Estrutura de Dados

Aula 01 – Apresentação da Matéria e Revisão

Prof. Marcos Nava

Ementa

- Apresentação da Matéria e Revisão
- Pilhas
- Filas
- Listas
- Listas Ligadas
- Listas Duplamente Ligadas
- Árvores binárias

Avaliações

- Projetos em dupla
- 4 projetos: Pilha, Fila, Lista, Simulador de Banco de Dados
- P1 + P2 + P3 com peso 5
- P4 com peso 5
- Veja as datas no Siga

Revisão - Vetor

- O que é um vetor?
- Exemplo de vetor
- Como os dados são armazenados na memória

Revisão - Vetor

- O que é um vetor?
  - Vetor é um agrupamento de várias variáveis de um mesmo tipo acessadas por um índice.

Revisão - Vetor

- Exemplo de vetor

```
#include <stdio.h>
int dados[5];
int main()
{
    dados[3] = 7;
    return 0;
}
```

Revisão - Vetor

- Como os dados são armazenados na memória

Imaginamos um inteiro com 2 bytes

```
int x[3];
x[0] = 4;
x[1] = 2;
X[2] = 735;
```

Endereço	Conteúdo Decimal	Conteúdo Binário
C000	4	00000100
C001	0	00000000
C002	2	00000010
C003	0	00000000
C004	223	11011111
C005	2	00000010

Revisão - Vetor

- Como os dados são armazenados na memória

Os bytes são armazenados invertidos e cada dois são usados para exprimir um índice.

Endereço	Conteúdo Decimal	Conteúdo Binário
C000	4	00000100
C001	0	00000000
C002	2	00000010
C003	0	00000000
C004	223	11011111
C005	2	00000010

Revisão - Vetor

- Como os dados são armazenados na memória

  Por exemplo, x[2] são representados pelos endereços C004 e C005:

  C004 – 11011111 - 223

  C005 – 00000010 – 2

  Quando realinhamos os bytes temos:

  0000001011011111

Revisão - Vetor

- Como os dados são armazenados na memória

  Fazendo a conversão:

0	0×2^{15}	0
0	0×2^{14}	0
0	0×2^{13}	0
0	0×2^{12}	0
0	0×2^{11}	0
0	0×2^{10}	0
1	1×2^{9}	512
0	0×2^{8}	0

1	1×2^{7}	128
1	1×2^{6}	64
0	0×2^{5}	0
1	1×2^{4}	16
1	1×2^{3}	8
1	1×2^{2}	4
1	1×2^{1}	2
1	1×2^{0}	1

Somando tudo temos:

$512 + 128 + 64 + 16 + 8 + 4 + 2 + 1 =$

735

Revisão - Ponteiros

- A memória do computador é um grande vetor e os ponteiros são os seus índices.
- Eles respeitam o tipo de dado para caminhar entre os índices
- Eles dão suporte para determinar como vemos estes dados

Revisão - Ponteiros

- Por exemplo:

```
int x[3];
x[0] = 4;
x[1] = 2;
x[2] = 735;
int *p;
p = x;
```

- Declaramos o vetor x
- Adicionamos os valores
- Criamos o ponteiro
- Apontamos para x.
- Note que todo vetor é um ponteiro, portanto tanto faz utilizarmos como no exemplo ou p = &x;

Revisão - Ponteiros

Endereço	Conteúdo Decimal
C000	4
C001	0
C002	2
C003	0
C004	223
C005	2

À partir daí temos p apontando para o primeiro inteiro.

p contém o endereço do primeiro inteiro, ou seja, C000.

Se imprimirmos *p teríamos 4

Se imprimirmos p teríamos C000

Se imprimirmos &p teríamos a posição de memória que p ocupa

Revisão - Ponteiros

Endereço	Conteúdo Decimal
C000	4
C001	0
C002	2
C003	0
C004	223
C005	2

Ao fazermos o comando p++, o ponteiro será acrescentado em um inteiro (2 bytes).

Assim p agora possui C002

Revisão - Struct

- Struct é uma maneira de agrupar variáveis de diferentes tipos em uma estrutura única.
- É quase um objeto. A diferença básica é que não tem código atrelado aos dados.
- É uma maneira simples de lidar com várias variáveis como se fosse um único agrupamento de bytes.

Revisão - Struct

```
struct estrutura
{
      int cod;
      char desc[40];
      float valor;
} produto;
```

- Foi criada uma estrutura e a variável produto foi criada
- Para acessar o valor do produto usamos:

```
produto.valor
```

Revisão - Struct

```
struct estrutura produto2;
produto2.cod = 2;
strcpy(produto2.desc,”bolo”);
produto2.valor = 12.45;

struct estrutura *p;
p = &produto2;
p->valor = 120.45;
```

- Foi criada uma outra variável chamada produto2 e colocado valores nos seus campos
- Criamos agora um ponteiro para a estrutura
- Finalmente alteramos seu valor. Quando temos um ponteiro não podemos mais usar o ponto (.) para acessar as variáveis membros, mas sim a seta (->)

Revisão - Recursividade

- É quando uma função chama-se a si mesma n vezes fazendo um loop controlado.
- Não deve ser usada para tudo
- Existem problemas que ficam muito mais simples com o uso da recursividade.

Revisão - Recursividade

- Vamos ver um exemplo usando o conceito de fatorial:
  - 5! = 5 x 4 x 3 x 2 x 1
    4! = 4 x 3 x 2 x 1
    3! = 3 x 2 x 1
    2! = 2 x 1
    1! = 1

Revisão - Recursividade

- Podemos simplificar (recursivamente) assim:
  - 5! = 5 x 4!
    4! = 4 x 3!
    3! = 3 x 2!
    2! = 2 x 1!
    1! = 1

Revisão - Recursividade

```
int fatorial(int x)
{
   int i;
   int fat = 1;
   for(i=1;i<=x;i++)
      fat = fat * i;
   return fat;
}
```

```
int fatorial(int x);
{
   if(x==1)
      return 1;
   else
      return x * fatorial(x-1);
}
```

Revisão - Recursividade

Imagine uma chamada assim:

```
int resultado;
resultado = fatorial(4);
printf(“%d”, resultado);
```

Revisão - Recursividade

```
int fatorial(int x);
{
    if(x==1)
        return 1;
    else
        return x * fatorial(x-1);
}
```

```
fatorial(4);
return 4 * fatorial(4-1);

fatorial(3);
return 3 * fatorial(3-1);

fatorial(2);
return 2 * fatorial(2-1);

fatorial(1);
return 1;
```

Revisão - Recursividade

Devolvendo os valores temos:

```
fatorial(4);
return 4 * fatorial(4-1);

fatorial(3);
return 3 * fatorial(3-1);

fatorial(2);
return 2 * 1;

1
```

Revisão - Recursividade

Devolvendo os valores temos:

```
fatorial(4);
return 4 * fatorial(4-1);

fatorial(3);
return 3 * 2;

2
```

Revisão - Recursividade

Devolvendo os valores temos:

```
fatorial(4);
return 4 * 6;

6
```

Revisão - Recursividade

Devolvendo os valores temos: 24

```
resultado = 24;
printf(“%d”, resultado);
```