Curso de Ciência da Computação

Algoritmos e Programação de Computadores 2per

Programação Orientada a Objetos POO

Profa. Fernanda dos Santos Cunha

Gerenciamento de memória dinâmico com os operadores new e delete

- Gerenciamento de memória dinâmico
  - Permite que os programadores aloquem e desaloquem memória para qualquer tipo predefinido ou definido pelo usuário.
  - É realizado pelos operadores `new` e `delete`.
    - Ex.: alocar memória dinamicamente para um vetor, em vez de declará-lo diretamente como um vetor de tamanho fixo.

Gerenciamento de memória dinâmico com os operadores new e delete

- Operador `new`
  - Aloca (isto é, reserva) armazenamento de tamanho apropriado para um objeto em tempo de execução.
  - Chama um construtor para inicializar o objeto.
  - Retorna um ponteiro do tipo especificado à direita de `new`.
  - Pode ser usado para alocar dinamicamente qualquer tipo fundamental (com `int` ou `double`) ou qualquer tipo de classe.

Gerenciamento de memória dinâmico com os operadores new e delete

- Operador `delete`
  - Destrói um objeto alocado dinamicamente.
  - Chama o destrutor do objeto.
  - Desaloca (isto é, libera) memória do armazenamento livre.
  - A memória pode então ser reutilizada pelo sistema para alocar outros objetos.

Gerenciamento de memória dinâmico com os operadores new e delete

- Inicializando um objeto alocado por `new`
  - Inicializador para uma variável de tipo primitivo recém-criada

```
double *ptr = new double( 3.14159 );
```

  - Lista de argumentos para o construtor de um objeto

```
Time *timePtr = new Time( 12, 45, 0 );
```

Gerenciamento de memória dinâmico com os operadores new e delete

- Armazenamento livre
  - *Heap* é uma região da memória do computador atribuída a cada programa para armazenar objetos criados em tempo de execução.
- Erro de programação:
  - Não liberar memória alocada dinamicamente quando ela não é mais necessária pode fazer com que o sistema fique sem memória prematuramente. Conhecido por "vazamento de memória".

Gerenciamento de memória dinâmico com os operadores new e delete

- O operador `new` pode ser usado para **alocar vetores dinamicamente**.
  - Alocar dinamicamente um vetor de inteiros de 10 elementos:

```
int *gradesArray = new int[ 10 ];
```

  - A área de dados do vetor é alocada dinamicamente.
  - O **`tamanho`** é especificado por meio de qualquer expressão integral que possa ser avaliada em tempo de execução.

Gerenciamento de memória dinâmico com os operadores new e delete

- Excluir um vetor alocado dinamicamente:

```
delete [] gradesArray;
```

  - Libera memória apontada por `gradesArray`.
  - Se o ponteiro apontar para um vetor de objetos:
    - Primeiro chama-se o destrutor para cada objeto no vetor.
    - Em seguida, desaloca-se a memória.
  - **OBS.:** Se a instrução não incluir os colchetes (`[]`) e `gradesArray` apontar para um vetor de objetos, apenas o primeiro objeto no vetor será excluído.

Gerenciamento de memória dinâmico com os operadores new e delete

- Utilizar `delete` ao invés de `delete []` para vetores de objetos pode provocar erros de lógica em tempo de execução.
- Para cada objeto no vetor receber uma chamada de destrutor, deve-se sempre excluir a memória alocada como vetor com o operador `delete []`.
- De modo semelhante, deve-se sempre excluir a memória alocada como um elemento individual com o operador `delete`.

Resumo

fig10_21.cpp

Exemplo do Deitel

```
1  // Figura 10.21: Employee.h
2  // Definição da classe Employee.
3  #ifndef EMPLOYEE_H
4  #define EMPLOYEE_H
5
6  class Employee
7  {
8  public:
9     Employee( const char * const, const char * const ); // construtor
10    ~Employee(); // destrutor
11    const char *getFirstName() const; // retorna o nome
12    const char *getLastName() const; // retorna o sobrenome
13
14    // função-membro static
15    static int getCount(); // retorna número de objetos instanciados
16 private:
17    char *firstName;
18    char *lastName;
19
20    // dados static
21    static int count; // número de objetos instanciados
22 }; // fim da classe Employee
23
24 #endif
```

