Estácio

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ESTRUTURA DE DADOS

AULA 6

RICARDO EIJI KONDO, Me.

TÓPICOS

UNIDADE III - Alocação Dinâmica de Memória e Recursividade

3.1 - Alocação Dinâmica de Memória

3.1.1 - Fundamentos

3.1.2 - Aplicação em C/C++: Ponteiros

3.1.2.1 - Operador de endereço (&), operador de indireção (*) e operador seta (->)

3.1.2.2 - Alocação e desalocação de memória

PONTEIROS E FUNÇÕES

- Para passar uma variável para um ponteiro, que é parâmetro de uma função, esta variável deve ser precedida pelo operador de endereço &.

```
1  #include <iostream>
2  using namespace std;
3
4  void AjustarPreco(float *Preco, float Taxa);
5
6  int main(void){
7      float PrecoProduto;
8      PrecoProduto = 500.00;
9      AjustarPreco(&PrecoProduto, 20);
10     cout << "Preço Reajustado: " << PrecoProduto << endl;
11 }
12
13
14 void AjustarPreco(float *Preco, float Taxa){
15     float ValorReajuste;
16     ValorReajuste = (*Preco * Taxa) / 100; // Calcula o valor do reajuste
17     *Preco = *Preco + ValorReajuste; // Reajusta o valor
18 }
```

- Exemplo vetores:

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  void GerarValores(float *PagtoM, float VlrPagto);
 5
 6  int main(void){
 7      int Ind;
 8      // Armazena o código do aluno e sua nota bimestral
 9      float PagtoMes[12];
10      GerarValores(PagtoMes, 150.80);
11
12      for(Ind = 0; Ind < 12; Ind++){
13          cout << "Pagamento Mes " << Ind << ": " <<PagtoMes[Ind] << endl;
14      }
15  }
16
17
18  void GerarValores(float *PagtoM, float VlrPagto){
19      for(int Ind = 0; Ind < 12; Ind++){
20          *(PagtoM + Ind) = VlrPagto;
21      }
22  }
```

PONTEIROS E FUNÇÕES

- Exemplo matriz:

```
1  #include <iostream>
2  using namespace std;
3
4  void Imprimir(int *TurmaAD);
5
6  int main(void){
7      // Armazena: Turma, Qtde Aluno, Qtde Disciplinas
8      int TurmaALunoDisc[3][3];
9      TurmaALunoDisc[0][0] = 100; // Nro da turma
10     TurmaALunoDisc[0][1] = 20; // Qtde de alunos
11     TurmaALunoDisc[0][2] = 6; // Qtde Discliplinas
12     TurmaALunoDisc[1][0] = 200; // Nro da turma
13     TurmaALunoDisc[1][1] = 18; // Qtde de alunos
14     TurmaALunoDisc[1][2] = 5; // Qtde Discliplinas
15     TurmaALunoDisc[2][0] = 300; // Nro da turma
16     TurmaALunoDisc[2][1] = 15; // Qtde de alunos
17     TurmaALunoDisc[2][2] = 4; // Qtde Discliplinas
18     Imprimir(&TurmaALunoDisc[0][0]);
19 }
20
21 void Imprimir(int *TurmaAD){
22     for(int Lin = 0; Lin < 3; Lin++) {
23         cout << "Nro da turma: " << *(TurmaAD + (Lin * 3)) << endl;
24         cout << "Qtde de alunos: " << *(TurmaAD + (Lin * 3) + 1) << endl;
25         cout << "Qtde Discliplinas: " << *(TurmaAD + (Lin * 3) + 2) << endl;
26         cout << endl << endl;
27     }
28 }
```

- Operador seta (->)
  - O operar seta, “->“, é utilizado quando o ponteiro está acessando os elementos de uma estrutura.

```
1  #include <iostream>
2  using namespace std;
3
4  struct DADOS_ALUNO{
5      int CodAluno;
6      char Nome[100];
7      int Turma;
8  };
```

- Exemplo

```
10  int main(){
11      DADOS_ALUNO AlunoA ;
12      DADOS_ALUNO *ptrAluno;
13      cout << "Digite o codigo do aluno: ";
14      cin >> AlunoA.CodAluno;
15      cout << "Digite o nome do aluno: ";
16      cin >> AlunoA.Nome;
17      cout << "Digite a turma: ";
18      cin >> AlunoA.Turma;
19
20      ptrAluno = &AlunoA;
21      cout << "Codigo do Aluno: " << ptrAluno->CodAluno <<endl;
22      cout << "Nome: " << ptrAluno->Nome <<endl;
23      cout << "Turma: " << ptrAluno->Turma <<endl;
24  }
```

- Atribuição de valores: `ptrAluno->CodAluno = 150;`

- Exemplo função

