Linguagem C: agregados heterogêneos, arquivos binários, recursividade

Prof. Críston

Algoritmos e Programação

Agregados heterogêneos

- Permitem agrupar variáveis de diferentes tipos em um único registro

```
struct nome_estrutura
{
    tipo1 campo1;
    tipo2 campo2;
    ...
    tipoN campoN;
} nome_variavel;
```

Exemplo

```
#include <string.h>
main()
{   struct aluno
    {
        int matricula;
        char nome[50];
        double nota;
        int faltas;
    } joao;

    joao.matricula = 1234;
    strcpy(joao.nome, "Joao da Silva");
    joao.nota = 8.0;
    joao.faltas = 5;

    printf("%d, %s, %f, %d\n", joao.matricula, joao.nome,
                               joao.nota, joao.faltas);
}
```

Exemplo

```
#include <string.h>
main()
{   struct aluno
    {
        int matricula;
        char nome[50];
        double nota;
        int faltas;
    };
    struct aluno joao;

    joao.matricula = 1234;
    strcpy(joao.nome, "Joao da Silva");
    joao.nota = 8.0;
    joao.faltas = 5;

    printf("%d, %s, %f, %d\n", joao.matricula, joao.nome,
                               joao.nota, joao.faltas);
```

Definição de um novo tipo

```
typedef tipo nome_tipo;
```

- Exemplos:

```
typedef int inteiro;
inteiro faltas;

typedef struct
    {
        int matricula;
        char nome[50];
        double nota;
        int faltas;
    } aluno;
aluno joao, maria;
```

Exemplo

```
main()
{   typedef struct
    {
        int matricula;
        char nome[50];
        double nota;
        int faltas;
    } aluno;
    aluno joao;

    joao.matricula = 1234;
    strcpy(joao.nome, "Joao da Silva");
    joao.nota = 8.0;
    joao.faltas = 5;

    printf("%d, %s, %f, %d\n", joao.matricula, joao.nome,
                               joao.nota, joao.faltas);
}
```

Inicialização e atribuição

```
typedef struct
{
    int x;
    double y;
} teste;

main()
{
    teste a, b = {1, 2.0};
//  a = {3, 4.0}; ERRO
    printf("%d, %f\n", a.x, a.y);
    a = b;
    printf("%d, %f\n", a.x, a.y);
}
```

Vetor de registros

```
// definição global
typedef struct
    {
        int matricula;
        char nome[50];
        double nota;
        int faltas;
    } aluno;

main()
{
   aluno turma[20];
   ...
   turma[4].nota = 7.0;
   ...
}
```

Lendo registros de um arquivo texto

```
main()
{
    aluno turma[30];
    int i, n = 0;

    FILE *f = fopen("alunos.txt", "r");
    while (fscanf(f, "%d %s %lf %d", &turma[n].matricula,
                  &turma[n].nome, &turma[n].nota,
                  &turma[n].faltas) == 4)
        n++;
    fclose(f);

    for (i=0; i<n; i++)
        printf("%d, %s, %f, %d\n", turma[i].matricula,
               turma[i].nome, turma[i].nota, turma[i].faltas);
}
```

Função para retornar a média da turma

```
double media_turma(aluno turma[], int num_alunos)
{
    double soma = 0;
    int i;
    for (i=0; i < num_alunos; i++)
        soma += turma[i].nota;
    return soma / num_alunos;
}

main()
{
    aluno turma[30];
    ...
    printf("media da turma = %f\n", media_turma(turma, n));
}
```

Gravando em um arquivo binário

```
main()
{
    aluno turma[30];
    int i, n = 0;

    FILE *f = fopen("alunos.txt", "r");
    while (fscanf(f, "%d %s %lf %d", &turma[n].matricula,
          &turma[n].nome, &turma[n].nota, &turma[n].faltas) == 4)
        n++;
    fclose(f);

    f = fopen("alunos.bin", "wb");
    fwrite(turma, sizeof(aluno), n, f);
    fclose(f);
}
```

Lendo de um arquivo binário

```
main()
{
    aluno turma[30];
    int i;

    int n = 2;
    FILE *f = fopen("alunos.bin", "rb");
    fread(turma, sizeof(aluno), n, f);
    fclose(f);

    for (i=0; i<n; i++)
        printf("%d, %s, %f, %d\n", turma[i].matricula,
               turma[i].nome, turma[i].nota, turma[i].faltas);
}
```

Algumas funções úteis na manipulação de arquivos

- **`fseek(FILE *f, long int num_bytes, int origem)`**
  - Origem 0 – início do arquivo
  - Origem 1 – ponto atual
  - Origem 2 – fim do arquivo
- **`rewind(FILE *f)`** – posiciona no início
- **`remove(char *nome_arquivo)`** – apaga arquivo
- **`fflush(FILE *f)`** – esvazia o buffer
- **`feof(FILE *f)`** – testa fim de arquivo (não-zero se fim)

Enumeração

- Fornece identificadores para a sequência numérica 0, 1, 2, 3…
- Define um novo tipo

```
enum nome_tipo {lista_valores} lista_variaveis;

enum nome_tipo {lista_valores};

enum {lista_valores} lista_variaveis;
```

