UNIVERSIDADE FEDERAL DE ITAJUBÁ

Algoritmos e Estrutura de Dados I

COM 111

Alocação Dinâmica

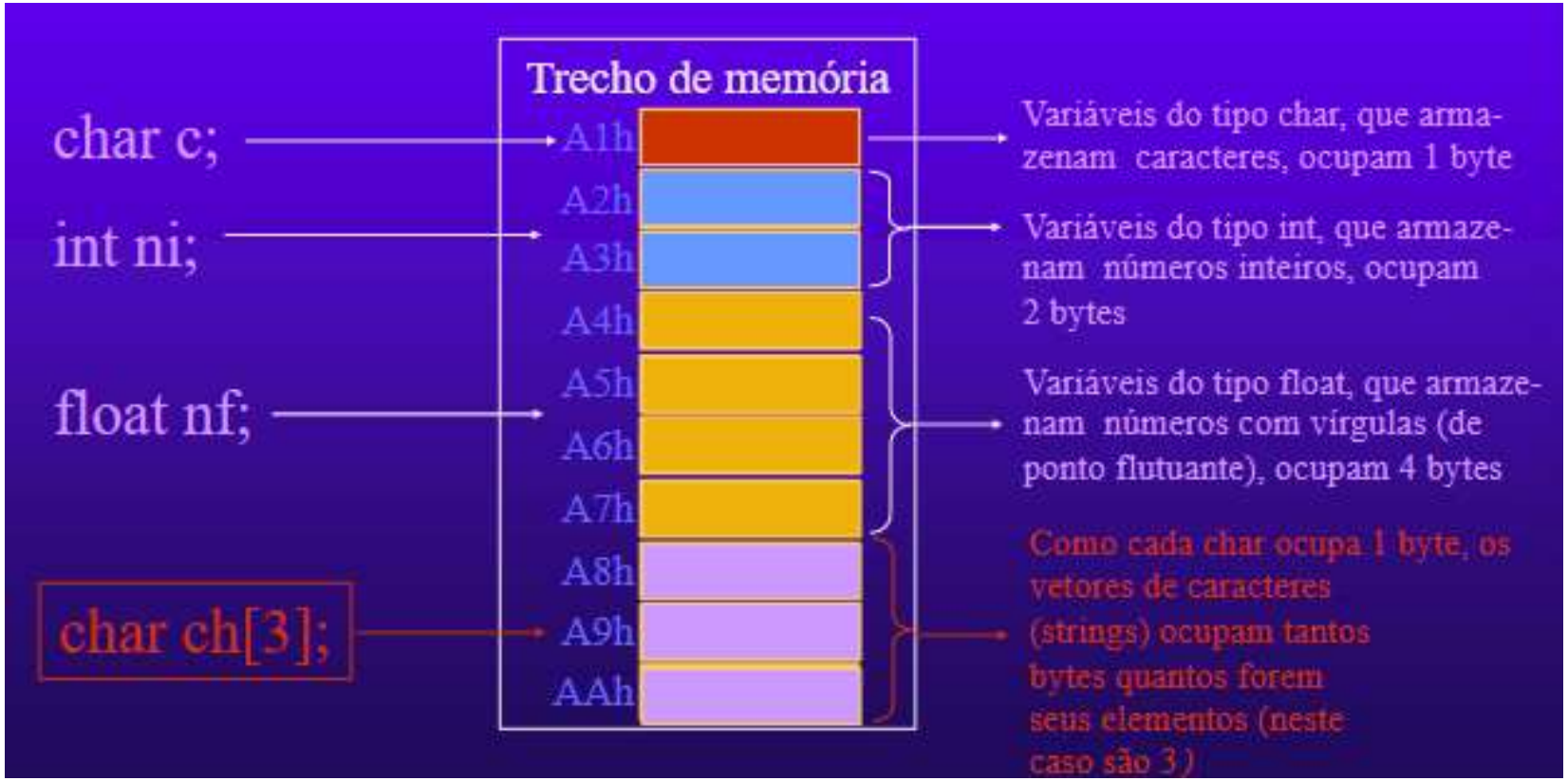
Vanessa Cristina Oliveira de Souza

Introdução

- **Variável** é um local reservado na memória para armazenar um tipo de dado.

