Disciplina:

Programação Computacional

Prof. Fernando Rodrigues
e-m@il: fernandorodrigues@sobral.ufc.br

Aula 12: Programação em C

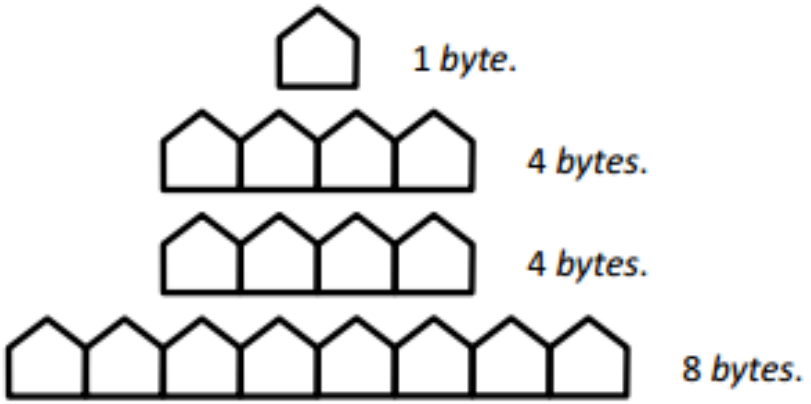
- Ponteiros
  - Ponteiros genéricos
  - Ponteiros para NULL
- Alocação Dinâmica de Memória

Identificadores e endereços

Na linguagem C, uma declaração faz a associação entre identificadores e endereços de memória.

O número de endereços (*bytes*) de memória depende do tipo utilizado na declaração.

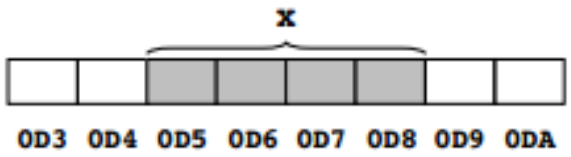
```
char c;
int i;
float x;
double y;
```



Identificadores e endereços

Considere a declaração de uma variável do tipo **`float`**, identificada por **`x`**:

```
float x;
```



Após a declaração, os 4 *bytes* que se iniciam no endereço **0D5** estão reservados e são acessíveis pelo programa através do identificador **x**.

```
printf("Endereço de x: %X", &x);  // irá exibir 0D5
```

Evolução

Linguagens de alto nível associam identificadores de variáveis a endereços de memória.

Ponteiros

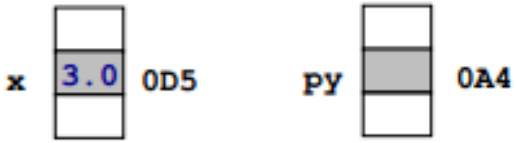
Outra forma de acesso à memória é através dos endereços dos *bytes*. Para isso, é preciso utilizar uma entidade denominada ponteiro.

Definição

Ponteiros são variáveis que armazenam apenas endereços de memória.

Para declarar uma variável do tipo ponteiro, utiliza-se o símbolo * entre o tipo e o identificador do mesmo:

```
float x = 3.0;    // x é uma variável do tipo float
float *py;        // py é ponteiro para variáveis do tipo float
```



Os operadores “*” e “&”

- Ao se trabalhar com ponteiros, duas tarefas básicas serão sempre executadas:
  - Acessar o endereço de memória de uma variável.
  - Acessar o conteúdo de um endereço de memória.
- Para realizar essas tarefas, vamos sempre utilizar apenas dois operadores: o operador “*” e o operador “&”.

Exemplo: operador “*” *versus* operador “&”	
“*”	Declara um ponteiro: **int *x;**
	Conteúdo para onde o ponteiro aponta: **int y = *x;**
“&”	Endereço onde uma variável está guardada na memória: **&y**

Aritmética de ponteiros

Se **pa** é um ponteiro para inteiros, que armazena o valor **DA3**, o comando:

```
pa = pa + 1;      // ou pa++;
```

causa uma mudança no valor de **pa**, que passa a apontar para o próximo endereço do tipo inteiro, ou seja, **pa** passa a armazenar o endereço **DA7**.

