

Algoritmos II



ALOCAÇÃO DINÂMICA
DE VETOR EM C++

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 <iostream>
using namespace std;

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++)
        cout<<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



- Os operadores `new` e `delete` são essenciais para a criação dos vetores dinâmicos, pois é com eles que se solicita ao sistema operacional a alocação (`new`) ou liberação (`delete`) de memória. Estes operadores 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. Também poderia ter sido feito: `float vet = new float [TAM];`

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.

Deve-se colocar os `[]` antes do nome do vetor pois eles dizem ao C++ que uma variável dinamicamente alocada foi eliminada, e o sistema então verifica o tamanho do vetor e remove aquela quantidade de variáveis indexadas.

```
#include <iostream>
using namespace std;

int main () {
    int TAM, i;
    cout<<"Tamanho: ";
    cin>>TAM;
    float *vet;
    vet = new float[TAM];
    for (i=0;i<TAM;i++)
        cin>>vet[i];
    for (i=0;i<TAM;i++)
        cout<<vet[i]<<" ";
    delete []vet;
    return 0;
}
```


Exemplo



```
#include <iostream>
using namespace std;

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

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

    b = dobro (a,tam);
    cout<<"Vetor A"<<endl;
    for (i=0;i<5;i++)
        cout<<a[i]<<" ";
    cout<<endl;
    cout<<"Vetor B"<<endl;
    for (i=0;i<tam;i++)
        cout<<b[i]<<" ";
    delete []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 <iostream>
using namespace std;

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

    for(int i=0;i<4;i++)//sem mudar o ponteiro
        cout << *(Pvetor+i) << endl;

    for(int i=0;i<4;i++)//mudando o ponteiro
        cout << *Pvetor++ << endl;

    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.