Alocação Dinâmica de Memória

Há duas maneiras para alocar memória para armazenar dados

1. Alocação estática de memória
- Memória para variáveis nomeadas é alocada pelo compilador
- O tamanho e tipo exato dos dados a serem armazenados devem ser conhecidos em tempo de compilação

2. Alocação dinâmica de memória
- Memória alocada em tempo de execução
- A quantidade exata de espaço ou o número de itens não precisam ser conhecidos pelo compilador
- O espaço de memória alocado na heap

Alocação Dinâmica de Memória

A alocação dinâmica de memória implica na distribuição da memória quando necessário e no seu restabelecimento após a sua utilização, permitindo a criação e liberação de elementos dinamicamente, o que possibilita a construção de determinadas estruturas de dados – tais como listas, filas, pilhas e árvores – que, se implementadas com estruturas estáticas, certamente seriam menos eficientes e mais complexas.

Alocação Dinâmica de Memória

Podemos alocar espaço de armazenamento dinamicamente enquanto o programa está sendo executado, mas não podemos criar novos nomes de variáveis durante a execução.

Por essa razão, a alocação dinâmica de memória requer:

1. Criar o espaço de memória dinamicamente.
2. Armazenar o endereço desse espaço em um ponteiro, de modo que possa ser futuramente acessado.

A alocação dinâmica de memória é feita através da utilização de variáveis apontadoras, as quais contém a localização (endereço) da informação armazenada em memória primária.

variável apontadora = apontador = ponteiro

Alocação Dinâmica de Memória

Alocando espaço com **`new`**

Para alocar memória dinamicamente utilizamos o operador unário **`new`**, seguido do tipo de dados a ser alocado.

```
new int;          // aloca dinamicamente um int
new double;       // aloca dinamicamente um double

new int[40];      // aloca dinamicamente um array de 40 ints
new double[TAM];
                  // aloca dinamicamente um array de size doubles
                  //  note que TAM pode ser uma variável
```

Alocação Dinâmica de Memória

Esses comandos não são muito úteis porque não há nomes para as áreas de memória alocadas, MAS o operador **`new`** retorna o endereço inicial da área de memória alocada e esse endereço pode ser armazenado em um ponteiro.

```
int *p;   // declara um ponteiro p
p = new int;
          // aloca dinamicamente um int e armazena o endereço em p

double *d;   // declara um ponteiro d
d = new double;
        // aloca dinamicamente um double e armazena o endereço em d

// podemos fazer também em uma única linha de comando
 int x = 40;
 int *V = new int[x];
 float *numeros = new float[x+10];
```

Alocação Dinâmica de Memória

Declaração de variável do tipo ponteiro

// declaração de variável
tipo *identificador;

ou

```
int *p;
```

// declaração de um tipo
typedef
tipo *identificador ;

ou

```
typedef
    int *ptINTEIRO;

// declaração de variável
    ptINTEIRO p;
```

Uma variável do tipo ponteiro contém o endereço de memória de uma **variável dinâmica**, onde é armazenada a informação

p é uma variável do tipo ptINTEIRO que aponta para uma área de memória do tipo int

OU

p é uma variável do tipo ptINTEIRO que endereça área de memória onde será armazenado um valor do tipo int

Alocação Dinâmica de Memória

Acessando a área de memória dinamicamente alocada

```
int *p = new int;
      // aloca dinamicamente espaço em memória para armazenar um inteiro,
      referenciado por p

*p = 10;        // armazena 10 na área de memória referenciada por p
cout << *p;     // escreve 10

double *numeros = new double[TAM];
                // array alocado dinamicamente em memória

for (int i = 0; i < TAM; i++)
    numeros[i] = 0;       // inicializa o array com 0

 numeros[5] = 20;         // notação de colchetes
 *(numeros + 7) = 15;     // notação de offset de ponteiro
                          //  corresponde a numeros[7]
```

