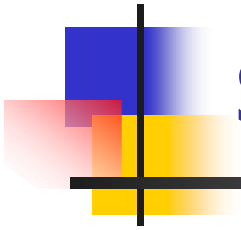


SIN211 Algoritmos e Estruturas de Dados

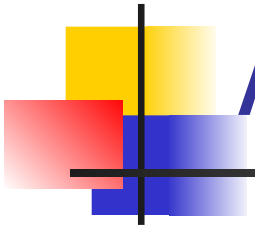
Prof. João Batista Ribeiro

joao42lbatista@gmail.com

UFV

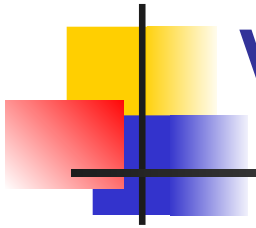
Universidade Federal de Viçosa

Slides baseados no material da Prof.ª Rachel Reis



Assuntos da Aula

- Variáveis e Ponteiros
- Estrutura e Ponteiros
- Passagem de Parâmetro por Referência
- Alocação de Memória
  - Alocação estática de memória
  - Alocação dinâmica de memória
    - *Malloc, Calloc, Free, Realloc*



Variáveis

- Possuem nomes que as identificam
- Armazena um valor
  - Fixo ou pode mudar durante a execução
- Possui um endereço na memória
  - O nome da variável não é nada mais que um apelido para a localização deste endereço

E o que são ponteiros??

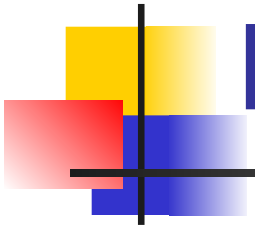
Ponteiros

- O ponteiro é uma variável que guarda o endereço de uma outra variável.
- Como declarar um ponteiro?

int *px; → **px é um ponteiro do tipo int**

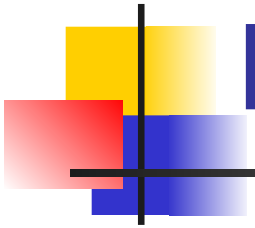
- Como atribuir um valor a um ponteiro?

```
int x = 5;          // declara uma varíavel x
int *px;            // declara uma variável do tipo ponteiro
px = &x;            // inicializa o ponteiro px com o endereço de x
*px = 10;           // altera o valor da variável x
```



Ponteiros

- O valor inicial de um ponteiro depois que ele é declarado é sempre 0 ou NULL.
- NULL não é um endereço válido.
- Logo, você deve **sempre** inicializar um ponteiro com um endereço.



Ponteiros

- Exemplo 1

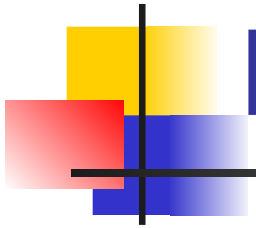
```
int a = 5;
int *p = NULL;
*p = a;
```

→ Incorreto

- Exemplo 2

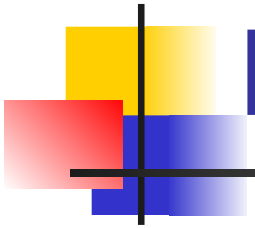
```
int a = 5;
int *p = NULL;
p = &a;
```

→ Correto



Estrutura

```
typedef struct sPessoa {
    char primNome[21];
    char ultNome[21];
     int idade;
} Pessoa;
```



Estrutura e Ponteiros

- As variáveis do tipo estrutura são tratadas da mesma forma que variáveis de tipos básicos

  – É possível definir variáveis do tipo ponteiro para estruturas

  – Ex: Pessoa *ptrPes;

Estrutura e Ponteiros - Exemplo

```
typedef struct sPessoa {
    char primNome[21];
    char ultNome[21];
    int idade;
}Pessoa;

int main() {
    Pessoa pes;
    Pessoa *ptrPes;
    ptrPes = &pes;
    strcpy (pes.primNome, "Joao");
    strcpy (pes.ultNome, "Silva");
    ptrPes->idade = 20;
}
```

Estrutura e Ponteiros

- Uma outra forma de acesso a membros de estruturas facilita a notação quando ponteiros para estruturas estão envolvidos

```
(*ptrPes).idade = 20;
```

ou

```
ptrPes->idade = 20;
```

Passagem de parâmetro por referência

- Como uma função pode alterar o valor das variáveis da função chamadora?

