Linguagem de Programação I

Aula 14 - Alocação Dinâmica de Memória

Objetivos da aula

- **Introduzir os conceitos de alocação dinâmica de memória em C e C++**
- Para isso, estudaremos:
  - Alocação estática x alocação dinâmica
  - Comandos básicos de gerenciamento de memória
- Ao final da aula espera-se que o aluno seja capaz de:
  - Distinguir a alocação estática da alocação dinâmica
  - Desenvolver programas capazes de gerenciar dinamicamente a memória do computador

Alocação estática x dinâmica

- As linguagens de programação C e C++ permitem dois tipos de alocação de memória:
  - Estática
  - Dinâmica
- Na alocação estática, o espaço de memória para as variáveis é reservado no início da execução, não podendo ser alterado depois

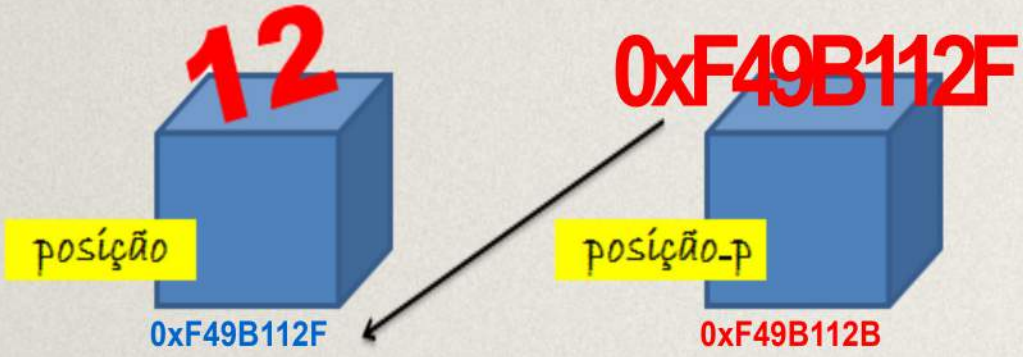
```
1. int numero;
2. int pontuacao[20];
```

- Na alocação dinâmica, o espaço de memória para as variáveis pode ser alocado dinamicamente durante a execução do programa
  - Ponteiros se fazem necessários

Revisão: Ponteiros (1)

- Tipo especial de variável que armazena endereços de memória e permite acessá-los diretamente

```
1.  ...
2.  int posicao = 12;
3.  int *posicao_p;
4.  posicao_p = &posicao;
5.  ...
```



Variável comum

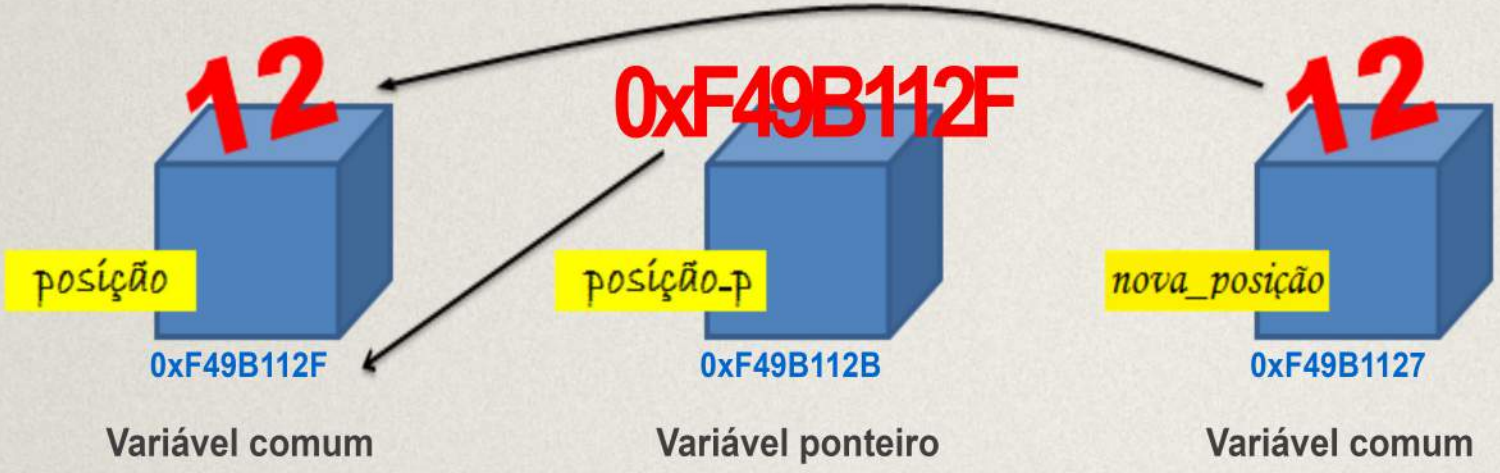
Variável ponteiro

	Endereço	Conteúdo
	...	...
	0xF49B1134	?
	0xF49B1133	?
posicao	0xF49B1132	00001100
posicao	0xF49B1131	00000000
posicao	0xF49B1130	00000000
posicao	0xF49B112F	00000000
posicao_p	0xF49B112E	00101111
posicao_p	0xF49B112D	00010001
posicao_p	0xF49B112C	10011011
posicao_p	0xF49B112B	11110100
	0xF49B112A	?
	0xF49B1129	?
	...	...