- Ao declarar uma variável, o programador está ALOCANDO MEMÓRIA!
  - Ao declarar uma variável, o **COMPILADOR** reserva uma região na memória para ela.
  - A quantidade de memória depende do tamanho do tipo de dado para o qual a variável foi declarada.

Introdução



- FONTE : http://slideplayer.com.br/slide/53993/

Introdução

- Veja os exemplos abaixo:
  - int i;
  - float k;
  - char c ;
  - int vet[30];

- Os códigos acima são declarações de variáveis
  - ALOCAM MEMÓRIA ESTATICAMENTE
    - OU SEJA, ANTES QUE O PROGRAMA COMECE SER EXECUTADO

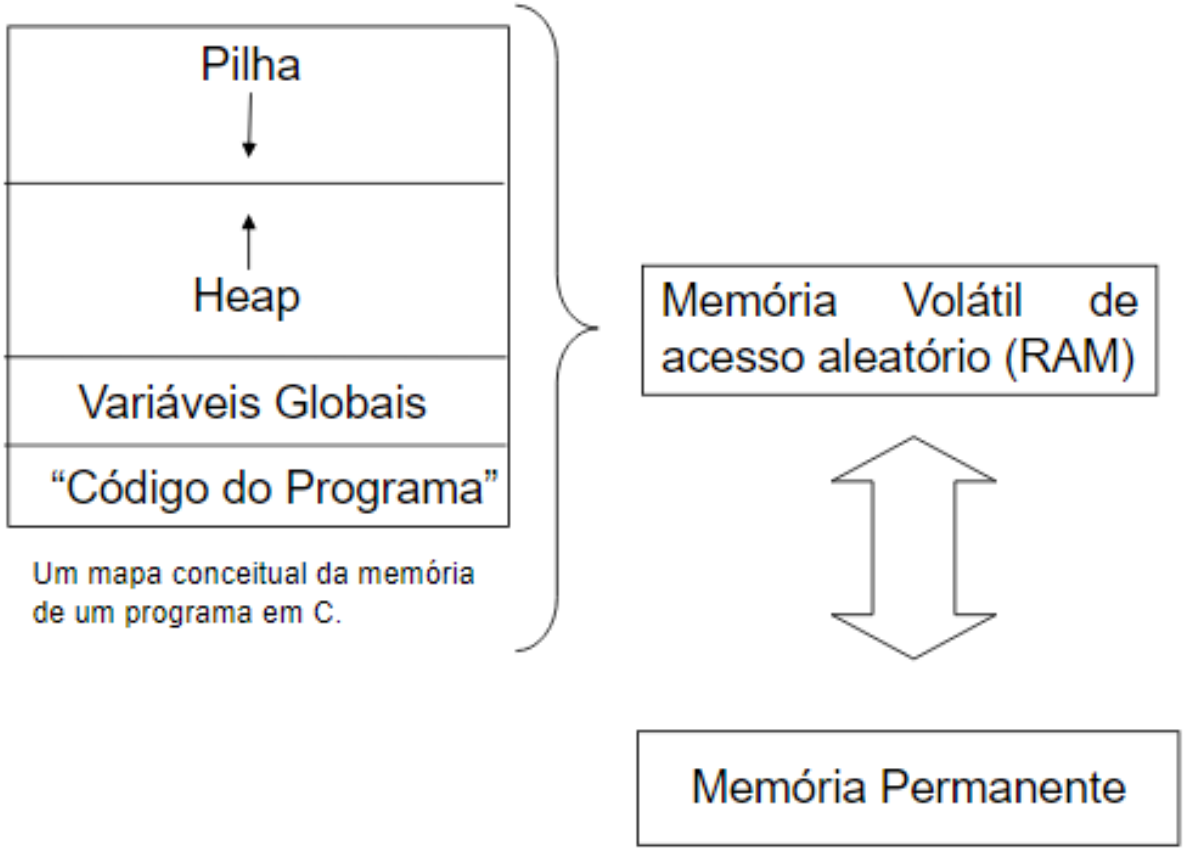
Alocação de Variáveis em C

- Variável Global
  - Alocada em tempo de compilação
- Variável Local
  - Alocada usando a pilha de execução do programa

Introdução


Variáveis Locais
Memória Livre
Variáveis Globais
Instruções de Programa
Variáveis Estáticas
Pilha
Heap
Área de Armazenamento Permanente

Introdução



Um mapa conceitual da memória de um programa em C.

