Algoritmos II

ALOCAÇÃO DINÂMICA
DE VETOR

Conceito

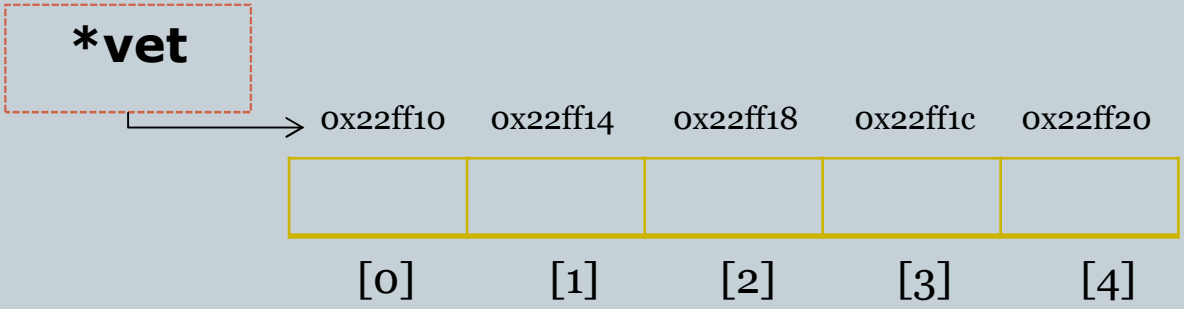
- Quando é necessário armazenar dados em memória sem que se conheça o tamanho ou a quantidade de dados necessários, pode-se recorrer ao recurso de alocação dinâmica de memória, que permite a alocação e liberação de áreas de memória a qualquer momento durante a execução do programa.

- O limite para alocação pode ser tão grande quanto a quantidade de memória física disponível no computador ou a quantidade de memória virtual.

Conceito

- Um **vetor dinamicamente alocado**, ou **vetor dinâmico**, é um vetor cujo tamanho não é especificado ao se escrever o programa, mas é determinado durante a execução do programa.
- Uma variável vetor nada mais é que uma espécie de variável ponteiro que aponta para a primeira variável indexada do vetor.

int vet[5];



Exemplo

- Veja o exemplo abaixo:

Tanto vet quanto ponteiro são variáveis do tipo ponteiro.

Aqui, a variável ponteiro passa a apontar para a posição para onde a variável vet está apontando.

Desta forma, pode-se utilizar a variável ponteiro para percorrer o vetor, pois ambas estão apontando para o mesmo endereço de memória.

```
#include <stdio.h>

int main() {
    int vet[10];
    int *ponteiro, i;
    ponteiro = vet;
    for (i=0;i<10;i++)
        ponteiro[i] = i * 2;
    for (i=0;i<10;i++)
        printf("%d\t",vet[i]);
    return 0;
}
```

Restrição

- **Restrição encontrada**: por exemplo, se fosse criada uma variável do tipo ponteiro chamada p2, e ela estivesse apontando para algum valor, não se pode fazer a seguinte atribuição:

```
vet = p2;
```

- Isso acontece porque uma variável vetor não é do tipo int*, mas sim uma versão const de int*. O valor da variável vet não pode ser alterado.

Conceito

- Até o momento, era necessário especificar o tamanho do vetor ao escrever o programa.
- Quando não se sabia o tamanho de vetor necessário estimava-se o tamanho maior possível, o que poderia gerar dois problemas: a estimativa poderia ser baixa, ou o programa ter muitas posições não utilizadas, acarretando em um desperdício de memória.
- Vetores dinâmicos evitam este problema, pois o usuário pode fornecer o tamanho do vetor ao iniciar a execução do programa.

Conceito

- Em C, a função malloc (memory allocation) e free são essenciais para a criação dos vetores dinâmicos, pois é com eles que se solicita ao sistema operacional a alocação (malloc) ou liberação (free) de memória. Estas funções alocam ou liberam memória para comportar o tipo de dado especificado.

- É necessário tomar cuidado ao utilizar esse tipo de recurso, para não alocar memória de forma indiscriminada sem a devida liberação, pois se isso ocorrer, a memória ficará cheia de áreas reservadas mas sem uso o que representa “lixo” e não deve ocorrer.

Exemplo

Aqui foi declarada uma variável ponteiro do tipo float e alocado um espaço de memória do tamanho especificado pelo usuário.

A variável ponteiro vet irá apontar para o primeiro endereço de memória alocado.

A variável ponteiro criada e o espaço alocado deverão ser do mesmo tipo.

Para utilização da função free(), simplesmente é passado o ponteiro que aponta para o início da memória alocada. O programa vai saber quantos bytes devem ser liberados, pois quando a memória foi alocada, ele guardou o número de bytes alocados em uma tabela de alocação interna.

```
#include <stdio.h>
#include <stdlib.h>

int main () {
   int TAM, i;
   printf("Tamanho: ");
   scanf("%d",&TAM);
   float *vet;
   vet = malloc(TAM * sizeof(float));
   for (i=0;i<TAM;i++)
      scanf("%f",&vet[i]);
   for (i=0;i<TAM;i++)
      printf("%.2f\t",vet[i]);
   free(vet);
   return 0;
}
```

Exemplo

```c
#include <stdio.h>
#include <stdlib.h>

int* dobro (int a[], int tam){
  int *temp = malloc(tam * sizeof(int));
  int i;
  for (i=0;i<tam;i++)
     temp[i] = 2 * a[i];
  return temp;
}
```

```c
int main () {
  int a[] = {1,2,3,4,5};
  int *b, i, tam = 5;

  b = dobro (a,tam);

  printf("Vetor A\n");
  for (i=0;i<5;i++)
    printf("%d\t",a[i]);
  printf("\n");

  printf("Vetor B\n");
  for (i=0;i<tam;i++)
     printf("%d\t",b[i]);
  free(b);
  return 0;
}
```

Algoritmos II

ARITMÉTICA DE PONTEIROS

Conceito

- São validas operações de soma e subtração.
- Quando se adiciona 1 a um ponteiro, o conteúdo é incrementado de um valor que corresponde à quantidade de bytes do tipo para o qual aponta. Por exemplo, ao se adicionar 1 a um ponteiro de int, o valor será incrementado de 4.
- A medida que se adiciona 1 a um ponteiro, pode-se acessar o próximo elemento de um vetor.
- O mesmo raciocínio se aplica a subtração.

Exemplo

```
#include <stdio.h>
#include <stdlib.h>

int main() {
	int vetor[]={10,20,30,40};
	int *Pvetor = vetor, i;

	for(i=0;i<4;i++)	//sem mudar o ponteiro
		printf("%d\n",*(Pvetor+i));

	for(i=0;i<4;i++)	//mudando o ponteiro
		printf("%d\n",*Pvetor++);

	return 0;
}
```

Conceito

- É uma aritmética de endereços, não de valores.
- É uma forma alternativa de se manipular vetores (e matrizes também).
- Não se pode executar multiplicação ou divisão de ponteiros. Só se pode acrescentar um inteiro a um ponteiro, subtrair um inteiro de um ponteiro ou subtrair dois ponteiros de mesmo tipo (eles devem apontar para o mesmo vetor).
- Ao se subtrair dois ponteiros, o resultado é o número de variáveis indexadas entre os dois endereços.