Revisão: Ponteiros (2)

```
1. int posicao = 12;
2. int *posicao_p = &posicao;
3. int nova_posicao = *posicao_p;
```

- Operadores utilizados para manipular ponteiros
  - Operador de acesso a memória **&** (**referenciamento**)
    - retorna o endereço de uma variável
  - Operador de indireção * (**desreferenciamento**)
    - retorna o conteúdo do endereço de uma variável apontada
    - * de indireção ≠ * de multiplicação ≠ * do tipo ponteiro



Alocação dinâmica

- A alocação dinâmica de memória é um mecanismo bastante útil na solução de problemas que exigem grandes conjuntos de dados
  - Meio pelo qual um programa pode obter memória enquanto está em execução, sendo gerenciado pelo próprio programador
- Ela pode oferecer grandes benefícios em termos de desempenho e de utilização de recursos
  - A memória alocada dinamicamente é obtida através do segmento de heap, onde apenas o espaço de memória necessário em um dado momento é efetivamente utilizado

Alocação Dinâmica

- As linguagens C e C++ permitem que o programador tenha um alto grau de controle sobre a máquina através da alocação dinâmica
- Elas possuem ambas dois comandos básicos para gerenciamento de memória
  - Comandos da linguagem C++:
    - **new** aloca memória
    - **delete** libera memória alocada
  - Comandos da linguagem C:
    - **malloc** aloca memória
    - **free** libera memória alocada

Alocação dinâmica em C++ (1)

- Operador new
  - Aloca uma área de memória do tamanho correspondente à representação do tipo declarado
  - Retorna um ponteiro do tipo declarado apontando para o início da área alocada, ou NULL caso não seja possível alocar a memória requisitada
- Sintaxe para alocação de uma variável ponteiro do tipo T

```
T *p = new T;
```

Alocação dinâmica em C++ (2)

- Exemplo:

```
1.  #include <iostream>
2.
3.  int main()
4.  {
5.      int *p = new int;  // alocação de variável ponteiro do tipo inteiro
6.
7.      /* IMPORTANTE: convém sempre verificar se a alocação ocorreu corretamente,
8.         ou seja, se o retorno do operador new é diferente de NULL */
9.
10.     if( p )
11.     {
12.         std::cout << "Memoria alocada" << std::endl;
13.         std::cout <<  p << std::endl;  // imprime o endereço de p
14.         std::cout << *p << std::endl;  // imprime o conteúdo de p
15.         *p = 10;                       // inicializa o conteúdo de p com 10
16.         std::cout << *p << std::endl;  // imprime o conteúdo de p
17.     }
18.     else
19.         std::cout << "Alocacao impossivel" << std::endl;
20.     return 0;
21. }
```

Alocação dinâmica em C++ (3)

- Operador delete
  - Libera a área de memória previamente alocada no sistema utilizando o seu endereço inicial como parâmetro
  - O sistema operacional se encarrega de gerenciar lacunas do heap
- Exemplo:

```
1.  int main()
2.  {
3.      int *p = new int;  // alocação de variável ponteiro do tipo inteiro
4.      if( p )            // verifica se a alocação ocorreu corretamente
5.          delete p;      // libera a memória alocada
6.      return 0;
7.  }
```

Erros comuns da alocação dinâmica

- Não alocar memória antes de acessar o conteúdo do ponteiro
  - Para acessar o conteúdo, sempre deve ser verificado se o ponteiro é válido
- Copiar o conteúdo do ponteiro ao invés do conteúdo da variável apontada
- Não liberar memória alocada previamente quando ela passar a ser desnecessária
- Tentar acessar o conteúdo de um ponteiro depois da sua memória já ter sido liberada
  - O valor nulo (0) deve ser sempre atribuído ao ponteiro após à sua liberação de memória

Exercite-se (1)

- O que está errado neste programa?

```
1.  #include <iostream>
2.
3.  int main()
4.  {
5.      int a, b, *p;
6.      a = 2;
7.      *p = 3;
8.      b = a + (*p);
9.      std::cout << a << std::endl;
10.     return 0;
11. }
```

Exercite-se (2)

- O que será impresso no seguinte programa?

```
1.  #include <iostream>
2.  int main()
3.  {
4.      double a, *p, *q;
5.      a = 3.14;
6.      std::cout << a << std::endl;
7.      p = &a;
8.      *p = 2.718;
9.      std::cout << a << std::endl;
10.     a = 5;
11.     std::cout << *p << std::endl;
12.     p = NULL;
13.     p = new double;
14.     *p = 20;
15.     q = p;
16.     std::cout << *p << std::endl;
17.     std::cout << a << std::endl;
18.     delete p;
19.     std::cout << *q << std::endl;
20.     return 0;
21. }
```

Alocação dinâmica de vetores

- Como vetores são ponteiros em linguagem C/C++, a alocação dinâmica de vetores se faz incrementando um multiplicador (número de elementos) na lógica utilizada para as variáveis simples

