FACAMP
FACULDADES DE CAMPINAS

Estruturas de Dados

Aula 5 – Tipos de dados compostos; alocação e liberação dinâmicas de memória.

1º semestre de 2020
Prof. José Martins Jr

Estruturas

- Os tipos básicos de dados permitem a representação de um conjunto limitado de situações
  - Números e caracteres, que podem ser agrupados em conjuntos de tamanho predefinido (arranjos)
- Pode-se querer definir elementos de dados compostos que possuem diversas características
  - Exemplo: representar dados de endereço (logradouro, número, CEP, bairro, cidade e UF), em um único registro
- Registros podem ser construídos, em C, usando estruturas (*structures*)

Declaração de estrutura

- A forma geral de declaração de uma estrutura, é:

```
struct nome {
        tipo membro1;
        tipo membro2;
        ...
}  variavel1, variavel2 ... ;
```

Exemplo de estrutura

- Uma estrutura para representar o endereço

```
struct endereco1 {
        char *logradouro;
        int numero;
        char *cep;
        char *bairro;
        char *cidade;
        char *uf;
};
```

- Declaração de variável que representa essa estrutura

```
struct endereco1 end;
```

Definição de tipo

- Pode-se definir um novo tipo (typedef) para a estrutura
- Duas formas

```
typedef struct {
        char *logradouro;
        int numero;
        char *cep;
        char *bairro;
        char *cidade;
        char *uf;
} endereco;
```

```
typedef struct _end endereco;
struct _end {
        char *logradouro;
        int numero;
        char *cep;
        char *bairro;
        char *cidade;
        char *uf;
};
```

- O novo tipo **endereco** pode ser usado em declarações

```
endereco end;
```

Tamanho de uma estrutura

- Corresponde à soma dos bytes dos campos
  - Ex.: em ambos, tamanho terá o valor 24

```
int tamanho = sizeof(struct endereco1);
        ou
int tamanho = sizeof(endereco);
```

- Porém, o total deve ser **múltiplo do maior tipo!!**
  - Se o total de bytes somados não for um múltiplo do maior tipo presente entre os campos, é alocada memória complementar, até totalizar o próximo múltiplo

Exemplo

- A soma de bytes da estrutura abaixo é 126

```
typedef struct {
        char logradouro[50];
        int numero;
        char cep[10];
        char bairro[30];
        char cidade[30];
        char uf[2];
} endereco2;
```

- Porém, se verificado o tamanho dela, é de 128

```
printf("%d\n", sizeof(endereco2));
```

Acesso aos campos

- Pode ser realizado de duas formas
  - Com uma variável do tipo declarado pela estrutura: coloca-se um **ponto** entre o nome da variável e o do campo

```
endereco e;
e.logradouro = "Rua X";
```

  - Ou pelo endereço (referência, ponteiro) para uma estrutura: nesse caso, utilizam-se **->** no lugar do ponto

```
endereco *ep, e;
ep = &e;
ep->logradouro = "Rua X";
```

ou

```
(&e)->logradouro = "Rua X";
```

Alocação estática

- Quando uma variável é declarada com um tipo estrutura
  - É reservado espaço suficiente na memória para alocá-la
  - A alocação estática de vários elementos de um tipo estruturado pode ser realizada normalmente, com o uso de arranjos e matrizes

- Exemplo
  - Alocação de espaço contíguo em memória para 10 estruturas do tipo endereco

```
endereco endereco3[10];
```

  - Pode-se acessar cada uma com o índice do arranjo

```
endereco3[2].numero = 100;
```

Obs.: veja o exemplo alocacao_estatica.c fornecido

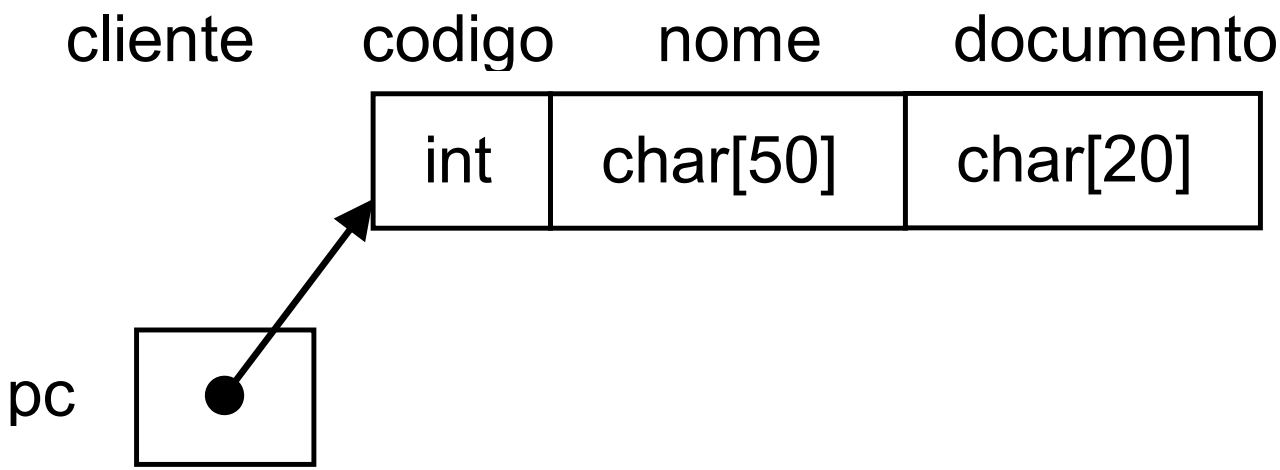
Alocação dinâmica

- Vantagem do uso de estruturas
  - Armazenar dados compostos, ou estruturados
- Desvantagem da alocação estática
  - Dados estruturados podem consumir bastante memória
  - A alocação estática, como em arranjos, não é a melhor solução

- Alocação dinâmica de memória
  - Alocação de espaço na memória, em tempo de execução
  - Usada para guardar dados simples ou compostos (registros)
  - Função `void *malloc(size_t size);`
    - Retorna um ponteiro para void que pode ser convertido para o tipo de ponteiro apropriado para o registro armazenado

Exemplo

```
typedef struct {
      int codigo;
      char nome[50];
      char documento[20];
} cliente;
cliente *pc;
pc = (cliente *) malloc(sizeof(cliente));
```



Acesso aos registros

- O acesso aos campos da estrutura pode ser realizado através do ponteiro, que guarda a sua referência na memória
- Exemplos:

```
pc -> codigo = 1;
strcpy(pc->nome,"José");
scanf("%s", pc->documento);
```

- Note que os campos nome e documento foram declarados como arrays e já foi alocada memória para armazená-los
  - Por esse motivo que foi usada a função strcpy, para cópia da constante (armazenada em outro lugar, quando o programa é executado)

Obs.: veja o exemplo alocacao_dinamica.c fornecido

Liberação da memória alocada

- Todo espaço alocado de memória **deve** ser liberado após o uso
- Função `void free(void *ptr);`
  - Pode ser aplicada à referência (ponteiro)
  - Indicará ao sistema que aquele espaço de memória não deve mais ser reservado e está liberado

- Exemplo:

```
free(pc);
```

Bibliografia

- KERNIGHAN, B. W., RITCHIE, D. M. C, a linguagem de programação: padrão ANSI. Rio de Janeiro: Campus, 1989. 289 p.
- LOOSEMORE, S.; STALLMAN, R. M. et al. The GNU C Library Reference Manual. Disponível no endereço eletrônico: www.gnu.org/software/libc/manual/