- FONTE : http://slideplayer.com.br/slide/384227/

Introdução

- Veja os exemplos abaixo:
  - int i;
  - float k;
  - char c ;
  - int vet[30];

- Os códigos acima são declarações de variáveis
  - ALOCAM MEMÓRIA ESTATICAMENTE
    - No caso do vetor, a alocação estática ‘enxerga’ a variável como um único bloco, que ocupa posições contíguas na memória.

Estruturas de Dados

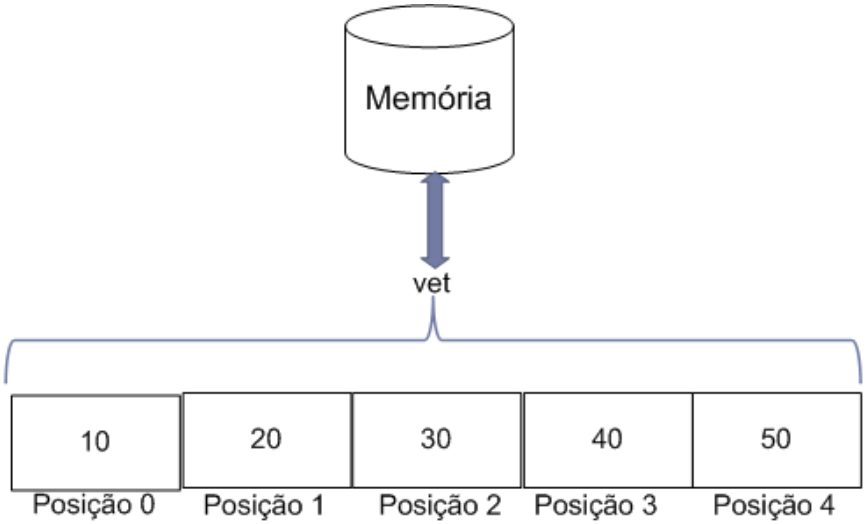
- Meio para armazenar e organizar dados na memória com o objetivo de facilitar o acesso e as modificações.
- As estruturas diferem umas das outras pela disposição ou manipulação de seus dados.
- Podem ser estáticas ou dinâmicas

Estruturas de Dados– Vetor Estático

- A forma mais simples de uma estrutura de dados é denominada **Vetor**, onde os elementos são armazenados na memória de forma indexada e contínua.

```
int main()
{
    int vet[5] = {10,20,30,40,50};
    return (0);
}
```

Vetor estático

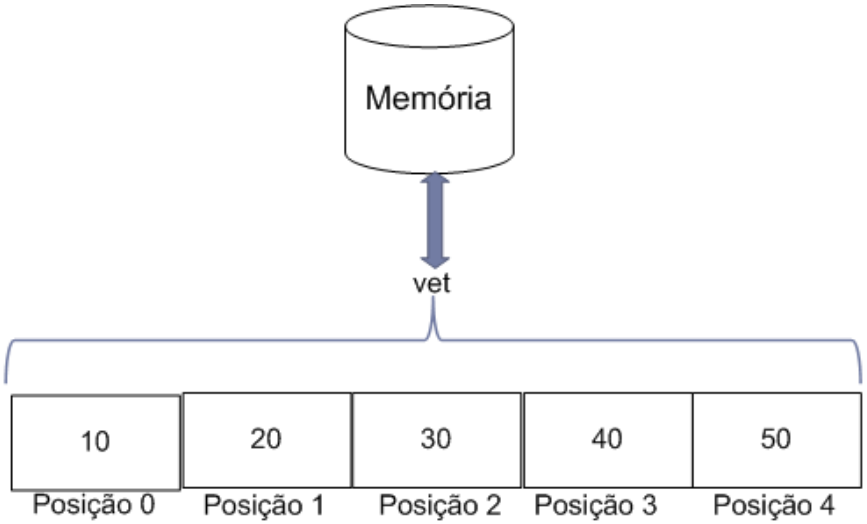


Estruturas de Dados– Vetor Estático

- A forma mais simples de uma estrutura de dados é denominada **Vetor**, onde os elementos são armazenados na memória de forma indexada e contínua.

```
int main()
{
    int vet[5] = {10,20,30,40,50};
    return (0);
}
```

Vetor estático



Posições contínuas de memória!!

