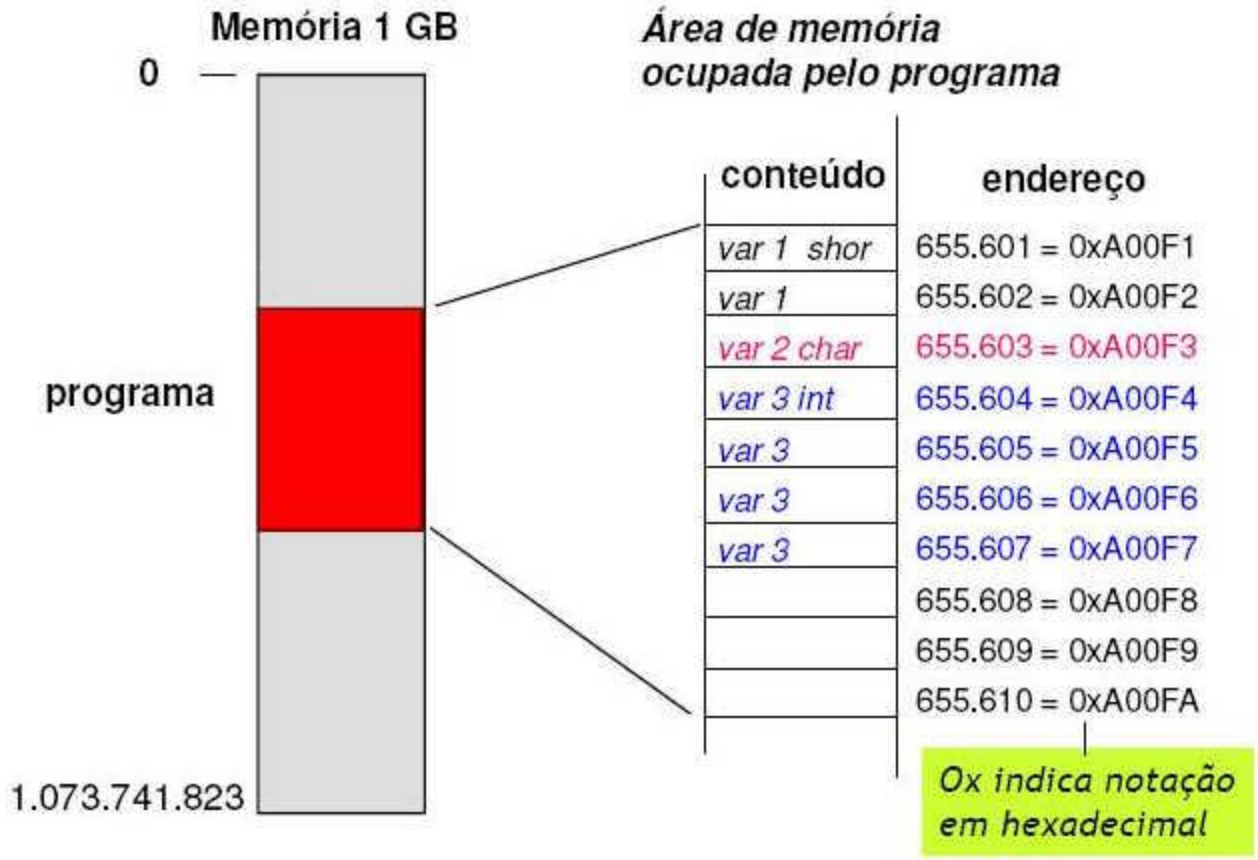
Ponteiros e registros

Professora Valéria

PONTEIROS: UTILIDADES

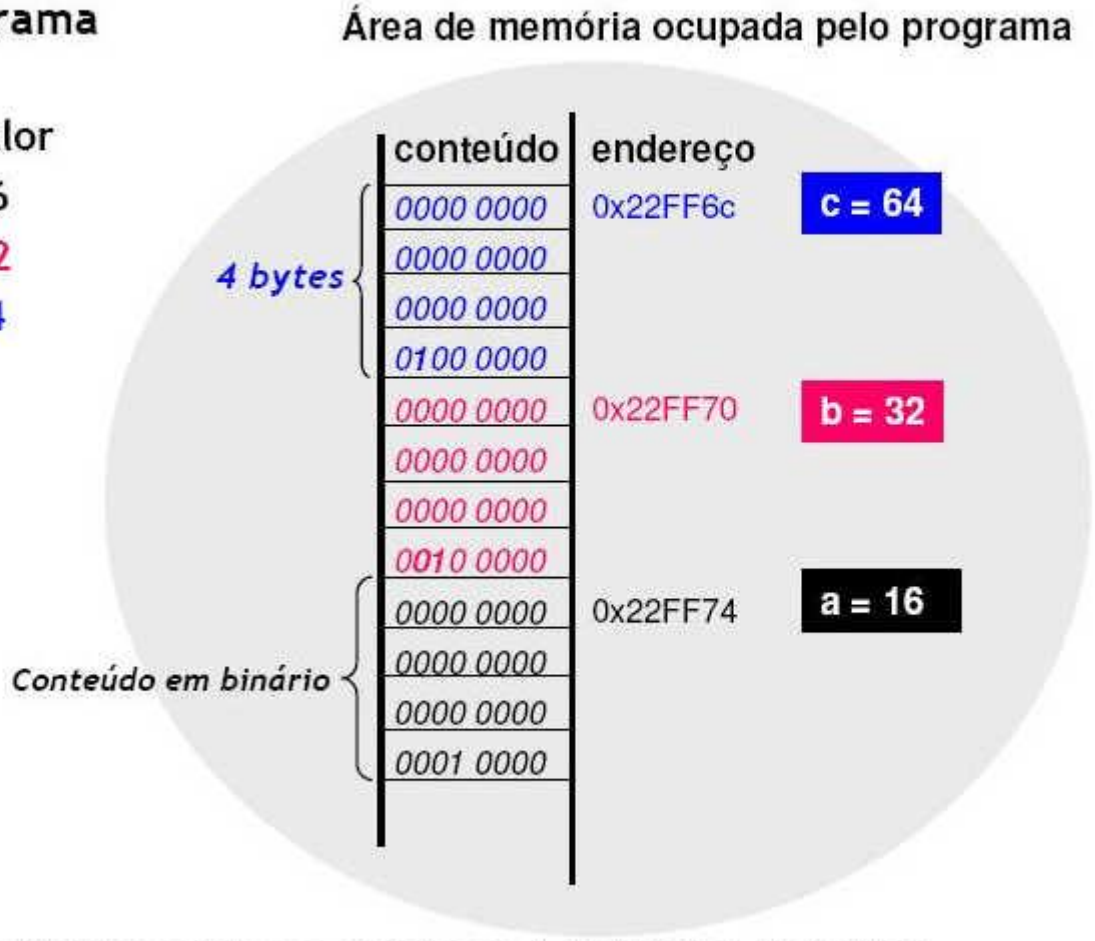
- Acessar elementos de um array
- Passar argumentos para um método quando este necessita modificar o argumento original
- Passar arrays e strings para métodos
- Obter memória do sistema
- Criar estruturas de dados, tais como listas encadeadas

- **Idéia básica:** todo byte na memória do computador tem um endereço; endereços são números que identificam os bytes da memória.