Alocação Dinâmica de Memória

Conteúdo do endereço de uma variável do tipo ponteiro

***identificador da variável**

`p`	endereço de memória
`*p`	conteúdo da área de memória endereçada por p

Alocação Dinâmica de Memória

Liberando espaço com **`delete`**

Para dealocar (liberar) memória dinamicamente utilizamos o operador unário **`delete`**, seguido do ponteiro que referencia a área de memória a ser liberada.

```
int *ptr = new int;    // aloca dinamicamente memória para int
// ...
delete ptr;            // libera a área de memória referenciada por ptr
```

Note que o ponteiro ptr continua existindo, podendo passar a referenciar outra área de memória

```
ptr = new int[10];     // ptr aponta para um array de 10 inteiros
```

Alocação Dinâmica de Memória

Liberando espaço com **`delete`**

No caso de um array

`delete [ ] identificador;`

```
int *V = new int[40]; // array alocado dinamicamente

delete [ ] V; // libera área de memória alocada para o array
```

Alocação Dinâmica de Memória

```
int *ptr;   //declaração do ponteiro
ptr = new int[5];  //inicialização após declaração
```

```
int *ptr1 = new int[5]; //inicialização na declaração
```

```
int *ptr = new int[3];
ptr[0] = 1;
ptr[1] = 2;
ptr[2] = 3;
cout << ptr[0] << " " << ptr[1] << " " << ptr[2];
```

Alocação Dinâmica de Memória

```
int *ptr;    //declaração do ponteiro
ptr = new int[5];  //inicialização após declaração
int *ptr1 = new int[5]; //inicialização na declaração

delete []ptr;
```

```
int *ptr = new int[3];
ptr[0] = 1;
ptr[1] = 2;
ptr[2] = 3;
cout << ptr[0] << " " << ptr[1] << " " << ptr[2];
delete []ptr;

delete []ptr1;
```

Alocação Dinâmica de Memória

Inicializando ponteiros

Um ponteiro pode ser inicializado com 0, NULL ou com um endereço.

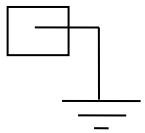
Um ponteiro inicializado com 0 ou NULL não aponta para área de memória nenhuma e é conhecido como ponteiro nulo.

A constante simbólica NULL é definida no arquivo de cabeçalho <iostream>

identificador da variável = NULL; // valor nulo de endereço

```
p = NULL;
```

graficamente



Alocação Dinâmica de Memória

Operador de endereço (&)

O operador de endereço (&) é um operador unário que retorna o endereço de memória de seu operando.

```
int y = 5;   // declara a variável y e atribui 5
int *yPtr;   // declara variável ponteiro yPtr

yPtr = &y;    // atribui o endereço de y a yPtr;
```

Alocação Dinâmica de Memória

```
// Pointer operators & and *

#include <iostream>

using namespace std;


int main() {

   int a = 7 // initialize a with 7

   int* aPtr = &a; // initialize aPtr with the address of int variable a


   cout << "The address of a is " << &a

      << "\nThe value of aPtr is " << aPtr;

   cout << "\n\nThe value of a is " << a

      << "\nThe value of *aPtr is " << *aPtr << endl;

}
```

Alocação Dinâmica de Memória

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

int cubeByValue(int); // prototype

int main() {
   int number = 5;

   cout << "The original value of number is " << number;

   number = cubeByValue(number); // pass number by value to cubeByValue

   cout << "\nThe new value of number is " << number << endl;

}

// calculate and return cube of integer argument

int cubeByValue(int n) {

   return n * n * n; // cube local variable n and return result

}
```

Alocação Dinâmica de Memória

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

void cubeByReference(int*); // prototype

int main() {

   int number = 5;

   cout << "The original value of number is " << number;

   cubeByReference(&number); // pass number address to cubeByReference

   cout << "\nThe new value of number is " << number << endl;

}

// calculate cube of *nPtr; modifies variable number in main

void cubeByReference(int* nPtr) {

   *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr

}
```