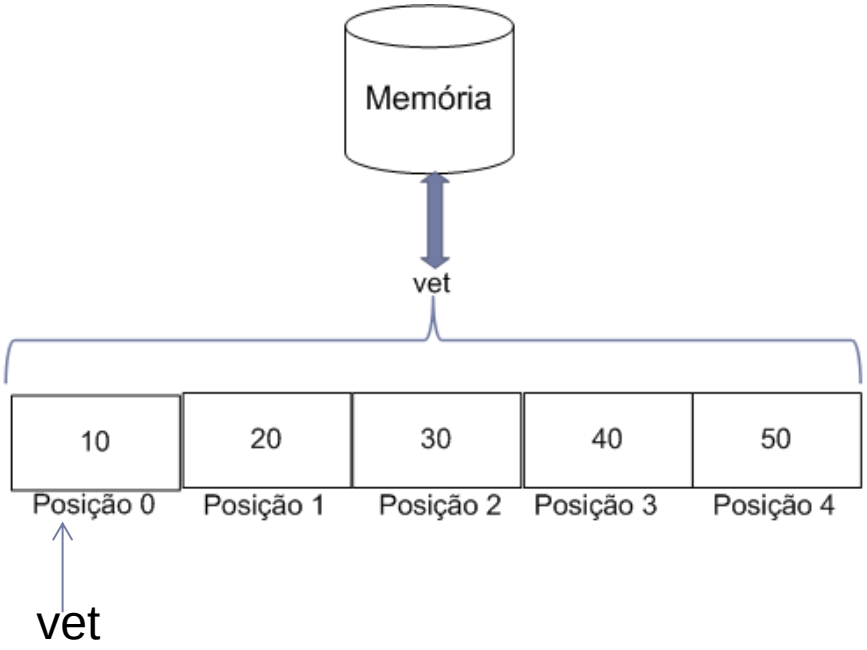
Seu tamanho e localização na memória não se alteram durante a execução

Estruturas de Dados– Vetor Estático

- A referência a uma posição de um vetor indica o cálculo de uma posição de memória a partir do início do vetor

```
int main()
{
    int vet[5] = {10,20,30,40,50};
    return (0);
}
```

vet[3] = vet + 3*sizeof(int)

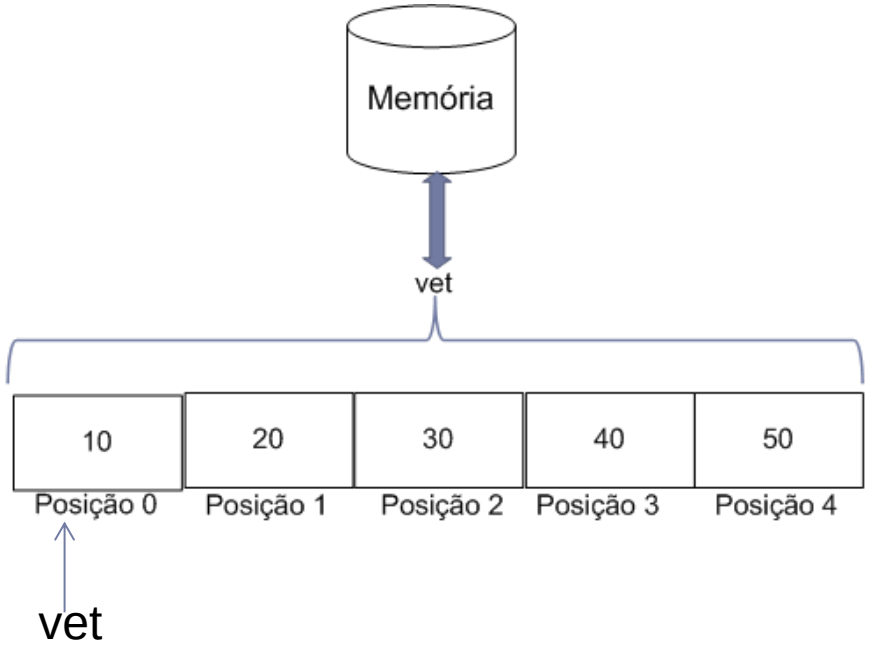


Estruturas de Dados– Vetor Estático

- A referência a uma posição de um vetor indica o cálculo de uma posição de memória a partir do início do vetor