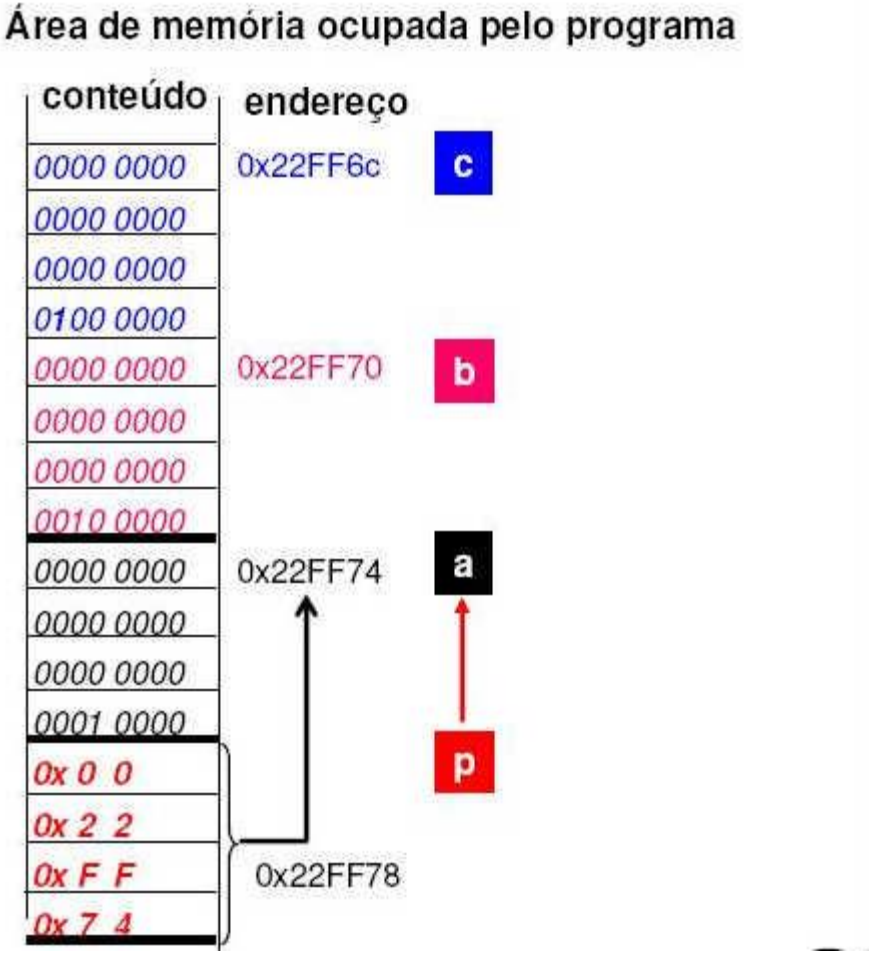
x	bin	dec
0	= 0000 =	0
1	= 0001 =	1
2	= 0010 =	2
3	= 0011 =	3
4	= 0100 =	4
5	= 0101 =	5
6	= 0110 =	6
7	= 0111 =	7
8	= 1000 =	8
9	= 1001 =	9
A	= 1010 =	10
B	= 1011 =	11
C	= 1100 =	12
D	= 1101 =	13
E	= 1110 =	14
F	= 1111 =	15

- **Resultado do programa anterior**
  - endereco valor
  - a: 0x22ff74 16
  - b: 0x22ff70 32
  - c: 0x22ff6c 64



- Observar que a diferença entre os endereços é de 4 bytes. Por que?

- `Ponteiro` é uma variável que armazena um endereço de uma variável
- *Exemplo: "p" aponta para a variável "a"*
- *Qual o conteúdo da variável p?*



- Ponteiros e vetores
  - O identificador de um vetor é equivalente ao endereço de seu primeiro elemento,
  - como um ponteiro é equivalente ao endereço do primeiro elemento que ele aponta,
  - assim de fato eles são (quase) a mesma coisa.

- A diferença:
  - Supondo essas 2 declarações:
    - **int numbers [20];**
    - **int * p;**
  - a seguinte operação será válida:
    - **p = numbers;**
  - mas a seguinte não:
    - **numbers = p;**
- Isso porque o vetor é um ponteiro constante, e nenhum valor pode ser associado a ele.