Alocação Dinâmica de Memória

Aritmética de ponteiros

Quando é adicionado 1 a um ponteiro, o seu conteúdo é incrementado de um valor correspondente à quantidade de bytes do tipo para o qual o ponteiro aponta. Raciocínio análogo se aplica à subtração de um valor.

```
int k; int *ptr = new int[3];
for (k=0; k < 3; k++)
   ptr[k] = k;
cout << *ptr << "  ";
ptr++;
cout << *ptr << "  ";
ptr++;
cout << *ptr << "  ";
delete []ptr;
```

Alocação Dinâmica de Memória

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

void exibe(int *);

int main(){
     int *ptr = new int[3];
     ptr[0] = 10;    ptr[1] = 20;    ptr[2] = 30;
     cout << "no main = ";
     cout << ptr[0] << "  "  << ptr[1] << "  "  << ptr[2]  << endl;
     exibe(ptr);
     delete [] ptr;
     return 0;
}

void exibe(int* V){
     cout << "na funcao exibe = ";
     cout << V[0] << "  "  << V[1] << "  "  << V[2]  << endl;
}
```

Alocação Dinâmica de Memória

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

void exibe(int *);

int main(){
     int *ptr = new int[3];
     ptr[0] = 10;    ptr[1] = 20;    ptr[2] = 30;
     cout << "no main = ";
     cout << ptr[0] << "  "  << ptr[1] << "  "  << ptr[2]  << endl;
     exibe(ptr);
     cout << ptr[0] << "  "  << ptr[1] << "  "  << ptr[2]  << endl;
     delete [] ptr;
     return 0;
}

void exibe(int* V){
     cout << "na funcao exibe = ";
     V[0]++;
     cout << V[0] << "  "  << V[1] << "  "  << V[2]  << endl;
}
```

Alocação Dinâmica de Memória

Ponteiros e CONST

1. Protegendo o que o ponteiro aponta

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

int main(){
     int x = 10, y = 20;
     const int *p = &x;  //ponteiro para um int const
     cout << *p << endl;
     p = &y;
     cout << *p << endl;
     *p = 99;  // ERRO: alteração do conteúdo do endereço em p
     return 0;
}
```

Alocação Dinâmica de Memória

Ponteiros e CONST

2. Protegendo o ponteiro

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

int main(){
     int x = 10, y = 20;
     int *const p = &x;  //ponteiro const para um int
     cout << *"Endereco = " << p  << " conteudo = " << *p << endl;
     *p = 99;
     cout << *"Endereco = " << p  << " conteudo = " << *p << endl;
     p = &y;  // ERRO: alteração do endereço contido em p
     return 0;
}
```

Alocação Dinâmica de Memória

Ponteiros e CONST

3. Protegendo o que o ponteiro aponta e o ponteiro

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

int main(){
     int x = 10, y = 20;
     const int *const p = &x;
     cout << "Endereco = " << p  << " conteudo = " << *p << endl;
     *p = 99; // ERRO: alteração do conteúdo do endereço contido em p
     p = &y;  // ERRO: alteração do endereço contido em p
     return 0;
}
```

Alocação Dinâmica de Memória

Ponteiros para ponteiros

```
#include <iostream>
using namespace std;
int main()
{
     int a=10;
     int *ptr;        // ponteiro de inteiro
     int **ptrPtr;    // ponteiro de um ponteiro inteiro
     ptr = &a;
     ptrPtr = &ptr;
     cout << "O valor final de ptrPtr eh " << **ptrPtr << endl;
     return 0;
}
```

Alocação Dinâmica de Memória

Ponteiros para ponteiros