  - Primeiro: o programa chamador, em vez de passar valores para a função, passa seus endereços

  - Segundo: a função chamada deve declarar os endereços recebidos como ponteiros.

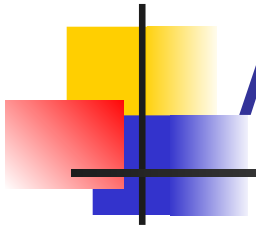
Passagem de parâmetro por referência

- Deve-se utilizar o operador '&' na chamada da função

```
int main() {
    int x = 3, y= 7;
    altera(&x, &y);
    printf("Valor de x: %d, \nValor de y: %d", x, y);
    return 0;
}
```

- Utiliza o '*' para declarar o ponteiro na definição da função

```
void altera(int *px, int *py) {
   *px = 37;
   *px = 79;
}
```

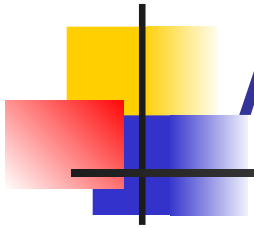


Alocação de Memória Estática

- Definição
  - As variáveis de um programa têm alocação estática se a quantidade total de memória utilizada pelos dados é previamente conhecida e definida de modo imutável, no próprio código-fonte do programa

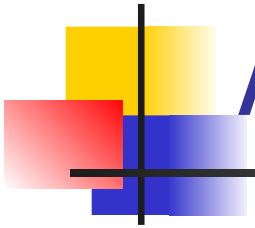
- Na Linguagem C, o espaço de memória utilizado por um programa para armazenar dados normalmente é indicado pelo programador no momento da declaração de variáveis

- Exemplo
  - char vetor [11];



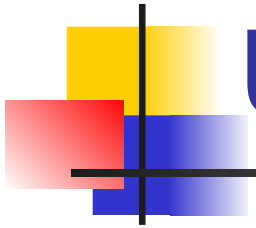
Alocação de Memória Estática

- Vantagens da Alocação Estática (Sequencial)
  - Indexação eficiente - facilita o acesso a uma posição qualquer de um vetor ou matriz
- Desvantagens da Alocação Estática
  - É necessário saber de antemão a quantidade máxima de dados de um conjunto a ser utilizada por um programa (pode acarretar em desperdício de espaço quando a quantidade máxima não é utilizada para atender a uma dada situação)
  - Inserção e remoção são custosas quando envolvem algum esforço para movimentar/deslocar elementos, de modo a abrir espaço para inserção (ex. inserção ordenada), ou de modo a ocupar espaço liberado por um elemento que foi removido



Alocação de Memória Dinâmica

- Definição
  - As variáveis de um programa têm alocação dinâmica quando suas áreas de memória – não declaradas no programa – passam a existir durante execução, ou seja, o programa é capaz de criar novas variáveis enquanto executa
- Como trabalhar com alocação dinâmica?
  - Para trabalhar com variáveis alocadas dinamicamente é necessário o uso de apontadores (ponteiros) e o auxílio de funções que permitam reservar (e liberar), em tempo de execução, espaços da memória para serem usados pelo programa

