ALOCAÇÃO DINÂMICA DE MEMÓRIA

A linguagem C/C++ possui recursos para alocação dinâmica de memória.

As funções que trabalham com alocação de memória se encontram na biblioteca **<stdlib.h>**

void *calloc(int num, int size);	aloca a memória para um vetor de **num** elementos, com tamanho **size** de cada um deles
void free(void *address);	libera a memória especificada por **address**
void *malloc(int num);	aloca a memória para **num** bytes
void *realloc(void *address, int newsize);	redimensiona a memória alocada para **adress** par novo tamanho **newsize**

As funções **calloc()** e **malloc()** procuram por uma área livre (de tamanho desejado) na memória operativa e retornam o endereço do início dessa área.

O tipo de retorno dessas funções é **void *** – é um ponteiro para um tipo vazio, por isso é recomendada a conversão de tipos explicita.

Para definir o tamanho normalmente é utilizada a função **sizeof(dataType)**, onde **dataType** – é o tipo de dados em questão.

A memória alocada dinamicamente será liberada de forma automática ao sair do programa, mas é considerado como boa prática de programação liberar a memória alocada dinamicamente usando a função **free(aPtr)**, onde o **aPtr** é o ponteiro para memória que foi alocada pelo programa.

VETORES

Para criar um vetor de tamanho definido pelo usuário basta alocar uma área continua de memória de tamanho apropriado.

Um caso de alocação dinâmica para vetor de tamanho variado é mostrado no exemplo a seguir.

Exemplo 1: Vetor com tamanho definido pelo usuário

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

int main()
{
   int *vPtr; // pointer
   int arraySize;
   int i;

   printf("\n Input array size : ");
   scanf("%d",&arraySize);

   // memory allocation
   vPtr = (int*) malloc(arraySize*sizeof(int));

   // data input
   printf("\n Data input : \n");
   for(i=0; i < arraySize; i++)
   {
      printf("v[%d] = ", i);
      scanf("%d", &vPtr[i]);
   }

   // data output
   printf("\n\n Data output : \n");
   for(i=0; i<arraySize; i++)
      printf("v[%d] = %d \n",i, vPtr[i]);

 return 0;
}
```

```
 Input array size : 2

 Data input :
v[0] = 1
v[1] = 2


 Data output :
v[0] = 1
v[1] = 2
```

Para acessar os elementos do vetor alocado dinamicamente podemos também trabalhar com a aritmética dos ponteiros.

Exemplo 2: Vetor com tamanho definido pelo usuário (ponteiros)

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

int main()
{
   int *vPtr;    // pointer
   int *auxPtr;  // pointer
   int arraySize;
   int i;

   printf("\n Input array size : ");
   scanf("%d",&arraySize);

   // memory allocation
   vPtr = (int*) malloc(arraySize*sizeof(int));

   // data input
   auxPtr = vPtr;
   printf("\n Data input : \n");
   for(i=0; i < arraySize; i++)
   {
      printf("v[%d] = ", i);
      scanf("%d", auxPtr);
      auxPtr++;
   }

   // data output
   auxPtr = vPtr;
   printf("\n\n Data output : \n");
   for(i=0; i<arraySize; i++)
   {
      printf("v[%d] = %d \n",i, *auxPtr);
      auxPtr++;
   }

 return 0;
}
```

```
 Input array size : 2

 Data input :
v[0] = 1
v[1] = 2

 Data output :
v[0] = 1
v[1] = 2
```

De forma geral, sempre é bom verificar se a operação de alocação da memória foi bem-sucedida.

Isso pode ser feito como mostra o exemplo a seguir.

Exemplo 3: Vetor alocado dinamicamente com teste de sucesso

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

int main()
{
   int *vPtr; // pointer
   int arraySize;
   int i;

   printf("\n Input array size : ");
   scanf("%d",&arraySize);

   // memory allocation
   vPtr = (int*) malloc(arraySize*sizeof(int));

   if( vPtr == NULL )
   {
      printf("\n Memory allocation error! \n");
   }
   else
   {
     // data input
     printf("\n Data input : \n");
     for(i=0; i < arraySize; i++)
     {
        printf("v[%d] = ", i);
        scanf("%d", &vPtr[i]);
     }

     // data output
     printf("\n\n Data output : \n");
     for(i=0; i<arraySize; i++)
        printf("v[%d] = %d \n",i, vPtr[i]);
    }

 free(vPtr);
 return 0;
}
```

```
Input array size : -1

 Memory allocation error!
```

MATRIZES

Para criar uma matriz alocada dinamicamente (com tamanho definido pelo usuário) a princípio temos duas opções.

1. Alocação de vetor n*m

Vamos considerar o seguinte caso: queremos criar dinamicamente uma matriz com **n** linhas e **m** colunas.

De forma geral, qualquer matriz é armazenada em uma área continua de memória.

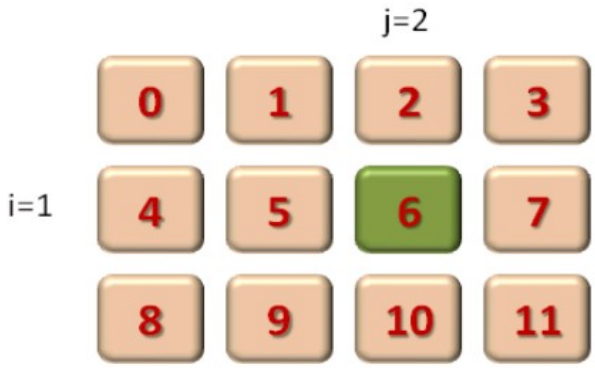
Sendo assim, caso queremos procurar por um elemento com índices [i] [j], podemos fazer isso de acordo com a fórmula:

