Estruturas de Dados - Aula Extra 2

Estruturas

Modelos de “pastas” para guardar variáveis:

```
struct nome_tipo {
tipo_var_1      nome_1;
tipo_var_2      nome_2;
...
tipo_var_n      nome_n;
};
```

Uso:

```
nome_tipo nome;
cin >> nome_tipo.nome_1;
cout << nome_tipo.nome_2;
```

Exemplos:

```
struct Aluno {
      int matricula;
      float nota;
      };
```

```
Aluno a;
a.matricula = 2012014308;
cin >> a.nota;
cout << a.matricula;
cout << a.nota;
```

Pode ser usado normalmente como variável.

Estruturas – Exemplo Prátco: pilha

O que define uma pilha?

vetor + topo

```
struct Pilha {
      int v[50];
      int topo;
};
```

Qual a vantagem?

```
int empilhar(Pilha p, int v);
int desempilhar(Pilha p, int &v);
```

Acompanhe o professor na aula!

Estruturas - Exercícios

1) Crie uma estrutura que represente um **produto** em estoque, contendo um número **identificador**, uma **quantidade** disponível e o **preço** atual do produto.
2) Crie uma função que **imprime** um produto.
3) Pegue a solução da aula de **Fila Circular** e modifique a fila para que ela trabalhe com os produtos, ao invés de simples números inteiros.
4) Modifique o programa para que ele leia as informações de vários produtos, até que o código de produto digitado seja zero.
5) Modifique o programa para que ele imprima todos os produtos da lista.
6) DESAFIO: você consegue transformar essa lista circular em uma estrutura?

Ponteiros

Cada variável tem um endereço na memória: podemos obter esse endereço usando o operador **&**:

```
int x = 0;
cout << &x;
```

Se quisermos guardar o endereço de uma variável na memória, precisamos usar uma variável de um tipo especial, chamada **variável apontadora** ou, simplesmente, **ponteiro**. Os ponteiros servem para guardar endereços de tipos específicos de variáveis. Assim, um ponteiro de inteiros serve para guardar endereços de variáveis inteiras e um ponteiro de floats serve para guardar endereços de floats.
Declaramos um ponteiro com o uso do símbolo ***** na frente do nome da variável:

```
int *p;
```

Um ponteiro é uma variável como outra qualquer e, obviamente, ele possui um endereço:

```
int *p;
cout << &p;
```

A diferença dessa variável para outras é que ela serve **apenas** para guardar endereços:

```
int *p;
int i = 0;
p = i;          // Isso está errado!
p = &i;         // Isso está correto!
```

Mas, porque o nome ponteiro?

Estas variáveis chamam-se apontadoras ou ponteiros porque elas não guardam um dado de nossa aplicação, mas sim o **endereço de um dado de nossa aplicação**. Assim, elas indicam (ou apontam) em que lugar da memória está o nosso dado.
Quando queremos que o computador acesse o valor apontado pela variável, usamos novamente o ***** na frente do nome da variável:

```
int i =5;
int *p;
p = &i;
cout << " P:" << p << endl;
cout << "*P:" <<  *p << endl;
i++;
cout << " P:" <<  p << endl;
cout << "*P:" <<  *p << endl;
```

Por que isso ocorre? Observeo endereço de i:

```
int i =5;
int *p;
p = &i;
cout << "&I:" << i << endl;
cout << " P:" << p << endl;
cout << "*P:" <<  *p << endl;
i++;
cout << "&I:" << i << endl;
cout << " P:" <<  p << endl;
cout << "*P:" <<  *p << endl;
```

Podemos usar ponteiros para estruturas também, mas usamos um símbolo diferente para acessar o conteúdo. Exemplo:

```
struct Aluno {
      int matricula;
      float nota;
      };

Aluno a, *p;
a.matricula = 1;
a.nota = 9.0;
p = &a;
cout << p->matricula << endl;
cout << p->nota << endl;
```

Os ponteiros são importantes porque, se quisermos criar um novo aluno durante o nosso programa (sem ter reservado espaço para ele), receberemos um ponteiro para ele. O comando usado para criar elementos "em tempo de execução" é o **new**, conforme indicado a seguir.

```
struct Aluno {
      int matricula;
      float nota;
      };

Aluno *p;
p = new Aluno;
p->matricula = 1;
p->nota = 9.0;
cout << p->matricula << endl;
cout << p->nota << endl;
```

Quando criamos elementos desta forma, é nossa responsabilidade apaga-los quando eles não são mais úteis, usando o comando **delete**. Acompanhe o programa com o professor.

Ponteiros - Exercícios

1) Com base na estrutura **Produto**, proposta nos exercícios de estrutura, faça um programa que crie um produto "em tempo de execução" com o **new** e, depois de imprimi-lo, remova-o com o delete.
2) Modifique o programa acima para que ele crie 5 produtos em tempo de execução, imprima os 5 e, posteriormente, remova-os da memória.
3) Modifique o programa acima para que, além de imprimir os conteúdos de cada produto, também imprima o endereço na memória de cada produto.