Curso: Ciência da Computação

Disciplina: Estrutura de Dados 1

Professor: Clayton Zambon

centro universitário unifacvest

2. Introdução

2.1. Vetores

2. Introdução
2.1. Vetores

VETORES

- O vetor é uma estrutura de dados indexada, unidimensional, que pode armazenar uma determinada quantidade de valores do mesmo tipo.

Os dados armazenados em um vetor são chamados de itens do vetor.

- Para localizar a posição de um item em um vetor usamos um número inteiro denominado índice do vetor.

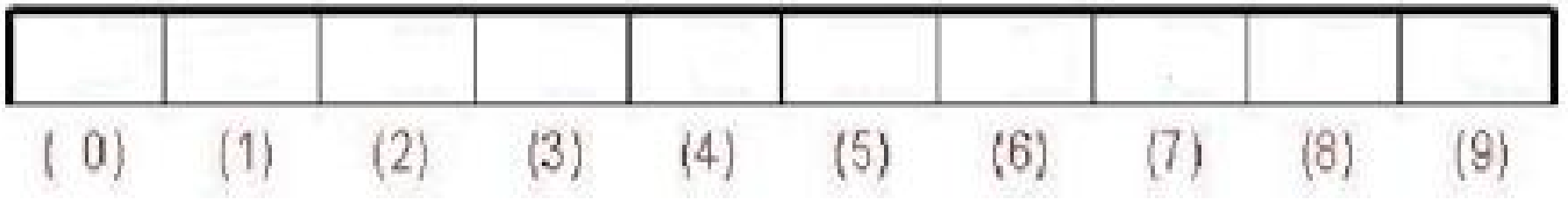
Vantagem de utilização do vetor

- Facilidade de manipular um grande conjunto de dados do mesmo tipo declarando-se apenas uma variável.

2. Introdução
2.1. Vetores

VETORES

Representação gráfica de um vetor.



2. Introdução
2.1. Vetores

VETORES

Vetor01.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
    //Declarando um vetor v[] de 10 posicoes e uma variável i
    int v[10], i;

    //Preenchendo valores nas posicoes do vetor
    for(i = 0; i < 10; i++)
        v[i] = i;

    //Exibindo o valor contido na posicao 2 do vetor
    printf("%d\n", v[2]);
    return 0;
}
```

O número 10 é a dimensão do Vetor

2. Introdução
2.1. Vetores

VETORES

Vetor01.c

```c
#include <stdio.h>


void f(int v[10])
{
    v[0] = 20;
}

int main(int argc, char** argv)
{
    //Declarando um vetor v[] de 10 posicoes e uma variável i
    int v[10], i;

    //Preenchendo valores nas posicoes do vetor
    for(i = 0; i < 10; i++)
        v[i] = i;

    //Exibindo todos os valores do vetor
    for(i = 0; i < 10; i++)
        printf("%d |", v[i]);


    return 0;
}
```

2. Introdução
2.1. Vetores

VETORES

Podemos somar e subtrair ponteiros veja:

*Vetor01.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
    //Declarando um vetor v[] de 10 posicoes e uma variável i
    int v[10], i;

    //Preenchendo valores nas posicoes do vetor
    for(i = 0; i < 10; i++)
        v[i] = i;

    //Exibindo todos os valores do vetor
    for(i = 0; i < 10; i++)
        printf("%d |", v + i );

    printf("\n\n");

    for(i = 0; i < 10; i++)
        printf("%d |", *(v + i) );

    return 0;
}
```

2. Introdução
2.1. Vetores

VETORES

Lembre-se:

Para trabalhar

Com o conteúdo

Eu utilizo o *

Asterisco.

Vetor01.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
    //Declarando um vetor v[] de 10 posicoes e uma variável i
    int v[10], i;

    //Preenchendo valores nas posicoes do vetor
    for(i = 0; i < 10; i++)
        v[i] = i;

    //Exibindo todos os valores do vetor
    //Exibe os enderecos de memoria de cada posicao do vetor
    printf("Enderecos de memoria de cada posicao do vetor\n");
    for(i = 0; i < 10; i++)
        printf("%d |", v + i );
    printf("\n\n");
    //Exibe os valores de cada posicao do vetor
    printf("Valores de cada posicao do vetor\n");
    for(i = 0; i < 10; i++)
        printf("%d |", *(v + i) );

    return 0;
}
```

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

- Veja que na declaração anterior do vetor, nós precisamos dimensionar previamente o vetor;
- Isso nos obriga a prever o número máximo do vetor;
- Este pré-dimensionamento é um fator limitante para nós. Imagine que no processo de desenvolvimento, nós não temos ideia ainda de quantos elementos precisaremos no vetor. Pode ter 10, 20, 5000.
- Você pode pensar: então faço a declaração de um vetor de 100 mil posições. Digamos que seu programa não chegue nem perto de 100 mil posições. Desta forma você estará subutilizando e desperdiçando a memória.
- Então se você coloca um valor alto, você pode não usar, se coloca um valor pequeno pode faltar posições;

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

- A linguagem C possui um meio de requisitar espaços de memória em tempo de execução. Por isso é chamado de alocação de memória dinâmica;
- Reservamos memória fazendo uma requisição em tempo de execução.
- Este espaço requisitado fica alocado dinamicamente até que seja explicitamente liberado pelo programa;
- E se eu não liberar explicitamente. O espaço de memória será liberado quando a execução do programa terminar.

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

- Existem funções na biblioteca **<stdlib.h>** que permitem alocar e liberar memória dinamicamente.
- A função mais básica quando formos ver sobre alocação dinâmica se chama **malloc;**
- **Malloc:** é um termo da computação que designa uma função da biblioteca padrão das linguagens C e C++ para requisitar alocação dinâmica de memória. É uma forma abreviada de escrever **Memory Allocation** (alocação de memória);

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

- Malloc: essa função malloc recebe como parâmetro o número de bytes que se deseja alocar e retorna o endereço inicial da área de memória alocada;
- Sizeof( ): Esta função retorno o tamanho em bytes do tipo desejado. Exemplo: sizeof(int) → irá retornar 4 bytes
- A função **Malloc** é utilizada para armazenar valores de qualquer tipo, inclusive de structs.

VETORES

- Alocação Dinâmica:

- Pelo fato da função Malloc armazenar valores de qualquer tipo, ela retorna um ponteiro genérico para um tipo qualquer representado por **Void *** que pode se convertido automaticamente pela linguagem para o tipo apropriado na atribuição.
- Então é comum ser feito a conversão de forma explícita utilizando um cast;

Exemplo: v = **(int*)**malloc(MAX * sizeof(int));

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

*VetorDinamico01.c