Declaração de uma variável tipo ponteiro para inteiro

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

int main() {
     int a = 16;
     int b = 64;
     cout << "endereco\tvalor" << endl;
     cout << "a: " << &a << "\t" << a << endl;
     cout << "b: " << &b << "\t" << b << endl;

     int* p; // ponteiro para inteiro
     p = &a;
     cout << "ponteiro: " << p << endl;
     p = &b;
     cout << "ponteiro: " << p << endl;
     getchar();
}
```

SINTAXE DE DECLARAÇÃO DE PONTEIROS

```
char*     cptr;
int*      iptr;
float*    fptr;
Contador* contptr;
```

```
char     *cptr;
int      *iptr;
float    *fptr;
Contador *contptr;
```

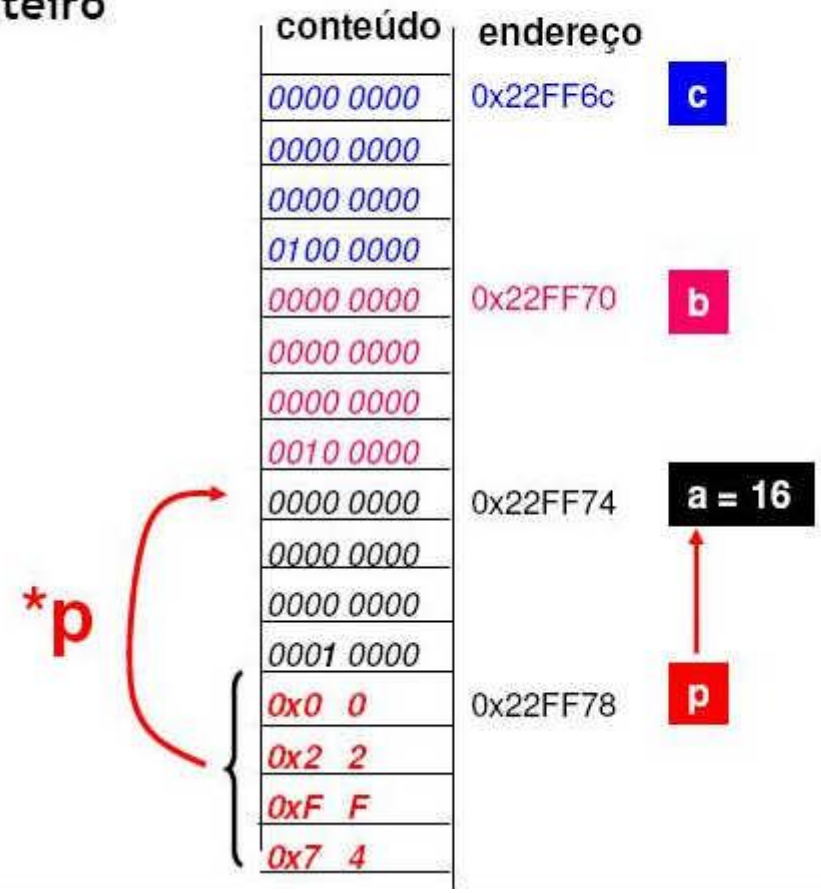
As duas formas são válidas

Mais utilizada

◇ Operador de indireção *

serve para acessar o conteúdo apontado por um ponteiro

*o valor retornado por *p é 16*



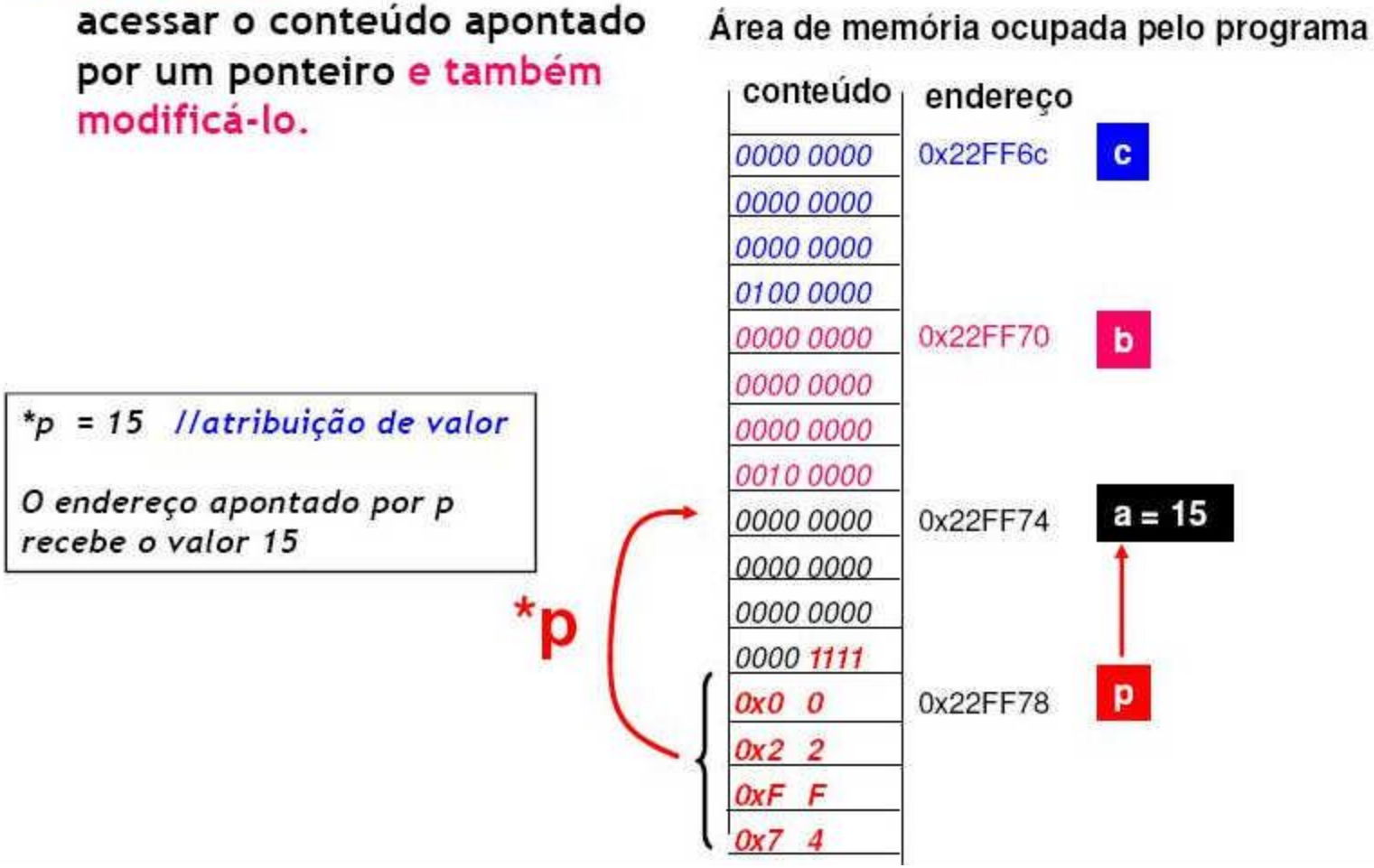
```
#include <iostream>
using namespace std;

