INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

ALGORITMOS II

Prof. Adilso Nunes de Souza

CONCEITO

- Alocação dinâmica de memória é um recurso, existente nas linguagens de programação, onde podemos deixar de reservar memória quando o programa ainda está na fase de desenvolvimento, quando não sabemos ainda de quanta memória o programa irá precisar para executar algum procedimento, e o fazemos durante a execução.

ALOCAÇÃO DINÂMICA E PONTEIRO

- O compilador reserva espaço na memória para todos os dados declarados explicitamente, mas se usarmos ponteiros precisamos colocar no ponteiro um endereço de memória existente. Para isto, podemos usar o endereço de uma variável definida previamente ou reservar o espaço necessário no momento que precisemos. Este espaço que precisamos reservar em tempo de execução é chamada de memória alocada dinamicamente.

OPERADOR NEW

- O operador “new” é utilizado na linguagem C++ para retorna o endereço onde começa o bloco de memória que foi reservado. Como retorna um endereço podemos colocá-lo num ponteiro. Assim, teremos um meio de manipular o conteúdo da memória alocada toda vez que mencionarmos o ponteiro, sem necessidade de uma variável anterior.

EXEMPLO

```
//Definindo o ponteiro e alocando a memória
int *px;
px = new int;
//Também pode ser declarado desta forma
int *px = new int;
//Alocando ó espaço e inicializando com o valor 4
int *px = new int(4);
//Agora com caracter
char *c = new char('A');
```

OPERADOR DELETE

- O operador delete não apaga o ponteiro mas sim a memória para onde o ponteiro aponta. Na verdade, a memória não é de fato apagada, ela retorna ao estado de disponível como memória livre para ser alocada novamente quando for necessário.
- Exemplo espaço comum:

```
delete px;
```

- Exemplo array:

```
delete [] pvet;
```

EXEMPLO ARRAY

```
//Alocando memória dinamicamente para um array
   int *px = new int[5];
   srand(time(NULL));
   for(int x = 0; x < 5; x++)
   {
      px[x] = rand() % 50;
   }
//ou desta forma
   for(int i = 0; i < 5; i++)
   {
      *(px + i) = rand() % 50;
   }
```

```
//ou usando aritmética de ponteiro
   for(int i = 0; i < 5; i++)
   {
      *pvet = rand() % 50;
      pvet++;
   }
```

EXEMPLO ARRAY TAMANHO VARIÁVEL

```
//criando um array de tamanho dinâmico.
  int x;
  cout << "Informe o tamanho do vetor: ";
  cin >> x;
  fflush(stdin);
  int *px = new int[x];

  srand(time(NULL));
  for(int i = 0; i < x; i++)
  {
     px[i] = rand() % 50;
  }
```

EXEMPLO ARRAY DE DUAS DIMENSÕES

- É possível manipular um array multidimensional (matriz) com o operador "**new**", mas somente a primeira dimensão pode ser definida em tempo de execução, as demais devem ser constantes.

EXEMPLO ARRAY DE DUAS DIMENSÕES

```
int n;
int (*notas)[3]; //declarando o ponteiro
cout << "Informe o numero de linhas: ";
cin >> n;
fflush(stdin);
notas = new int[n][3];
for(int i = 0; i < n; i++)
{
    for(int j = 0; j < 3; j++)
    {
        cin >> notas[i][j];
    }
}
```

EXEMPLO ARRAY DE DUAS DIMENSÕES

- Para manipular a matriz em uma função:

```
void mostra(int (*notas)[3], int n)
{
   for(int i = 0; i < n; i++)
   {
      for(int j = 0; j < 3; j++)
      {
         cout << notas[i][j] << "\t";
      }
      cout << "\n";
   }
}
```

EXEMPLO ARRAY DE DUAS DIMENSÕES

- Caso o desenvolvedor não tenha conhecimento prévio de nenhuma das dimensões da matriz, o que pode ser feito?
- Como já é de conhecimento que se tratando de array a memória é alocada sequencialmente, é possível utilizar um array de uma dimensão para trabalhar a quantidade de elementos da matriz.

EXEMPLO ARRAY DE DUAS DIMENSÕES

```
int l, c;
cout << "Informe o numero de linhas: ";
cin >> l;
fflush(stdin);

cout << "Informe o numero de colunas: ";
cin >> c;
fflush(stdin);

int *mat = new int[l * c];
int i;
srand(time(NULL));
for(i = 0; i < (l * c); i++)
{
   mat[i] = rand() % 500;
}
```

REFERÊNCIAS

- SCHILDT, Herbert. C++: Guia para iniciantes. Rio de Janeiro: Editora Ciência Moderna LTDA. 2002.
- PEREIRA, Silvio do Lago. Estrutura de Dados Fundamentais: Conceitos e Aplicações, 12. Ed. São Paulo, Érica, 2008.
- LORENZI, Fabiana. MATTOS, Patrícia Noll de. CARVALHO, Tanisi Pereira de. Estrutura de Dados. São Paulo: Ed. Thomson Learning, 2007.
- VELOSO, Paulo. SANTOS, Celso dos. AZEVEDO, Paulo. FURTADO, Antonio. Estrutura de dados. Rio de Janeiro: Ed. Elsevier, 1983 27ª reimpressão.