```
#include <iostream>
using namespace std;
int main()
{
     int a=10;
     int *ptr;        // ponteiro de inteiro
     int **ptrPtr;    // ponteiro de um ponteiro inteiro
     ptr = &a;
     ptrPtr = &ptr;
     cout << "O valor final de ptrPtr eh " << **ptrPtr << endl;
     return 0;
}
```

Saída: O valor final de ptrPtr eh 10

Alocação Dinâmica de Memória

Ponteiros para ponteiros

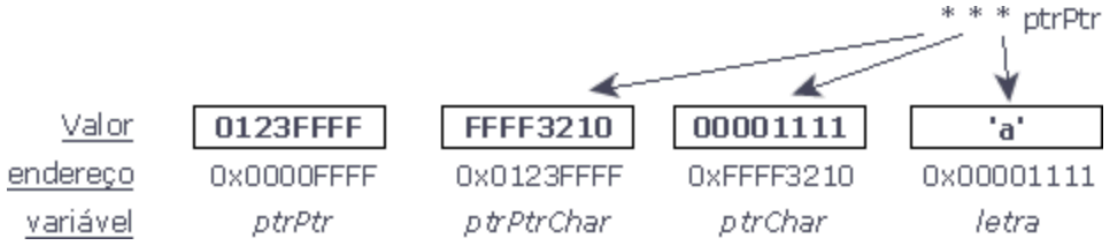
```
#include <iostream>

using namespace std;

int main()
{
    char letra ='a';
    char *ptrChar;
    char **ptrPtrChar;
    char ***ptrPtr;
    ptrChar = &letra;
    ptrPtrChar = &ptrChar;
    ptrPtr = &ptrPtrChar;
    cout << "O valor final de ptrPtr eh " << ***ptrPtr << endl;
    return 0;
}
```

Alocação Dinâmica de Memória

Ponteiros para ponteiros




```
#include <iostream>

using namespace std;

int main()
{
    char letra =’a';
    char *ptrChar;
    char **ptrPtrChar;
    char ***ptrPtr;
    ptrChar = &letra;
    ptrPtrChar = &ptrChar;
    ptrPtr = &ptrPtrChar;
    cout << "O valor final de ptrPtr eh " << ***ptrPtr << endl;
    return 0;
}
```

Saída: O valor final de ptrPtr eh a

Alocação Dinâmica de Memória

Ponteiros para ponteiros

```
#include <iostream>

using namespace std;

int main()
{
    float var1 = 25.5;
    float var2 = 72.8;
    float *ptr;
    float **ptrPtr;
    ptr = &var1;
    ptrPtr = &ptr;
    cout << "O valor final de ptrPtr eh " << **ptrPtr << endl;
    cout << "O valor de ptr eh " << *ptr << endl;
    cout << "O conteudo de ptr eh" << ptr << endl;
    cout << "O endereco de ptr eh" << &ptr << endl;
    *ptrPtr = &var2; //novo enderco no ponteiro intermediario
    cout << "O valor final de ptrPtr eh " << **ptrPtr << endl;
    cout << "O valor de ptr eh " << *ptr << endl;
    cout << "O conteudo de ptr eh " << ptr << endl;
    cout << "O endereco de ptr eh" << &ptr << endl;
    return 0;
}
```

Alocação Dinâmica de Memória

Ponteiros para ponteiros

```
#include <iostream>

using namespace std;

int main()
{
    float var1 = 25.5;
    float var2 = 72.8;
    float *ptr;
    float **ptrPtr;
    ptr = &var1;
    ptrPtr = &ptr;
    cout << "O valor final de ptrPtr eh " << **ptrPtr << endl;
    cout << "O valor de ptr eh " << *ptr << endl;
    cout << "O conteudo de ptr eh" << ptr << endl;
    cout << "O endereco de ptr eh" << &ptr << endl;
    *ptrPtr = &var2; //novo enderco no ponteiro intermediario
    cout << "O valor final de ptrPtr eh " << **ptrPtr << endl;
    cout << "O valor de ptr eh " << *ptr << endl;
    cout << "O conteudo de ptr eh " << ptr << endl;
    cout << "O endereco de ptr eh" << &ptr << endl;
    return 0;
}
```

Saída

```
O valor final de ptrPtr eh 25.5
O valor de ptr eh 25.5
O conteudo de ptr eh0x7fff5245a98c
O endereco de ptr eh0x7fff5245a980
O valor final de ptrPtr eh 72.8
O valor de ptr eh 72.8
O conteudo de ptr eh 0x7fff5245a988
O endereco de ptr eh0x7fff5245a980
```