Resumo

Employee.cpp (1 de 3)

```
1  // Figura 10.22: Employee.cpp
2  // Definições de função-membro da classe Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // protótipos de strlen e strcpy
8  using std::strlen;
9  using std::strcpy;
10
11 #include "Employee.h" // definição da classe Employee
12
13 // define e inicializa o membro de dados static no escopo de arquivo
14 int Employee::count = 0;
15
16 // define a função-membro static que retorna o número de
17 // objetos Employee instanciados (static declarado em Employee.h)
18 int Employee::getCount()
19 {
20    return count;
21 } // fim da função static getCount
22
```

Resumo

Employee.cpp (2 de 3)

```
23 // o construtor aloca dinamicamente espaço para o nome e o sobrenome e
24 // usa strcpy para copiar o nome e o sobrenome para o objeto
25 Employee::Employee( const char * const first, const char * const last )
26 {
27    firstName = new char[ strlen( first ) + 1 ];
28    strcpy( firstName, first );
29
30    lastName = new char[ strlen( last ) + 1 ];
31    strcpy( lastName, last );
32
33    count++; // incrementa contagem estática de empregados
34
35    cout << "Employee constructor for " << firstName
36       << ' ' << lastName << " called." << endl;
37 } // fim do construtor Employee
38
39 // o destrutor desaloca memória dinamicamente alocada
40 Employee::~Employee()
41 {
42    cout << "~Employee() called for " << firstName
43       << ' ' << lastName << endl;
44
45    delete [] firstName; // libera memória
46    delete [] lastName; // libera memória
47
48    count--; // decrementa contagem estática de empregados
49 } // fim do destrutor ~Employee
```

Vetores `char` dinamicamente alocados.

Desalocando a memória reservada.

Resumo
Employee.cpp
(3 de 3)

```
50
51 // retorna o nome do empregado
52 const char *Employee::getFirstName() const
53 {
54    // const antes do tipo de retorno impede que o cliente modifique
55    // dados private; o cliente deve copiar a string retornada antes de
56    // o destrutor excluir o armazenamento para impedir um ponteiro indefinido
57    return firstName;
58 } // fim da função getFirstName
59
60 // retorna sobrenome do empregado
61 const char *Employee::getLastName() const
62 {
63    // const antes do tipo de retorno impede que o cliente modifique
64    // dados private; o cliente deve copiar a string retornada antes de
65    // o destrutor excluir o armazenamento para impedir um ponteiro indefinido
66    return lastName;
67 } // fim da função getLastName
```

Resumo
fig10_23.cpp
(1 de 2)

```
1  // Figura 10.23: fig10_23.cpp
2  // Driver para testar a classe Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Employee.h" // Definição da classe Employee
8
9  int main()
10 {
11    // utiliza o nome da classe e o operador de resolução de escopo binário para
12    // acessar a função static number getCount
13    cout << "Number of employees before instantiation of any objects is "
14       << Employee::getCount() << endl; // utiliza o nome da classe
15
16    // utiliza new para criar dinamicamente dois novos Employees
17    // operador new também chama o construtor do objeto
18    Employee *e1Ptr = new Employee( "Susan", "Baker" );
19    Employee *e2Ptr = new Employee( "Robert", "Jones" );
20
21    // chama getCount no primeiro objeto Employee
22    cout << "Number of employees after objects are instantiated is "
23       << e1Ptr->getCount();
24
25    cout << "\n\nEmployee 1: "
26       << e1Ptr->getFirstName() << " " << e1Ptr->getLastName()
27       << "\nEmployee 2: "
28       << e2Ptr->getFirstName() << " " << e2Ptr->getLastName() << "\n\n";
```

Criando dinamicamente `Employees` com `new`.

Resumo

fig10_23.cpp
(2 de 2)

Liberando memória para a qual um ponteiro aponta.

```
29
30    delete e1Ptr; // desaloca memória
31    e1Ptr = 0; // desconecta o ponteiro do espaço de armazenamento livre
32    delete e2Ptr; // desaloca memória
33    e2Ptr = 0; // desconecta o ponteiro do espaço de armazenamento livre
34
35    // não existe nenhum objeto, portanto chama a função-membro static getCount
36    // utilizando o nome da classe e o operador de resolução de escopo binário
37    cout << "Number of employees after objects are deleted is "
38       << Employee::getCount() << endl;
39    return 0;
40 } // fim de main
```

Desconectando um ponteiro de qualquer espaço na memória.

```
Number of employees before instantiation of any objects is 0
Employee constructor for Susan Baker called.
Employee constructor for Robert Jones called.
Number of employees after objects are instantiated is 2

Employee 1: Susan Baker
Employee 2: Robert Jones

~Employee() called for Susan Baker
~Employee() called for Robert Jones
Number of employees after objects are deleted is 0
```