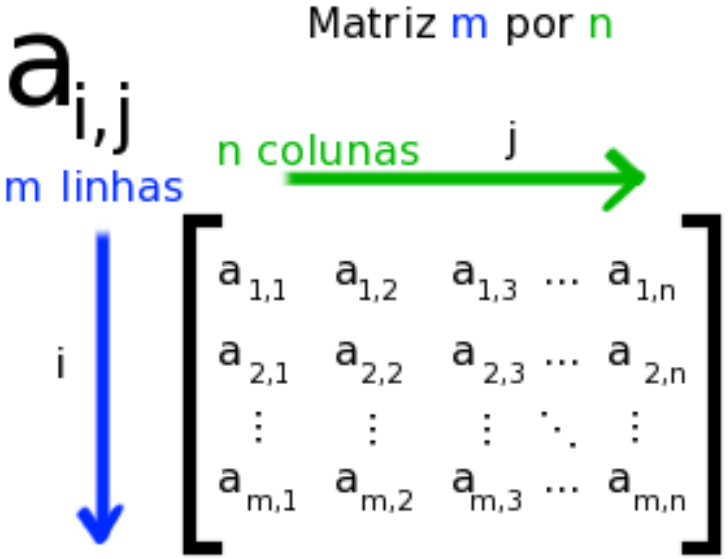
vet[3] = vet + 3*sizeof(int)

```c
int main()
{
    int vet[5] = {10,20,30,40};
    printf("%d ", vet[2]);
    printf("%d ", *(vet+2));
    return (0);
}
```



Estruturas de Dados– Matriz

- E uma matriz??



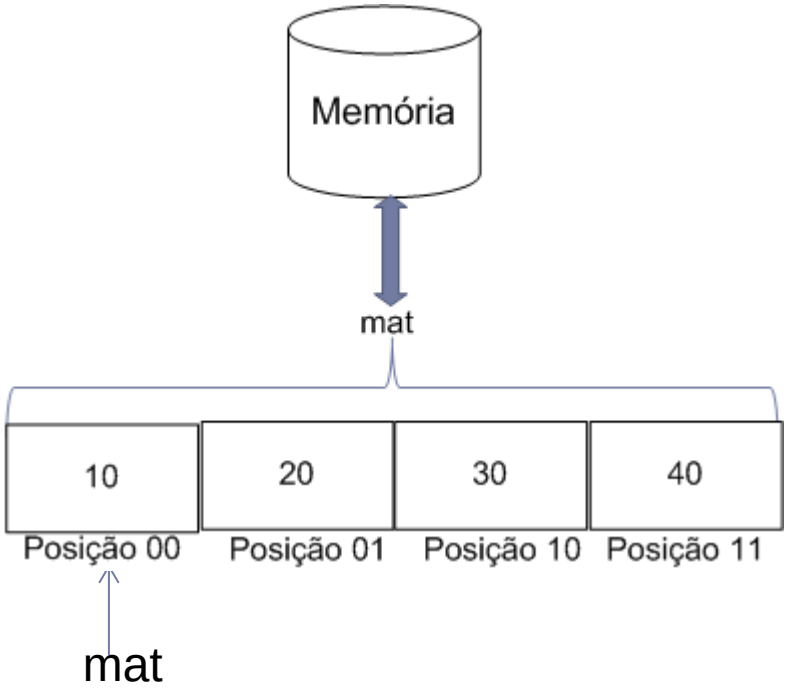
Estruturas de Dados– Matriz Estática

- E uma matriz??

```
int main()
{
    int mat[2][2] = {10,20,30,40};
    int i,j;
    return (0);
}
```

mat[l][c]= mat + (l*tamCol + coluna)*sizeof(int)

mat[1][0]= mat + (1*2 + 0)*sizeof(int)