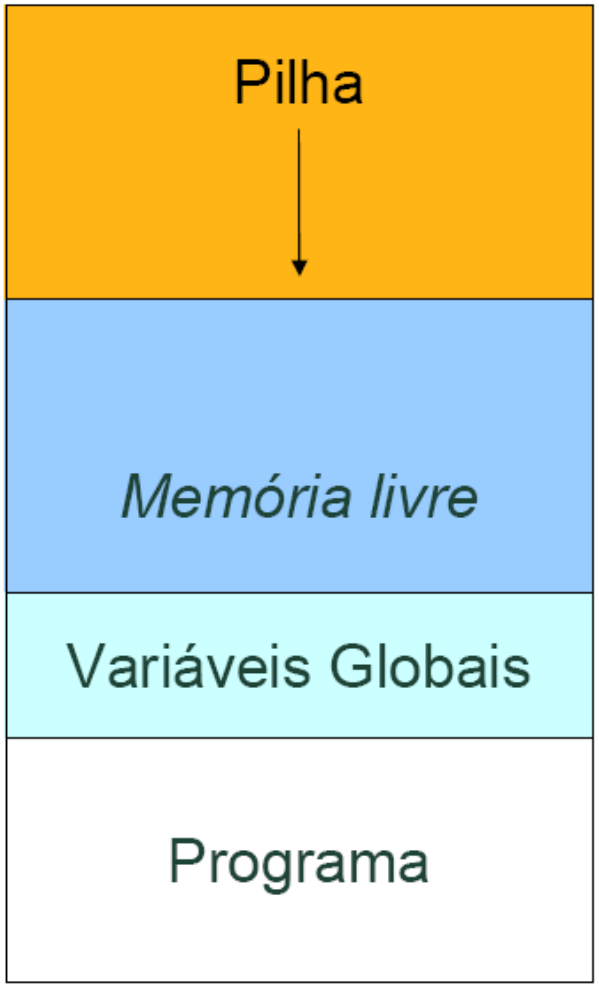


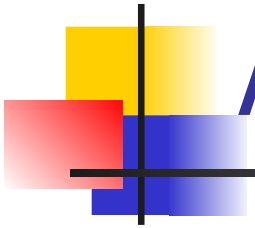
Uso da memória principal

- A memória principal pode ser usada para armazenar variáveis locais e globais – incluindo *arrays* e estruturas.
  - Variáveis globais: o armazenamento é fixo durante todo o tempo de execução do programa
  - Variáveis locais: o armazenamento é feito na pilha do computador

- Este tipo de uso da memória exige que o programador saiba, antemão, a quantidade de armazenamento necessária para todas as situações

Uso da memória principal

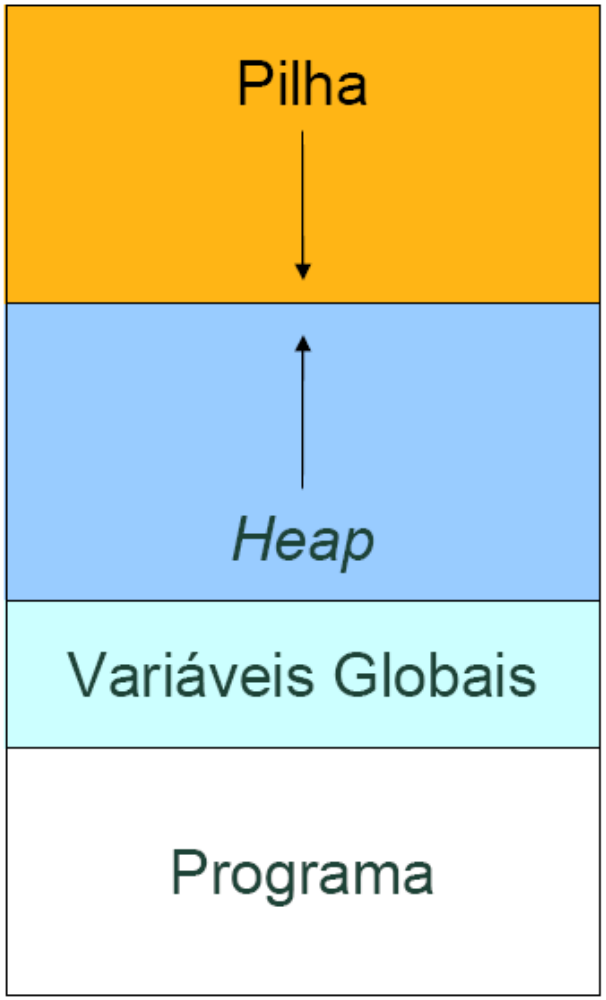


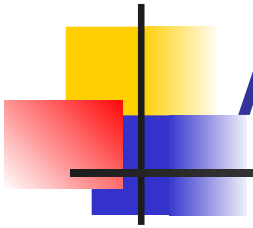


Alocação Dinâmica

- Para oferecer um meio de armazenar dados em **tempo de execução**, há um subsistema de **alocação dinâmica**
- O armazenamento de forma dinâmica é feito na região de memória livre, chamada de ***heap***
- A pilha cresce em direção inversa ao *heap*
  - Conforme o uso de variáveis na pilha e a alocação de recursos no *heap*, a memória poderá se esgotar e um novo pedido de alocação de memória falhar