```
1  #include <iostream>
2  using namespace std;
3
4  struct DADOS_ALUNO{
5      int CodAluno;
6      char Nome[100];
7      int Turma;
8  };
9
10 void Imprimir(DADOS_ALUNO *ptrAluno);
11
12 int main(){
13     DADOS_ALUNO AlunoA ;
14     cout << "Digite o codigo do aluno: ";
15     cin >> AlunoA.CodAluno;
16     cout << "Digite o nome do aluno: ";
17     cin >> AlunoA.Nome;
18     cout << "Digite a turma: ";
19     cin >> AlunoA.Turma;
20     Imprimir(&AlunoA);
21 }
```

```
23 void Imprimir(DADOS_ALUNO *ptrAluno){
24     cout << "Codigo do Aluno: " << ptrAluno->CodAluno <<endl;
25     cout << "Nome: " << ptrAluno->Nome <<endl;
26     cout << "Turma: " << ptrAluno->Turma <<endl;
27 }
```

ALOCAÇÃO DINÂMICA DE MEMÓRIA

- Ao desenvolver um programa, principalmente utilizando estrutura de dados, pode ser difícil prever com antecedência quantos elementos terá esta estrutura.
- Este espaço que precisamos reservar em tempo de execução é chamada de memória alocada dinamicamente.

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      // Aloca memória
6      int *ptrNroAlunos = new int;
7      *ptrNroAlunos = 150;
8      cout << "Quantidade de alunos: " << *ptrNroAlunos ;
9      //Desaloca memória
10     delete ptrNroAlunos;
11 }
```

ALOCAÇÃO DINÂMICA DE MEMÓRIA

- O operador "new" retorna o endereço onde começa o bloco de memória que foi reservado, desta forma podemos colocá-lo num ponteiro.
  - **new** é um operador cuja função é alocar dinamicamente memória

- Se não necessitamos mais dos dados que estão num endereço de memória dinâmica, devemos apagá-la!
  - Para liberar essa memória utiliza-se o operador **delete** – este operador entrega ao sistema operacional ou sistema de gerenciamento de memória os espaços de memória reservados dinamicamente.
  - O operador delete não apaga o ponteiro mas sim a memória para onde o ponteiro aponta

ALOCAÇÃO DINÂMICA DE MEMÓRIA

- Exemplo:

```
1  #include <iostream>
2  #include <string.h>
3  using namespace std;
4  struct DADOS_ALUNO{
5      int CodAluno;
6      char Nome[100];
7      int Turma;
8  };
```

```
10 int main(){
11     // Aloca memória
12     DADOS_ALUNO *prtAluno = new DADOS_ALUNO[3];
13     prtAluno[0].CodAluno = 10;
14     strcpy(prtAluno[0].Nome, "Maria");
15     prtAluno[0].Turma = 320;
16     prtAluno[1].CodAluno = 20;
17     strcpy(prtAluno[1].Nome, "José");
18     prtAluno[1].Turma = 320;
19     prtAluno[2].CodAluno = 30;
20     strcpy(prtAluno[2].Nome, "João");
21     prtAluno[2].Turma = 320;
22
23     for(int Ind = 0; Ind < 3; Ind++){
24         cout << "Código do aluno: " << prtAluno[Ind].CodAluno << endl;
25         cout << "Nome do aluno: " << prtAluno[Ind].Nome << endl;
26         cout << "Turma: " << prtAluno[Ind].Turma << endl;
27         cout << endl << endl;
28     }
29     //Desaloca memória
30     delete []prtAluno;
31 }
```

ALOCAÇÃO DINÂMICA DE MEMÓRIA

- Exemplo:

```
1   #include <iostream>
2   using namespace std;
3   int main ()
4   {
5     int nElementos;
6     cout << "Digite a quantidade de elementos:";
7     cin >> nElementos;
8     int * iPtr = new int[nElementos];     //colocamos um ponteiro no inicio da memória dinâmica
9
10    for (int i = 0; i < nElementos; i++) //Podemos preecher o espaço de memória da forma que quisermos
11    {
12      cout << "Digite o valor #" << i + 1 << " : ";
13      cin >> iPtr[i];
14    }
15
16    for (int i = 0; i < nElementos; i++)  //Mostramos o que foi preenchido ...
17      cout << "Resultado valor #" << i + 1 << " e "<< iPtr[i] << endl;
18    delete iPtr;
19
20    return 0;
21  }
```

EXERCÍCIOS

- Faça um programa que preencha um vetor de 5 posições usando um ponteiro e exiba o vetor preenchido também usando ponteiro, use funções para preencher e exibir o vetor

- Faça um programa que leia um valor n e crie dinamicamente um vetor de n elementos e passe esse vetor para uma função que vai ler os elementos desse vetor. Depois, no programa principal, o vetor preenchido deve ser impresso. Além disso, antes de finalizar o programa, deve-se liberar a área de memória alocada.