int main() {
     int a = 16;
     int b = 64;
     cout << "valor de p\tconteudo apontado" << endl;
     int* p; // ponteiro para inteiro
     p = &a;
     cout << p << "\t" << *p << endl;
     p = &b;
     cout << p << "\t" << *p << endl;
     getchar();
}
```

**p é o conteúdo do endereço apontado por p; este conteúdo é interpretado como um número inteiro por causa do tipo do ponteiro (int*)*

◇ `operador *` serve para acessar o conteúdo apontado por um ponteiro e também modificá-lo.

*p = 15 //atribuição de valor

O endereço apontado por p recebe o valor 15



OPERADOR DE INDIREÇÃO

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

int main() {
     int a = 16;
     int b = 64;
     cout << "valor de p\tconteudo apontado" << endl;
     int* p; // ponteiro para inteiro
     p = &a;
     *p = 15; // idêntico: a=15
     cout << p << "\t" << *p << endl;

     b = *p; // idêntico: b=a
     cout << p << "\t" << *p << endl;
     getchar();
}
```

PONTEIRO PARA VOID

- O endereço colocado num ponteiro deve ser de uma variável de mesmo tipo do ponteiro

- *Exemplo:*
  - `int a;`
  - `int* p = &a;`
  - `char* pc = &a; // ERRO`

- `void *p` é uma ponteiro que pode apontar para qualquer tipo de dado

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

int main() {
     int a[5]={1,2,3,5,8};

     // acesso normal
     for (int i=0; i<5; i++)
         cout << a[i] << " ";

     cout << endl << endl;

     // acesso por ponteiro
     for (int j=0; j<5; j++)
         cout << *(a+j) << " ";
     getchar();
}
```

a variável a é um ponteiro para inteiro

ar[j] similar int *(a+j);

- **No exemplo anterior, inclua abaixo da linha indicada pela linha seguinte:**
  - cout << a << endl;
- **Em seguida, coloque**
  - cout << *a << endl;
- **Agora, coloque**
  - cout << a+1 << endl;
  - e compare como valor obtido pela primeira linha
- **Finalmente coloque**
  - cout << *(a+1) << endl;

```
#include <stdio.h>
#include <iostream>
using namespace std;
int main() {
        int v[5] = {1,2,3,4,5};
        int *p;
        p = v;
       for(int i = 0; i<5; i++){
            cout<<v[i] << " ";    }
        cout<<"\n";
        for(int i = 0; i<5; i++){
            cout<<*(v+i) << " ";    }
        cout<<"\n";
        for(int i = 0; i<5; i++){
            cout<<*(p+i) << " ";    }
        return 0;
}
```

PASSAGEM DE ARGUMENTOS

- **Argumentos podem ser passados aos métodos por 3 maneiras distintas:**
  - Por valor: uma cópia do valor da variável é passado ao método
    - o valor da variável original não é modificado
    - Existe em JAVA
  - Por referência: o endereço da variável é passado ao método
    - o valor da variável original é modificado
    - Alias
    - Existe em JAVA
  - Por ponteiro: o endereço da variável é passado ao método
    - o valor da variável original é modificado
    - Ponteiro
    - NÃO EXISTE EM JAVA

```
// Este programa ilustra passagem de argumentos
// por referencia. Observar que o argumento
// r em paraSegundos e a variavel h sao
// simbolos alternativos (alias) e portanto sao a mesma
// coisa.

#include <iostream>
using namespace std;

void paraSegundos(int &r) {
     cout << "endereco r = " << &r << endl;
     r = r*3600;
}

int main() {
    int h = 5;
    cout << "endereco h = " << &h<< endl;
    cout << "h = " << h << endl;

    paraSegundos(h); // passa a referencia da var h na tabela
                     // de simbolos (indice)
    cout << "h = " << h << endl;
    getchar();
}
```

CPPRepositorio/Ponteiros/PassagemRef.cpp

PASSAGEM POR REFERÊNCIA

Tabela de símbolos

Referência	Símbolo	Endereço	Conteúdo
1	h alias r	0x22FF74	5 *(inteiro)*
2	...	...	
...	...	...	

r e h são dois símbolos alternativos

r aliás h = r, de outra forma, h

```cpp
// Este programa ilustra passagem de argumentos
// por ponteiro. Observar que o argumento
// r em paraSegundos e a variavel h sao
// simbolos que apontam para o mesmo endereco, mas
// ocupam posicoes distinas na memoria.