Alocação Dinâmica



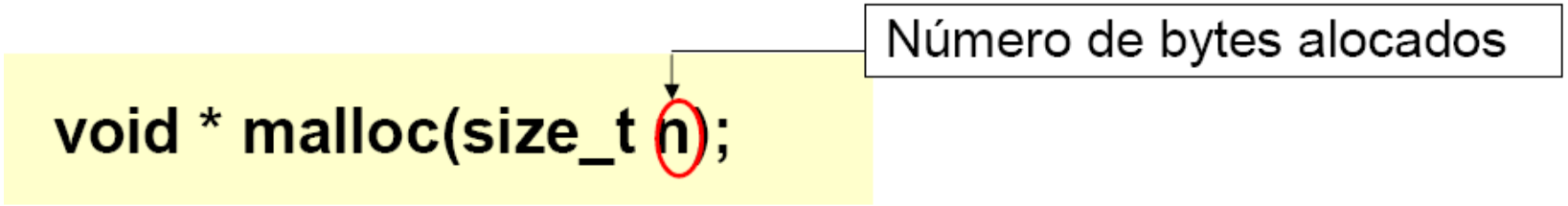


Alocação Dinâmica

- No núcleo do sistema de alocação dinâmica em C estão as funções:
  - **malloc()**
  - **free()**
- Quando **malloc()** é usada, uma porção de memória livre é alocada
- Quando **free()** é usada, uma porção de memória alocada é novamente devolvida para o sistema
- Os protótipos das funções de alocação dinâmica estão na biblioteca **stdlib.h**

Alocação Dinâmica - *malloc*

- Função *malloc*



/*
retorna um ponteiro void para n bytes de memória não iniciados. Se não há memória disponível malloc retorna NULL
*/

- Exemplo de uso:

```
int *pi;
pi= (int *) malloc (sizeof(int));
// aloca espaço para um inteiro
```

Alocação Dinâmica - *malloc*

```
ptr = malloc(size);
```

- A função **malloc()** devolve um ponteiro para o primeiro byte de uma região de memória do tamanho ***size***
- Caso não haja memória suficiente, retorna nulo (**NULL**)

Alocação Dinâmica - *malloc*

```
float *v;
int n;
printf("Quantos valores? ");
scanf("%d", &n);
v = (float *) malloc(n * sizeof(float) );

if(v!=NULL)
   // manipula a região alocada
```

- Uma porção de memória capaz de guardar **n** números reais (float) é reservada, ficando o apontador **v** a apontar para o endereço inicial dessa porção de memória
- O *cast* da função malloc() - (float *) - garante que o apontador retornado é para o tipo especificado na declaração do apontador. Certos compiladores requerem obrigatoriamente o *cast*

Alocação Dinâmica - *free*

- A função *free* é usada para liberar o armazenamento de uma variável alocada dinamicamente

```
int *pi;
pi= (int *) malloc (sizeof(int));

free(pi);
```

Alocação Dinâmica - *free*

```
free(ptr);
```

- A função **free()** devolve ao *heap* a memória apontada pelo ponteiro **ptr**, tornando a memória livre
- Deve ser chamada apenas com ponteiro previamente alocado

Alocação Dinâmica – Exemplo 1

```
int main () {
      int *p;
      p = (int *) malloc( sizeof(int) );
      if ( p == NULL ) {
           printf("Não foi possível alocar memória.\n");
           exit(1);
       }
     *p = 5;
      printf("%d\n", *p);
      free(p);
      return 0;
}
```

Alocação Dinâmica – Exemplo 2

```
struct sEndereco {
    char rua[20];
    int numero;
};
```

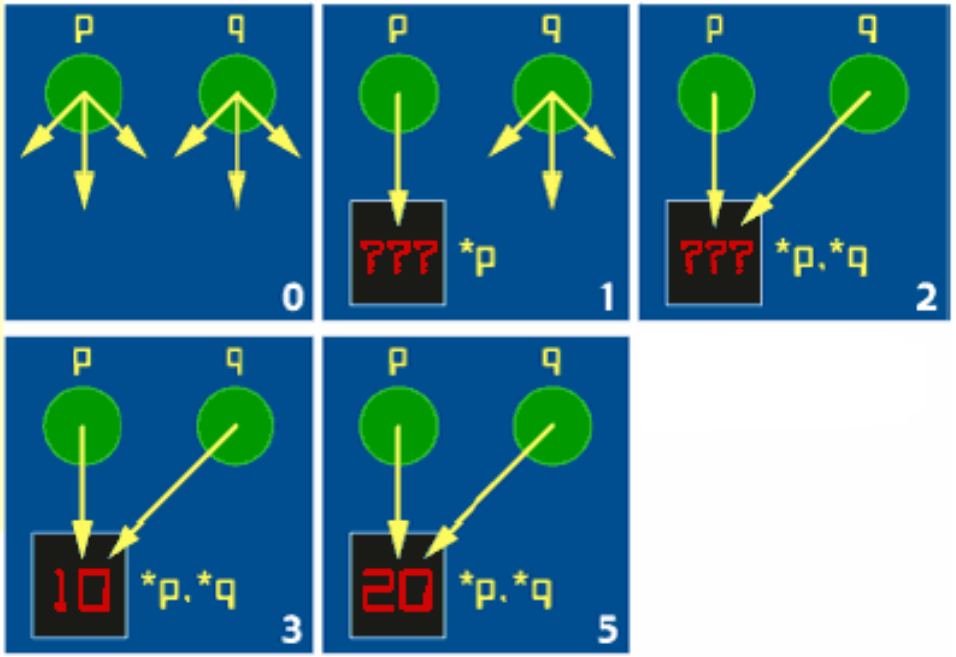
```
int main () {
    struct sEndereco *pend;
    pend = (struct sEndereco *)malloc( sizeof(struct sEndereco) );
    if ( pend == NULL ) {
        printf("Não foi possível alocar memória.\n");
        exit(1);
    }
    return 0;
}
```