Alocação estática

```
1.  #include <iostream>
2.
3.  int main()
4.  {
5.      int a[10], *b;
6.      b = a;
7.
8.      b[5] = 100;
9.      std::cout << a[5] << std::endl;
10.     std::cout << b[5];
11.     return 0;
12. }
```

Alocação dinâmica

```
1.  #include <iostream>
2.
3.  int main()
4.  {   // b = a não é permitido aqui
5.      int a[10], *b;
6.      b = new int[10];
7.
8.      b[5] = 100;
9.      std::cout << a[5] << std::endl;
10.     std::cout << b[5];
11.     return 0;
12. }
```

Alocação dinâmica de matrizes (1)

- Alocação de matrizes se faz da mesma forma que para vetores, incrementada do conceito de indireção múltipla
- A indireção múltipla (ponteiro de ponteiros) se aplica a qualquer dimensão desejada

```
1.  #include <iostream>
2.  int main()
3.  {
4.      float **matriz;    // ponteiro de ponteiros para a matriz
5.      int linhas = 10, colunas = 15;
6.      matriz = new float*[linhas];  // aloca as linhas da matriz
7.
8.      if( matriz != NULL )
9.          for( int i = 0; i < linhas; i++ )
10.         {
11.             matriz[i] = new float[colunas]; // aloca as colunas da matriz
12.             if( matriz[i] == NULL ) {
13.                 std::cout << "Memoria Insuficiente" << std::endl;
14.                 break;
15.             }
16.         }
17.     return 0;
18. }
```

Alocação dinâmica de matrizes (2)

- A liberação de memória das matrizes deve ser efetuada para todos os ponteiros da indireção múltipla

```
1.  #include <iostream>
2.
3.  int main()
4.  {
5.      float **matriz;  // ponteiro de ponteiros para a matriz
6.      int linhas = 10, colunas = 15;
7.
8.      ... // Considerando a alocação de memória efetuada
9.      if( matriz != NULL )
10.     {
11.         for( int i = 0; i < linhas; i++ )
12.             delete matriz[i];  // libera as colunas da matriz
13.         delete matriz;         // libera as linhas da matriz
14.     }
15.     return 0;
16. }
```

Alocação dinâmica de registros

- Registros são tipos compostos definidos pelo usuário que podem ser alocados dinamicamente da mesma forma que tipos primitivos

```
1.  #include <iostream>
2.  typedef struct {
3.      int idade;
4.      double salario;
5.  } Registro;
6.
7.  int main()
8.  {
9.      Registro *r;
10.     r = new Registro;
11.     if( r )
12.     {
13.         r->idade = 30;
14.         r->salario = 1000.;
15.         delete r;
16.         r = 0;  // garante que o ponteiro não aponta mais
17.     }           // para o espaço de memória liberado
18.     return 0;
19. }
```

Alocação dinâmica em C (1)

- Função malloc (cstdlib)

```
void* malloc( unsigned int numero_bytes );
```

  - Aloca uma área de memória com numero_bytes bytes
  - Retorna um ponteiro do tipo void para o início da área alocada, ou NULL caso não seja possível alocar a memória requisitada
    - O conteúdo deste ponteiro pode ser atribuído a qualquer variável do tipo ponteiro através de um typecasting

- Sintaxe para alocação de uma variável ponteiro do tipo T

```
T *p = (T*) malloc( sizeof( T ) );  // cast para o ponteiro do tipo T
```

Alocação dinâmica em C (2)

- Exemplo:

```
1.  #include <iostream>
2.  #include <cstdlib>
3.
4.  int main()
5.  {
6.      int *p = (int*) malloc( sizeof( int ) );
7.      // alocação de variável ponteiro do tipo inteiro
8.      // (int*) é o cast para ponteiro do tipo inteiro
9.
10.     if( p )
11.         std::cout << "Memória alocada" << std::endl;
12.     else
13.         std::cout << "Alocacao impossivel" << std::endl;
14.     return 0;
15. }
```

Alocação dinâmica em C (3)

- Função free (cstdlib)

```
void free( void *endereco );
```

  - Libera a área de memória previamente alocada no sistema utilizando o seu endereço inicial como parâmetro
  - O sistema operacional se encarrega de gerenciar lacunas do heap

- Exemplo:

```
1. #include <cstdlib>
2. int main()
3. {
4.     int *p = (int*) malloc( sizeof( int ) );  // alocação da variável
5.     if( p )                 // verifica se a alocação ocorreu corretamente
6.         free( p );          // libera a memória alocada
7.     return 0;
8. }
```

Resumo da aula

- A alocação dinâmica é um mecanismo que permite um programa obter memória durante a sua execução, sendo gerenciado pelo próprio programador
- Ela oferece grandes benefícios em termos de desempenho e de utilização de recursos
- As linguagens C e C++ permitem que o programador tenha um alto grau de controle sobre a máquina através da alocação dinâmica
- Os comandos básicos para gerenciamento de memória são new e delete em linguagem C++ e malloc e free em linguagem C
- Deve-se ter bastante cuidado com a manipulação de ponteiros utilizados para alocação dinâmica