```
index = i * m + j;
```

onde

i – é o número da linha interesse,
j – é o número da coluna de interesse
m – é a quantidade das colunas

Por exemplo, caso temos uma matriz 3 x 4:



elemento [1][2], ou então com i = 1, j = 2

o elemento em questão pode ser localizado da seguinte forma:

```
index = 1 * 4 + 2 = 6
```

O tamanho da memória necessária para alocação de uma matriz **n x m** é definida como:

```
n *m * (dataType)
```

onde **dataType** – é o tipo de elementos da matriz (int, float, double,..).

Nesse caso o compilador não é informado de forma explicita que se trata de uma matriz. Por isso as tentativas de acesso aos elementos de forma tradicional, como **mat[i][j],** serão consideradas incorretas.

O acesso deverá ser feito da forma:

```
* (matPtr+i*m+j)
```

onde:
matPtr – o ponteiro para matriz
m – a quantidade das colunas
i – a linha
j – a coluna

Exemplo 4: Matriz alocada dinamicamente (como vetor unidimensional)

```c
    int *matPtr; // pointer
    int i, j, n, m;

    printf("\n Quantidade de linhas: ");
    scanf("%d",&n);

    printf("\n Quantidade de colunas: ");
    scanf("%d",&m);

    // memory allocation
    matPtr = (int*) malloc(n*m*sizeof(int));

    // data input
    printf("\n Entrada de dados : \n");
    for(i=0; i < n; i++)
     {
       for(j=0; j < m; j++)
       {
         printf("mat[%d][%d] = ", i,j);
         scanf("%d", (matPtr + i*m +j) );
       }
     }

     // data output
    printf("\n\n Saida de dados: \n");
    for(i=0; i < n; i++)
     {
       for(j=0; j < m; j++)
        {
          printf("%5d ", *(matPtr + i*m +j));
        }
       printf("\n");
     }
```

Quantidade de linhas: 2
Quantidade de colunas: 3

Entrada de dados :
mat[0][0] = 1
mat[0][1] = 2
mat[0][2] = 3
mat[1][0] = 4
mat[1][1] = 5
mat[1][2] = 6

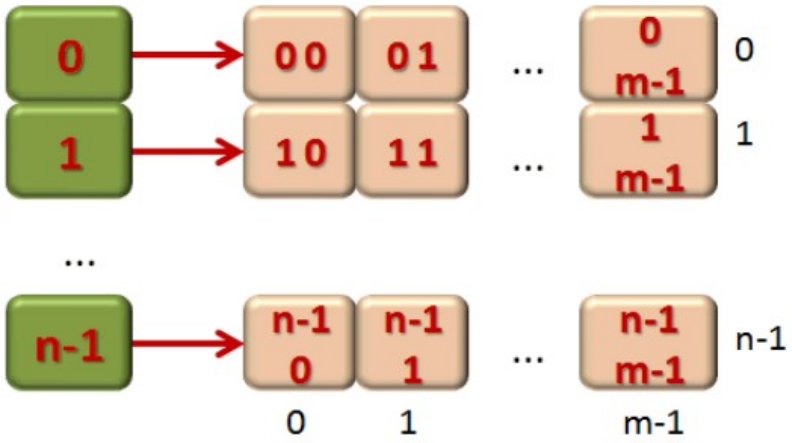
Saida de dados:
1 2 3
4 5 6

2. Criação de um vetor de ponteiros

Uma outra possibilidade para criação de uma matriz **n x m** é alocação de memória utilizando um vetor de ponteiros.

Para fazer isso é necessário:

- alocar a memória para um vetor de ponteiros
- alocar a memória para vetores unidimensionais (linhas da matriz)
- gravar os endereços das linhas dentro do vetor dos ponteiros



Representação gráfica

Nesse caso o compilador é informado explicitamente sobre a quantidade de linhas e colunas, portanto os elementos poderão ser acessados da forma convencional como **mat[i][j].**

Exemplo 5: Matriz alocada dinamicamente (como vetor bidimensional)

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

int main()
{
   int **matPtr; // pointer
   int i, j, n, m;

   printf("\n Quantidade de linhas: ");
   scanf("%d",&n);

   printf("\n Quantidade de colunas: ");
   scanf("%d",&m);

   // memory allocation for lines
   matPtr = (int**) malloc( n * sizeof( int* ));

   // data input
   printf("\n Entrada de dados : \n");
   for(i=0; i < n; i++)
    {
      // memory allocation for coluns
      matPtr[i] = (int*) malloc( m * sizeof( int ));
      for(j=0; j < m; j++)
      {
        printf("mat[%d][%d] = ", i,j);
        scanf("%d", &matPtr[i][j] );
      }
    }

    // data output
   printf("\n\n Saida de dados: \n");
   for(i=0; i < n; i++)
    {
      for(j=0; j < m; j++)
       {
         printf("%5d ", matPtr[i][j]);
       }
      printf("\n");
    }

 return 0;
}
```

```
Quantidade de linhas: 2
Quantidade de colunas: 3

 Entrada de dados :
mat[0][0] = 11
mat[0][1] = 12
mat[0][2] = 13
mat[1][0] = 14
```

```
mat[1][1] = 15
mat[1][2] = 16

 Saida de dados:
   11    12    13
   14    15    16
```