Enumeração

```
enum dias_semana {domingo, segunda, terca, quarta, quinta,
   sexta, sabado};

main()
{
    enum dias_semana d1, d2;
    d1 = segunda;
    d2 = 3;
    if (d1 == terca)
        printf("d1 = terca");
    printf("%d\n", d1);
    printf("%d\n", quinta);
}
```

União

- Permite declarar uma região de memória onde podem ser armazenados valores de vários tipos

```
union nome_tipo
{
    tipo1 nome1;
    tipo2 nome2;
    ...
    tipoN nomeN;
}
```

União

```
union inteiro_real
{
    int inteiro;
    double real; // float real
};

main()
{
    union inteiro_real x;
    printf("%d\n", sizeof(x));
    x.inteiro = 4;
    printf("%d %f\n", x.inteiro, x.real);
    x.real = 5.2;
    printf("%d %f\n", x.inteiro, x.real);
}
```

Recursividade

- Quando uma função chama ela própria
- Alguns problemas têm definição naturalmente recursiva
  - Ex.: números de fibonacci
  - Problema pode ser desmembrado em sub-problemas semelhantes
- Desvantagem: pode produzir muitas chamadas de funções
  - Sempre que uma função A chama uma função B, os valores das variáveis locais de A são guardados para serem utilizados quando a função B encerrar
  - Este processo consome CPU e memória
- Devemos sempre lembrar da condição de parada da recursão, senão teremos loop infinito

Multiplicar dois números naturais

```
int multiplica(int a, int b)
{
    int i, produto = 0;
    for (i=1; i <= b; i++)
        produto += a;
    return produto;
}

// versão recursiva
int multiplica(int a, int b)
{
    if (b == 1) return a; // condição de parada
    return a + multiplica(a, b-1);
}
```

Números de Fibonacci

```
int fibonacci(int n)
{
    if (n < 3) return 1;
    return fibonacci(n-2) + fibonacci(n-1);
}

main()
{
    int i;
    for (i=1; i<10; i++)
        printf("%d ", fibonacci(i));
}
```

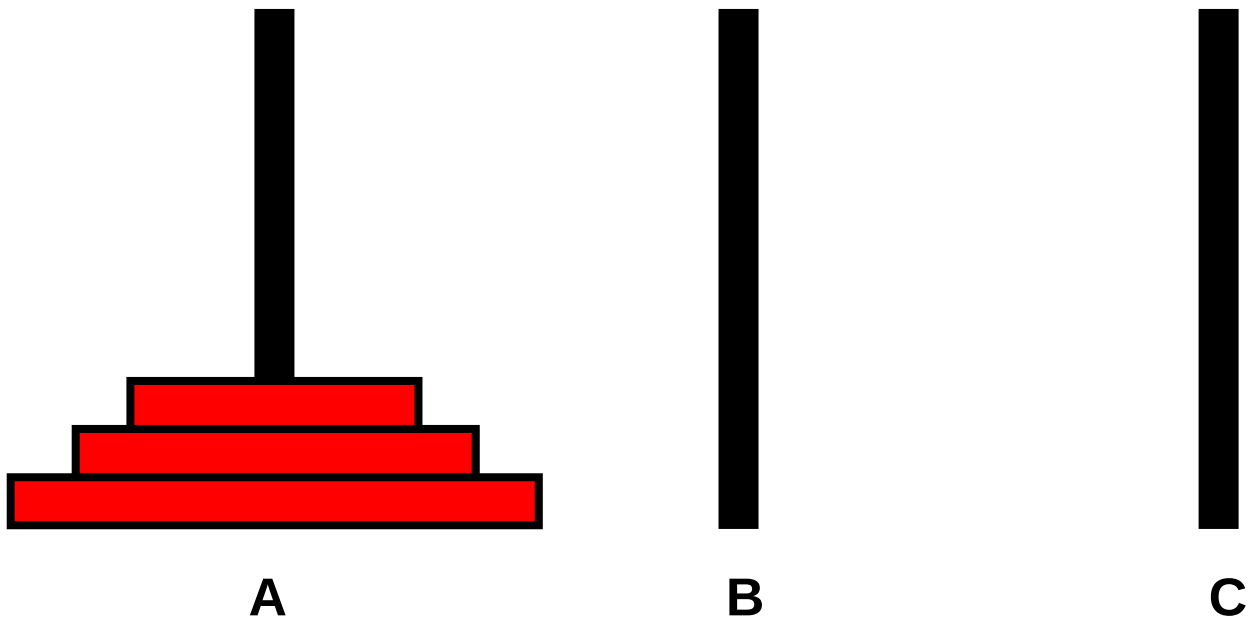
Fatorial

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

main()
{
    int i;
    for (i=1; i <= 10; i++)
        printf("%d! = %d\n", i, fatorial(i));
}
```

Torres de Hanoi

- Transferir os discos da torre A para a torre C, podendo utilizar a torre B
- Um disco não pode ser colocado sobre um disco menor



Torres de Hanoi

```
void hanoi(int n, char origem, char auxiliar, char destino)
{
    if (n == 1) printf("%c -> %c\n", origem, destino);
    else
    {
        hanoi(n-1, origem, destino, auxiliar);
        printf("%c -> %c\n", origem, destino);
        hanoi(n-1, auxiliar, origem, destino);
    }
}

main()
{
    hanoi(3, 'A', 'B', 'C');
}
```