```c
#include <stdio.h>
#include <stdlib.h>
//Declaração de um valor global máximo
//Caso eu precise aumentar o valor altero somente aqui
#define MAX 10

int main(int argc, char** argv)
{
    //Declaramos um ponteiro inteiro porque a funcao malloc
    //tem como retorno um endereco inicial de memoria alocado
    int *v;

    //(MAX * sizeof(int)) = 10 * 4
    //Atribuindo ao poonteiro o endereco de memoria desejado
    //Funcao malloc com CAST de inteiro do tamanho MAX * sizeof(int)
    v = (int*)malloc(MAX * sizeof(int));

    return 0;
}
```

Cast para inteiro

Captura o tamanho do tipo int

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

- É interessante verificar se a alocação foi bem sucedida;
- Se for bem sucedida o **v** armazenará o endereço de memória contínua de memória;
- Senão houver espaço livre o suficiente para alocação irá retornar **NULL.** Este valor **null** está definido na biblioteca <stdlib.h>

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

*VetorDinamico01.c

```c
#include <stdio.h>
#include <stdlib.h>
//Declaração de um valor global máximo
//Caso eu precise aumentar o valor altero somente aqui
#define MAX 10

int main(int argc, char** argv)
{
    //Declaramos um ponteiro inteiro porque a funcao malloc
    //tem como retorno um endereco inicial de memoria alocado
    int *v;

    //(MAX * sizeof(int)) = 10 * 4
    //Atribuindo ao poonteiro o endereco de memoria desejado
    //Funcao malloc com CAST de inteiro do tamanho MAX * sizeof(int)
    v = (int*)malloc(MAX * sizeof(int));

    //Verifica se a alocacao de memoria foi bem sucedida
    //NULL e um valor contido na biblioteca <stdlib.h>
    if(v == NULL)
    {
        printf("Memória insuficiente\n");
        exit(1); //Aborta o programa e retorna 1 para o S.O
    }

    v[0] = 10;
    v[1] = 20;

    printf("%d\n", v[0]);
    return 0;
}
```

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

- **FREE:** utilizamos esta função para liberar o espaço de memória alocado dinamicamente;
- Esta função **FREE** recebe como parâmetro o ponteiro da memória a ser liberado que no exemplo será o **v.**
- Para um programa pequeno que acaba rápido não faz sentido, mas imagine a utilização dele em um programa grande.

  Exemplo: **free();**
- Após executar o FREE você não pode mais acessar aquele espaço de memória liberado;

2. Introdução
2.1. Vetores

VETORES

- Alocação Dinâmica:

*VetorDinamico01.c

```c
#include <stdio.h>
#include <stdlib.h>
//Declaração de um valor global máximo
//Caso eu precise aumentar o valor altero somente aqui
#define MAX 10

int main(int argc, char** argv)
{
    //Declaramos um ponteiro inteiro porque a funcao malloc
    //tem como retorno um endereco inicial de memoria alocado
    int *v;

    //(MAX * sizeof(int)) = 10 * 4
    //Atribuindo ao poonteiro o endereco de memoria desejado
    //Funcao malloc com CAST de inteiro do tamanho MAX * sizeof(int)
    v = (int*)malloc(MAX * sizeof(int));

    //Verifica se a alocacao de memoria foi bem sucedida
    //NULL e um valor contido na biblioteca <stdlib.h>
    if(v == NULL)
    {
        printf("Memória insuficiente\n");
        exit(1); //Aborta o programa e retorna 1 para o S.O
    }
    v[0] = 10;
    v[1] = 20;
    printf("%d\n", v[0]);

    //Libera o espaço de memoria que v estava ocupando
    free(v);
    return 0;
}
```

Referências

EDELWEISS, Nina; GALANTE, Renata. Estruturas de Dados. Porto Alegre, BOOKMAN, 2009.

HEINZLE, Roberto. Estruturas de Dados: implementações com C e Pascal. Blumenau, DIRETIVA, 2006.

TENENBAUM, Aron M. Estrutura de Dados usando C. São Paulo, Makron Books, 1995.

FORBELLONE, André Luiz Villar: EBERSPÄCHER, Henri Frederico. Lógica de Programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo, PRENTICE HALL, 2005.

KOFFMAN, Elliot B.; WOLFGANG, Paul A. T. Objetos, abstração, estruturas de dados e projeto usando C++. Rio de Janeiro, LTC, 2008.

PEREIRA, Silvio do lago. Estruturas de dados fundamentais: conceitos e aplicações. São Paulo, Érica, 1996.

VILLAS, Marcos Viana et al. Estruturas de dados – Conceitos e técnicas de implementação. Rio de Janeiro, Campus, 1993.

VELOSO, Paulo et al. Estrutura de dados. Rio de Janeiro, Campus, 1996.

Obrigado!