroz, feijão, etc.... o que você achar interessante e inclua em seu programa).

## 2- Cada **produto do supermercado** deve estar associado a uma estrutura com os seguintes elementos:

- código do produto – inteiro (definido nas enumerações já comentadas)
- **quantidade** do produto **comprada** inicialmente – inteiro
- **quantidade** do produto **atual** – inteiro
- ano da **compra** – inteiro
- mês da **compra** – inteiro
- dia da **compra** – inteiro
- ano da **validade** – inteiro
- mês da **validade** – inteiro
- dia da **validade** – inteiro
- preço de **compra** – real
- preço de **venda** – real
- **Forma** de compra - inteiro

O último campo, acima, "**Forma de compra**" é um int. que vai caracterizar se o produto é comprado , exemplo: por Kilos, unidades, Litros, etc. (Possivelmente deve ser uma enumeração, mas não é tratado neste trabalho)

Faça a declaração do protótipo desta estrutura e a inclua em seu programa.

Você pode incluir outros campos que achar interessante.

Pode deve criar um ***array*** (vetor) formado por essa estrutura, com tantos elementos quanto os itens vendidos no supermercado e que estejam em seu estoque.

# estrutura “**produtos_supermercado**”

Por exemplo essa estrutura poderia ter os campos:

| Cod.pro. | Q_c | Q_a | D_c | D_v | M_c | M_v | A_c | A_v | P_c | P_v | Form |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 10110 | 10500 | 3473 | 20 | 26 | 2 | 2 | 2016 | 2016 | 1.10 | 7,50 | kilo |
| 10111 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | kilo |
| 10112 | 5000 | 2100 | 18 | 24 | 2 | 2 | 2016 | 2016 | 1.33 | 10.30 | kilo |
| 10113 | 4000 | 3500 | 17 | 23 | 2 | 2 | 2016 | 2016 | 1.23 | 12.40 | kilo |

Neste , por exemplo, **Form** pode ter os valores : kilo =1, litro =2, unidades = 3, etc..

No exemplo acima, isso quer dizer que as quantidades (Q_) estão associadas ao kilo, (por exemplo foram compradas 10500 kilos de Maças do tipo 10110) assim como os preços, (P_) que serão por kilo (custa 7,50 o kilo destas) .

# 3- Quando se calcula a **validade**

Alguns campos da estrutura de produtos são calculados, automaticamente (a partir dos dados vindo dos caixas e estoque e calculados pelo seu programa). Outros fornecidos pelo funcionário responsável pelas compras e estoque. De modos que, em um dado momento **cada produto (cuja quantidade não seja zero , 0) tem todos os campos sempre conhecidos** e podendo ser acessados.

Os **campos de validade** (dia, mês e ano) são fornecido com o produto, (quando esse tem alguma quantidade) a menos que esse seja do tipo "Frutas e Legumes" (ou outros perecíveis não tratados neste trabalho, como carnes e laticios em pedaçõs, etc.) quando **a validade é calculada como de 6 dias** a partir da data de compra.

# 4- Função a ser criada: **preço**

O **campo de preço** é atualizado automaticamente (quando o produto tem alguma quantidade) como sendo **1,6 x preço de compra**, se o produto estiver a 1 mês pelo menos da validade ou a **1,5 x preço de compra** contrário.

Mas ainda , sempre com o menor preço, pois se tiver a 1 dia da validade passa a ser **1,1 x preço de compra**, e se a **2 dias da validade é de 1,2 x preço de compra,** e 3 dias **da validade é de 1,3 x preço de compra e se a 1 semana da validade é de 1,4 x preço de compra.**

# 5 - Função aviso de **comprar** mais

O **controle de estoque** é baseado numa verificação simples que emite um **aviso** (quando o produto tem alguma quantidade) ao gerente quando o estoque de um produto for menor do que 10% do estoque inicial (ou seja 90% já tiver sido vendido).

Faça uma função que verifique isso e a inclua em seu programa.

Pode incluir outras funções se quiser, mas a nota no trabalho será baseada na execução destas

# Resumindo:

O seu programa deve incluir pelo menos as funções:

**Validade, Preço** e **Comprar** (mais para o estoque) acima definidas e aparecerem indicadas em como serem acionadas em sua Tela inicial.

Definir pelo menos 4 enumerações dos produtos mencionados (maca, banana, biscoito e outro – cada um com outro produto diferente) .

E ter a estrutura "**produtos_supermercado**" comentada.

**Entregue além do código em C o seu executável por e-mail para "erickr@id.uff.br". No subject da e-mail por "PROG V - TRAB 7"**

# Dúvidas?

- ????