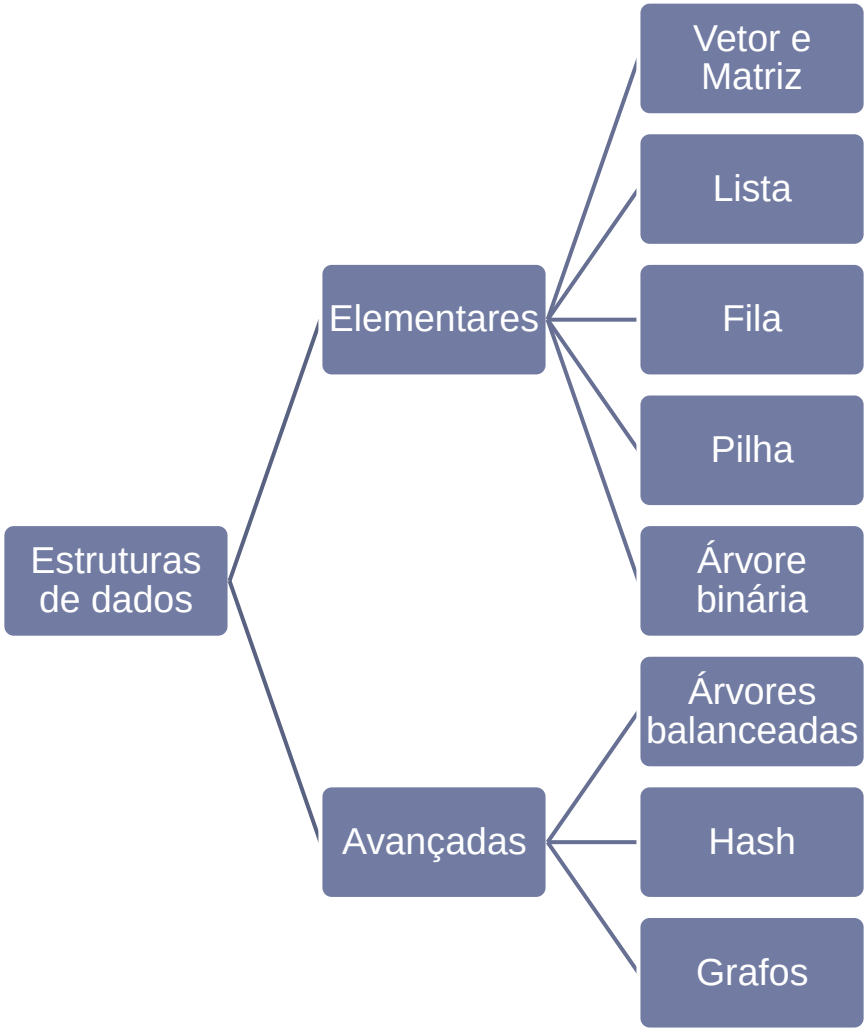
Estruturas de Dados

- Estruturas de dados estáticas são mais simples e fáceis de usar.

- Mas para a maioria das aplicações não se sabe a priori o número de elementos necessários na resolução do problema.

- Solução : ESTRUTURAS DE DADOS DINÂMICAS
  - a quantidade de elementos pode ser alterada sempre que necessário.

Estruturas de Dados Dinâmicas



Alocação Dinâmica de Memória

Alocação Dinâmica

- Alocação dinâmica é o meio pelo qual um programa pode obter memória enquanto está em execução.
  - Haverá momentos em que um programa precisará usar quantidades de armazenamento variáveis.
  - Ou seja, a quantidade de memória a alocar só se torna conhecida durante a *execução* do programa.
  - Para lidar com essa situação é preciso recorrer à alocação *dinâmica* de memória.

Alocação Dinâmica

- Exemplo:
  - Imagine um programa que tenha a função *mostrar_polinômio*, que recebe o nome do polinômio, o grau e seus coeficientes e imprime na tela o polinômio.

$$P(x)=a_n x^n+a_{(n-1)} x^{(n-1)}+...+a_2 x^2+a_1 x+a_0$$

  - **Entrada:**
    - Nome do Polinômio : P
    - Grau do Polinômio : 3
    - Coeficientes do Polinômio : 1 2 -3 4
  - **Saída:**
    - *P(x) = 1.0 + 2.0x – 3.0x^2 + 4x^3*

Alocação Dinâmica

- Exemplo:
  - Imagine um programa que tenha a função *mostrar_polinômio*, que recebe o nome do polinômio, o grau e seus coeficientes e imprime na tela o polinômio.