Apenas os operadores aritméticos **+** e **–** podem ser utilizados sobre ponteiros. Incrementos ou decrementos no valor de um ponteiro fazem com que o mesmo aponte para posições de memória imediatamente posteriores ou anteriores à atual, respectivamente.

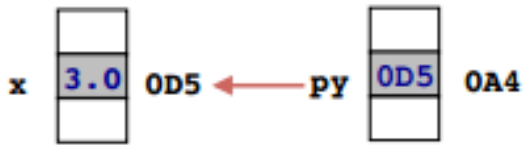
Com isso, pode-se utilizar ponteiros para percorrer toda a memória do computador em incrementos inteiros definidos pelo usuário.

Ponteiros

Para armazenar endereços em um ponteiro basta atribuir-lhe o endereço de alguma variável do mesmo tipo base:

```
py = &x;          // py aponta para x
```

Esquematicamente:




```
char   c, *pc;
int    i, *pi;
double x, *px;

pc = &c;
printf("Tamanho de c: %dB\tEndereco de c: %X\t Próximo endereco: %X\n", sizeof(c), pc, pc+1);
pi = &i;
printf("Tamanho de i: %dB\tEndereco de i: %X\t Próximo endereco: %X\n", sizeof(i), pi, pi+1);
px = &x;
printf("Tamanho de x: %dB\tEndereco de x: %X\t Próximo endereco: %X\n", sizeof(x), px, px+1);
```

Tipos de referência

- Referência direta: acesso à memória através do identificador associado ao endereço.
- Referência indireta: acesso à memória através do ponteiro associado ao endereço.

```
int a, *pa;
pa = &a;
```

```
// Referência direta
a = 500;
printf("Valor de a: %d\n", a);
printf("Endereco de a: %X\n", &a);
a++;
printf("%d" ,a);
```

```
// Referência indireta
*pa = 500;
printf("Valor de a: %d\n", *pa);
printf("Endereco de a: %X\n", pa);
(*pa)++;
printf("%d" ,*pa);
```

ATENÇÃO: pa ≠ *pa

pa significa um endereço; ***pa** significa "o conteúdo armazenado no endereço **pa**" ou "o valor apontado por **pa**."

Ponteiros e vetores

Trabalhar com ponteiros em estruturas seqüenciais (*strings*, vetores e matrizes) pode melhorar o desempenho de programas.

O endereço inicial de um vetor corresponde ao nome do mesmo.

```
char s[80], *ps, c;
ps = s;           // equivalente à ps = &s[0];

// Atribuir o elemento da posição 4 da string s à variável c

c = s[4];         // indexação de vetores, ou
c = *(ps + 4);    // aritmética de ponteiros
```

Alocação de memória

A linguagem C exige que todas as constantes e variáveis de um programa sejam declaradas antes de serem utilizadas.

Isso determina, por exemplo, que estruturas seqüenciais (vetores, matrizes e *strings*) sejam declarados com um tamanho fixo pré-determinado (alocação estática de memória), comprometendo a utilização do programa na execução de dados de diferentes dimensões.

Ponteiros permitem definir o tamanho de estruturas seqüenciais em tempo de execução do programa.

Este recurso denomina-se alocação dinâmica de memória.

Exercícios I

1) Escreva um programa que contenha duas variáveis inteiras. Leia essas variáveis do teclado. Em seguida, compare seus endereços e exiba o conteúdo do maior endereço.

2) Crie um programa que contenha um array de float contendo 10 elementos. Imprima o endereço de cada posição desse array.

3) Crie um programa que contenha um array de inteiros contendo cinco elementos. Utilizando apenas aritmética de ponteiros, leia esse array do teclado e imprima o dobro de cada valor lido.