Alocação Dinâmica – Exemplo 3

```
int main () {
    int *p, *q;
    p = (int *) malloc(sizeof(int));
    q = p;
    *p = 10;
    *q = 20;
    free(p);
    q = NULL;
    return 0;
}
```

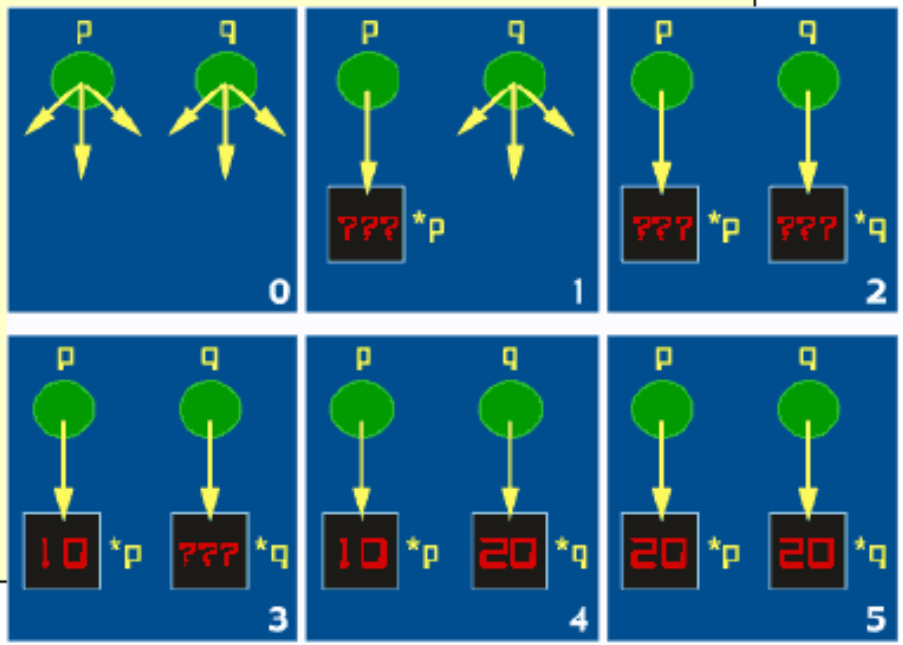
Alocação Dinâmica – Exemplo 3

```
int main(void){
   int *p, *q;
   p = (int *)malloc(sizeof(int));
   q = p;
   *p = 10;
  *q = 20;
   free(p);
   q = NULL;
   return 0;
}
```



Alocação Dinâmica – Exemplo 4

```
int main(void){
   int *p, *q;
   p = (int *) malloc(sizeof(int));
   q = (int *) malloc(sizeof(int));
   *p = 10;
  *q = 20;
   *p = *q;
   free(p);
   free(q);
   return 0;
}
```



Alocação Dinâmica – Exemplo 5

```
int main () {
        int *array1;
        array1 = malloc(100 * sizeof(int));
        if (array1 == NULL ) {
            printf("Não foi possível alocar memória.\n");
            exit(1);
        }
        array1[99] = 301;
        printf("%d\n", array1[99]);
        free(array1);
        return 0;
}
```

Alocação Dinâmica – Exemplo 6

```
int main () {
   CLIENTE *pc;
   pc = (CLIENTE *) malloc( 50 * sizeof(CLIENTE));
   gets( pc[0].nome );
   scanf("%d", &pc[0].idade );
   printf("%s", pc[0].nome);
   printf("%d", pc[0].idade);

   free(pc);
   return 0;
}
```

```
typedef struct cli{
   char nome[30];
   int idade;
} CLIENTE;
```