$$P(x)=a_n x^n+a_{(n-1)} x^{(n-1)}+...+a_2 x^2+a_1 x+a_0$$

  - **Entrada:**
    - Nome do Polinômio : Q
    - Grau do Polinômio : 6
    - Coeficientes do Polinômio : 1 2 -3 4 5 -6 7
  - **Saída:**
    - *Q(x) = 1.0 + 2.0x – 3.0x^2 + 4x^3 + 5x^4 -6x^5 + 7x^6*

Alocação Dinâmica

- Exemplo:
  - Imagine um programa que tenha a função *mostrar_polinômio*, que recebe o nome do polinômio, o grau e seus coeficientes e imprime na tela o polinômio.

  - *void mostrar_polinomio (int n, float *c, char p)*
    - *n é o grau do polinômio*
    - *c é o vetor de coeficientes*
    - *p é o nome do polinômio*

Alocação Dinâmica

- Exemplo:
  - Imagine um programa que tenha a função *mostrar_polinômio*, que recebe o nome do polinômio, o grau e seus coeficientes e imprime na tela o polinômio.

  - *void mostrar_polinomio (int n, float *c, char p)*
    - A cada execução do programa, o tamanho do vetor de coeficientes é diferente e depende do grau do polinômio, que será definido pelo usuário, em tempo de execução.
    - Portanto, o ideal é alocar esse vetor em tempo de execução, depois de conhecer o grau do polinômio.

Subsistema de Alocação Dinâmica

- A linguagem C oferece um subsistema para alocação dinâmica, cujas principais funções são:
  - calloc
  - malloc

  Aloca memória
  - free

  Libera memória
  - realloc

  Realoca a quantidade de memória alocada, para mais ou para menos

As funções pertencem a biblioteca stdlib.h

Subsistema de Alocação Dinâmica

- As funções de alocação de memória alocam a quantidade de **bytes** definida pelo programador e retornam um **ponteiro** para o início da memória alocada.
  - Se não for possível alocar a memória, as funções retornam NULL
- O endereço retornado pelas funções são do tipo *void* e, por isso, faz-se necessário realizar um *cast* para o tipo desejado.
- O espaço alocado dinamicamente permanece reservado até que explicitamente seja liberado pelo programa.
  - Se o programa não liberar um espaço alocado, este será automaticamente liberado quando a execução do programa terminar.

Subsistema de Alocação Dinâmica

- Função malloc
  - void *malloc(size_t size);
  - Aloca uma quantidade de memória igual a *size* bytes

```
char *p;
p = (char *) malloc(1000);
```

  - Após a atribuição, *p* aponta para o primeiro dos 1000 bytes de memória livre.

Subsistema de Alocação Dinâmica

- Função malloc
  - void *malloc(size_t size);

```
int *p;
p = (int *) malloc(50*sizeof(int));
```

    - Aloca espaço para 50 inteiros

- A função *sizeof* garante portabilidade
  - Compilador com int de 2 bytes
  - Compilador com int de 4 bytes

Subsistema de Alocação Dinâmica

- Função calloc
  - void *calloc(size_t num, size_t size);
  - Aloca uma quantidade de memória igual a *num*size* bytes

```
float *p;
p = (float *) calloc(50, sizeof(float));
```

  - Aloca espaço para 50 inteiros

Subsistema de Alocação Dinâmica

- As funções malloc e calloc retornam NULL caso não consigam alocar a memória por, por exemplo, não haver espaço no Heap.

- Por isso é importante validar a alocação antes de usar o ponteiro.

```
float *p;
p = (float *) calloc(50, sizeof(float));

if (!p)
{
    printf("Erro ao alocar a memória");
    exit(1);
}
```

Subsistema de Alocação Dinâmica

- As funções malloc e calloc retornam NULL caso não consigam alocar a memória por, por exemplo, não haver espaço na Heap.