Outro exemplo


```
#include <iostream>
#include <cstring>
using namespace std;
class Employee{
    char *nome;
public:
    Employee();
    ~Employee(); // agora será necessário !!!
    void printData();
    void getData();
};
Employee::Employee(){
     nome = NULL; // "zerando" o ponteiro
     cout << "Obj criado na memoria" << endl;
}
```

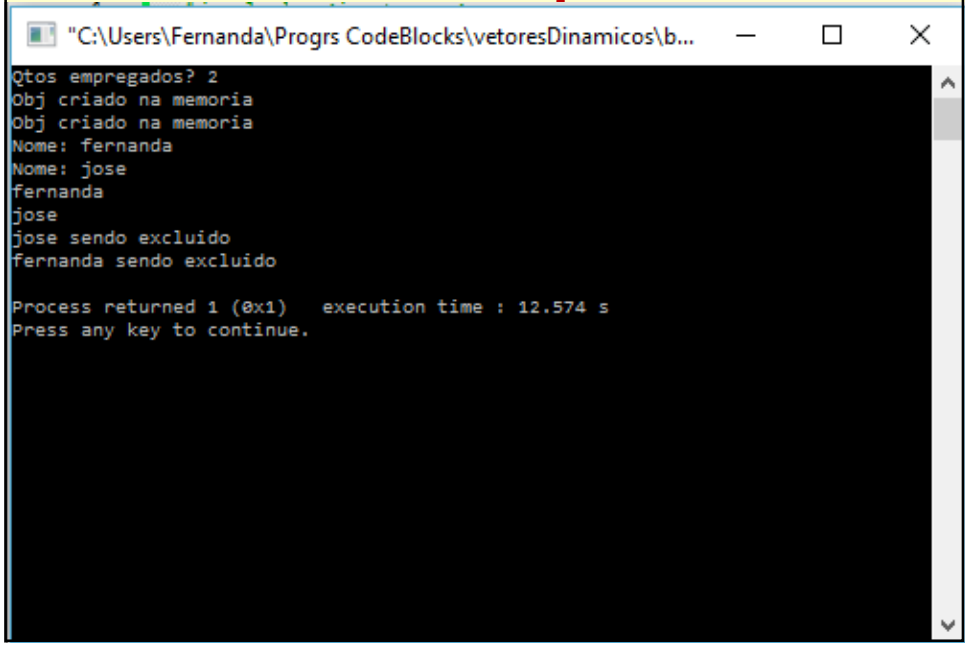
Outro exemplo


```
Employee::~Employee(){
     cout << nome << "sendo excluido" << endl;
     delete [] nome; }
void Employee::printData(){
     cout << nome << endl; }
void Employee::getData(){
    string n;
    cout << "Nome: ";
    getline(cin, n);
    nome = new char[n.size()+1]; // aloca mem
    strcpy(nome, n.c_str());
    // copia conteudo da string n para nome
}
```

Outro exemplo


```
int main() {
     int n;
     cout << "Qtos empregados? ";
     cin >> n; cin.ignore();
     Employee *vetDinam = new Employee[n];
     // cria vetor com n objetos Employee, ja
     // invocando os construtores s/parametros
     for(int i=0;i<n;i++)
          vetDinam[i].getData();
     for(int i=0;i<n;i++)
          vetDinam[i].printData();
     delete [] vetDinam;
     return 1; }
```

Outro exemplo



Exercício: Implemente a classe abaixo:

```
#ifndef VETOR_H_INCLUDED
#define VETOR_H_INCLUDED
class Vetor {
    int tam; // tamanho
    int *ptr; // ponteiro p/1º elemento do array
public:
    Vetor(int = 2); // construtor padrão, min. 2 elem
    ~Vetor(); // destrutor
    int getTam() const; // tamanho
    void preencheVetor();
    void mostraVetor();
    void ordenaVetor();
    int pesquisaElemento(); // traz posicao ou -1
    void insereNovoElemento (int elemento); // no fim
    void excluiElemento(int posicao);
};
```

Exercício


```
#include <iostream>
#include "Vetor.h"
using namespace std;

Vetor::Vetor( int T ) {
    // valida o VETOR
    tam = (T>0 ? T : 2 ); // 2 elementos no mínimo
    ptr = new int[ tam ]; // aloca espaco de dados
    for (int i =0; i < tam; i++ )
        ptr[i] = 0; // inicializa o vetor com 0s
}

Vetor::~Vetor( ) {
    delete [] ptr;
}
```