#include <iostream>
using namespace std;

void paraSegundos(int *r) {
     cout << "endereco r = " << &r << endl;
     *r = *r * 3600;
}

int main() {
    int h = 5;
    cout << "endereco h = " << &h<< endl;
    cout << "h = " << h << endl;

    paraSegundos(&h); // passa o endereco de h
    cout << "h = " << h << endl;
    getchar();
}
```

CPPRepositorio/Ponteiros/PassagemPonteiro.cpp

PASSAGEM POR PONTEIRO

Tabela de símbolos

Referência	Símbolo	Endereço	Conteúdo
1	h	0x22FF74	5 *(inteiro)*
2	r	0x22FF50	0x22FF74
...	...	...	

r e h são dois símbolos diferentes, cada um ocupa uma posição de memória.

r é um ponteiro que aponta para h

ALOCAÇÃO DE MEMÓRIA EM C E C++

Alocação de memória

- O tamanho do vetor é determinado na sua declaração.
- Enquanto que na alocação dinâmica o tamanho do vetor pode ser redimensionado do decorrer do programa.
- No entanto, ao final o espaço alocado deve ser liberado.

Alocação de memória em C

- Utilização de malloc() contida na biblioteca malloc.h e possui a seguinte sintaxe:
  tipoDoPonteiro= malloc(valorParaAlocação * tamanhoDoTipo);

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

int main(void)
{
	unsigned short int tamanho;
	char *string;	/*ponteiro para char, é necessário que seja uma ponteiro para ser alocado*/

	printf("\nDigite o tamanho da string: ");
	scanf("%d",&tamanho);

	string= malloc( tamanho * sizeof(char) ); /*o sizeof ajuda a aumentar a portabilidade*/

	printf("\nDigite a string: ");
	scanf("%s",string);
	printf("\n%s",string);

	free(string);	/*libera a memória alocada*/

	return 0;
}
```

Alocação de memória em C: 1ª forma

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

int main(void)
{
	unsigned short int linhas, colunas, i;
	int *vetor, **matriz; /*um vetor grande e um ponteiro para ponteiros que será a matriz*/

	printf("\nDigite o numero de linhas e colunas: ");
	scanf("%d,%d",&linhas,&colunas);

	vetor= malloc( linhas * colunas * sizeof(int) );
	*matriz= malloc( linhas * sizeof(int) );

	for(i=0;i<linhas;i++)
		matriz[ i ]= &vetor[ i * colunas];

			/*a partir deste ponto pode ser usado matriz[a][ b ] sendo a o número da linha e b o número da
coluna para qualquer parte do programa*/

	 free(vetor);
	 free(matriz);  /*faz a liberação da memória alocada*/

	 return 0;
}
```

Alocação de memória em C: 2ª forma

```
#include <stdio.h>
#include <malloc.h>
/*includes*/

int main(void)
{
      unsigned short int linhas, colunas, i;
      int **matriz;

      printf("\nDigite o número de linhas e colunas: ");
      scanf("%d,%d",&linhas, &colunas);

      *matriz= malloc( linhas * sizeof( int ) );
      for(i=0;i<linhas;i++)
            matriz[ i ]= malloc( colunas * sizeof( int ) );

      /*a partir deste ponto pode ser usado matriz[a][ b ] sendo a o número da linha e b o número da coluna
para qualquer parte do programa*/

        for(i=0; i<linhas; i++)
              É uma free(matriz[ i ]);
        free(matriz);  /*faz a liberação da memória alocada*/

        return 0;
}
```

Alocação de memória em C++

```
#include <iostream>
 using namespace std;
 int main ()
 {
   int numTests;
   cout << "Enter the number of test scores:";
   cin >> numTests;
   int * iPtr = new int[numTests];          //colocamos um ponteiro no inicio da memória dinâmica
   for (int i = 0; i < numTests; i++)
   {
      cout << "Enter test score #" << i + 1 << " : ";
      cin >> iPtr[i];
   }
   for (int i = 0; i < numTests; i++)
      cout << "Test score #" << i + 1 << " is "<< iPtr[i] << endl;
   delete [] iPtr;
   system ("pause");
   return 0;
 }
```

Pesquisa

- Como fazer alocação dinâmica de memória de matrizes em C++?

Exercícios

- Faça um programa, utilizando funções e alocação dinâmica de memória
  - a) Calcular a média de um vetor de inteiros de tamanho dado pelo usuário.
  - b) Achar o maior elemento deste vetor.
  - c) Ordenar este vetor.