- Por isso é importante validar a alocação antes de usar o ponteiro.

```
float *p;
p = (float *) malloc(50*sizeof(float));

if (p == NULL)
{
    printf("Erro ao alocar a memória");
    exit(1);
}
```

Subsistema de Alocação Dinâmica

- Os códigos são equivalentes!

```c
float *p;
p = (float *) malloc(50*sizeof(float));

if (p == NULL)
{
    printf("Erro ao alocar a memória");
    exit(1);
}
```

```c
float *p;
p = (float *) calloc(50, sizeof(float));

if (!p)
{
    printf("Erro ao alocar a memória");
    exit(1);
}
```

Subsistema de Alocação Dinâmica

- malloc x calloc
  - Ambas alocam memória e retornam um ponteiro para o início da memória alocada
  - A função calloc ‘inicializa’ a memória após sua alocação
    - Coloca zero em todas as posições de memória alocada
  - Caso a memória alocada com malloc seja acessada logo após sua alocação, o retorno será lixo de memória
  - Por inicializar a memória, a função calloc é mais lenta que a malloc

Alocação Dinâmica

- Exercício:
  - Implementar um programa para o cenário abaixo.

  - Imagine um programa que tenha a função *mostrar_polinômio*, que recebe o nome do polinômio, o grau e seus coeficientes e imprime na tela o polinômio.
    - *void mostrar_polinomio (int n, float *c, char p)*
    - **Entrada:**
      - Nome do Polinômio : P
      - Grau do Polinômio : 3
      - Coeficientes do Polinômio : 1 2 -3 4
    - **Saída:**
      - *P(x) = 1.0 + 2.0x – 3.0x^2 + 4x^3*

Alocação Dinâmica

- Exercício:
  - Implementar um programa que cria um vetor dinâmico com malloc.
    a) Testar os limites da memória heap
    b) Criar um vetor de 200 posições e imprimir na tela o valor de suas posições logo após a alocação.
    c) Repetir a letra b, alocando a memória com calloc.

Subsistema de Alocação Dinâmica

- A linguagem C oferece um subsistema para alocação dinâmica, cujas principais funções são:
  - calloc
  - malloc

  Aloca memória
  - free

  Libera memória
  - realloc

  Realoca a quantidade de memória alocada, para mais ou para menos

As funções pertencem a biblioteca stdlib.h

Subsistema de Alocação Dinâmica

- Função free
  - void *free(void *ptr);
  - Devolve ao heap a memória apontada por *ptr*, tornando a memória disponível para alocação futura.
  - A função free só deve ser usada com um ponteiro que foi previamente alocado com uma das funções do sistema de alocação dinâmica (malloc(), calloc() ou realloc()).

Subsistema de Alocação Dinâmica

- Função free
  - void *free(void *ptr);

```
float *p;
p = (float *) malloc(50*sizeof(float));

if (p == NULL)
{
     printf("Erro ao alocar a memória");
     exit(1);
}

free(p);
```

  - Libera na heap o espaço apontado por *p* (50*sizeof(float) bytes)

Subsistema de Alocação Dinâmica

- Função realloc
  - void *realloc(void *ptr, size_t size);
  - Modifica o tamanho da memória previamente alocada apontada por *ptr* para aquele especificado por *size*.
    - *size* pode ser maior ou menor que o original
    - Se *size* for zero, a memória é liberada
  - A função retorna um ponteiro para o bloco redimensionado de memória
    - Pode ser necessário copiar dados para outro bloco
    - Pode não haver memória necessária para a realocação e, neste caso, a função retorna nulo.

Subsistema de Alocação Dinâmica

- Função realloc
  - void *realloc(void *ptr, size_t size);

```
float *p;
p = (float *) malloc(50*sizeof(float));

if (p == NULL)
{
    printf("Erro ao alocar a memória");
    exit(1);
}

p = realloc(p, 1000*sizeof(float));
if (p == NULL)
{
    printf("Erro ao realocar a memória");
    exit(1);
}
free(p);
```

Alocação Dinâmica

- Exercício:
  - Implementar um programa que cria um vetor dinâmico com 5 posições.
    a) Inicializar os valores randomicamente.
       - v[i] = rand()%100;
    b) Realocar o vetor para 10 posições e inicializar os demais 5 valores
    c) Realocar o vetor para 3 posições
    d) Liberar memória
    e) Após cada alocação/inicialização, chamar a função imprime_vetor.

Alocação Dinâmica

- Exercício:

```
 Alocando vetor para 5 posicoes
41
67
34
0
69

 Realocando vetor para 10 posicoes
41
67
34
0
69
24
78
58
62
64

 Realocando vetor para 3 posicoes
41
67
34
```