Alocação Dinâmica - *calloc*

Número de bytes alocados: n * size

```
void * calloc(size_t n, size_t size);
```

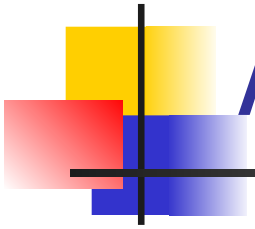
/*calloc retorna um ponteiro para um vetor com n elementos de tamanho size cada um ou NULL se não houver memória disponível. Os elementos são iniciados em zero
*/

- Exemplo de uso

```
float *ai = (float *) calloc (n, sizeof(float));
/* aloca espaço para um vetor de n float */
```

- Toda memória não mais utilizada deve ser liberada pela função free:

```
free(ai);  /* libera todo o vetor*/
```



Alocação Dinâmica - *realloc*

- É possível alterar o tamanho do bloco de memória reservado, utilizando a função ***realloc()***

- Esta função salva os valores anteriormente digitados em memória, até ao limite do novo tamanho (especialmente importante quando se reduz o tamanho do bloco de memória)

Alocação Dinâmica - *realloc*

- Exemplo de uso da função realloc:

```
int *a;
a = (int *) malloc( 10 * sizeof(int) );
...
a = (int *) realloc( a, 23 * sizeof(int) );
...
free(a);
```

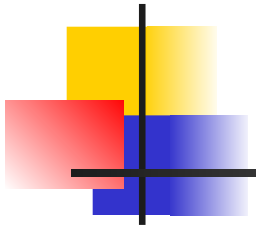
Novo tamanho do bloco de memória

- A chamada da função *realloc*() recebe como argumentos um apontador para o bloco de memória previamente reservado pela função *malloc*() ou *calloc*() de forma a identificar qual a porção de memória será redimensionada, e o novo tamanho absoluto para o bloco de memória

Alocação Dinâmica - *realloc*

```
ptr = realloc(ptr, size);
```

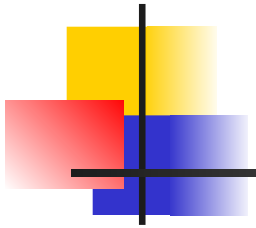
- A função **realloc()** modifica o tamanho da memória alocada previamente e apontada pelo ponteiro ***ptr***
- Caso não haja memória suficiente, retorna nulo (**NULL**)



Leituras Recomendadas

- SCHILDT, H. C Completo e Total. 3.ed. Pearson-Makron Books. 1997.
  - Ponteiros: Pág. 113
  - Alocação dinâmica: Pág. 128
  - Funções: Pág. 138
  - Estrutura: Pág. 167

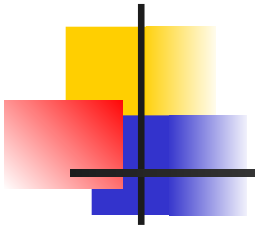
- FORBELLONE, A.L.V.; EBERSPACHER, H.F. Lógica de Programacao: a construcao de algoritmos e estruturas de dados. 3.ed. Pearson-Makron Books.2005.
  - Vetores: Pág. 69
  - Estruturas: Pág. 85



Exercícios

1 Codifique, compile e execute um programa em C que contenha uma estrutura data com os campos: dia (inteiro), mês (caracter) e ano (inteiro). Crie uma única estrutura e armazene a data do **seu** aniversário. Em seguida crie uma função separada para imprimir as informações armazenas na estrutura.

2 Codifique, compile e execute um programa em C que contenha uma estrutura data com os campos: dia (inteiro), mês (caracter) e ano (inteiro). Crie um vetor de datas para armazenar/imprimir a data de aniversário dos 36 alunos matriculados na disciplina de programação. Implemente funções distintas, além da função main, para armazenar/imprimir os valores armazenados no vetor.



Exercícios

3 Codifique, compile e execute um programa em C que contenha uma estrutura data com os campos: dia (inteiro), mês (caracter) e ano (inteiro). Crie um vetor de datas para armazenar/imprimir a data de aniversário dos n alunos matriculados em uma disciplina qualquer, sendo n um valor digitado pelo usuário. Implemente funções distintas, além da função main, para armazenar/imprimir os valores armazenados no vetor.

4 Crie um programa em C que aloque dinamicamente um vetor de automóveis de uma concessionária de carros com os seguintes dados: modelo, ano de fabricação e tipo de combustível.