Estrutura de Dados 1

Alocação Dinâmica

Profª Juliana Franciscani

Roteiro

01 Alocação Dinâmica

02 Exemplos

03 Vetor e Matriz

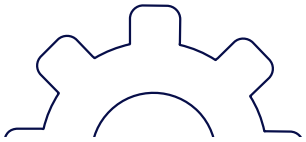
04 Struct

05 Exercícios

Introdução

- Necessário em um programa reservar espaço para as informações que serão processadas.

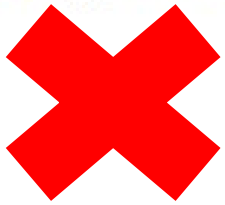
- Uso de variáveis
  - Uma variável é uma posição de memória que armazena uma informação que pode ser modificada pelo programa.
  - Ela deve ser definida antes de ser usada.
  - Alocação Automática *X* Alocação Dinâmica



Nem sempre é possível saber, em tempo de execução, o quanto de memória um programa irá precisar.

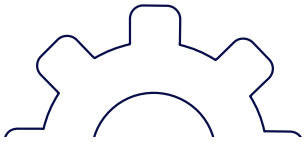
Faça um programa para cadastrar o preço de **N** produtos, em que **N** é um valor informado pelo usuário

```
int N, i;
double produtos[N];
```



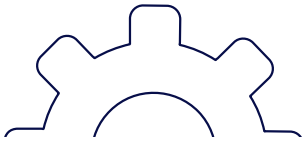
```
int N,i;

cin>> N;

double produto[N];
```

Funciona, mas não é o mais indicado



Alocação Dinâmica

- Permite ao programador criar “variáveis” em tempo de execução
- Alocar memória para novas variáveis quando o programa está sendo executado
- Quantidade de memória é alocada sob demanda, ou seja, quando o programa precisa
- Menos desperdício de memória:
  - Espaço é reservado até liberação explícita (ao desalocar não é mais acessível)
  - Espaço alocado e não liberado explicitamente é automaticamente liberado ao final da execução



Alocando memória

```
int *ptr;
int p[5];
ptr = p;
```

Alocando 5 posições de memória em int *ptr

Memória		
posição	**variável**	**conteúdo**
001		
002		
003	int *ptr	NULL
004		
005		
006		
007		
008		
009		
010		

Memória		
posição	**variável**	**conteúdo**
001		
002		
003	int *ptr	005
004		
005	p[0]	10
006	p[1]	9
007	p[2]	3
008	p[3]	2
009	p[4]	7
010		

Alocação Dinâmica

A linguagem C++ usa funções para o sistema de alocação dinâmica com os operadores **NEW** e **DELETE**.

NEW:

- Alocar memória dinamicamente para uma variável.
- Pode ser usado para alocar qualquer tipo de variável

```
tipo ponteiro = new tipo;
```

DELETE:

- utilizado para liberar a memória

que foi alocada pelo **new.**

```
delete ponteiro;
```

```
delete ptr;
```

```
int *ptr = new int;
int *ptr = new int ();
int *ptr = new int {};
int *ptr = new int (100);
int *ptr = new int {100};
int *ptr = new (std::nothrow) int;
```

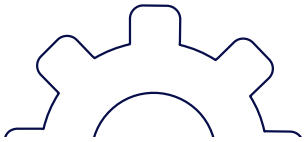
NEW e DELETE

```
#include<iostream>

using namespace std;

int main(){
    int *ptr = new (std::nothrow) int;
    if(!ptr)
        cout<<"Erro: Memória insuficiente!" << endl;
    else{
        *ptr = 100;
        cout << "Conteúdo: " << *ptr;
        delete ptr;
    }
    return 0;
}
```

Alocação Dinâmica

- Não vale a pena usar alocação dinâmica para variáveis básicas, tipo um int, um float, um char, uma string.

- Qual o propósito?
  - Alocar vetor dinâmico
  - Alocar variáveis de blocos, mais complexas.



Alocação dinâmica - Arrays

NEW []:

```
int *vetor = new int[3];

int *vetor = new int[3] ();
int *vetor = new int[3] {};

int *vetor = new int[3] {1,0,4};

int *vetor = new int[3] {8};

int **vetor2 = new int*[4];
```

DELETE[]:

```
delete[] vetor;
delete[] vetor2;
```

O delete[] funciona da mesma forma para qualquer tipo de vetor alocado. Mesmo se for vetor de ponteiros...

```cpp
#include<iostream>
#include<windows.h>
using namespace std;
int main(){
    SetConsoleCP(1252);
    SetConsoleOutputCP(1252);
    int *vetor, soma=0, qt;
    cout << "Informe a quantidade de números: ";
    cin >> qt;
    vetor = new (std::nothrow) int[qt];
    if(!vetor)
        cout<<"Erro: Memória insuficiente!" << endl;
    else{
        for(int i=0;i<qt;i++){
            cout << "Informe um valor: ";
            cin >> vetor[i];
            soma = soma + vetor[i];
        }
        cout<< "O total é: "<< soma;
        delete[] vetor;
    }
    return 0;
```

Alocação dinâmica - MATRIZ

```
int **p, i, j, N =2, M=3;

p = new int*[N];

for(i=0;i<N;i++){
     p[i]= new int[M];
     for(j=0;j<M;j++)
          cin >> p[i][j];
}
for(i=0;i<N;i++){
     for(j=0;j<M;j++)
          cout << p[i][j] << " " ;
     cout << endl;
}
 for(j=0;j<M;j++)
     delete[] p[j];

delete[] p;
```

Alocação de struct

- Assim como os tipos básicos, também é possível fazer a alocação dinâmica de estruturas.

- As regras são exatamente as mesmas para a alocação de uma **struct**.

- Podemos fazer a alocação de
  - uma única **struct**
  - um array de **structs**

Alocação de struct

- Para alocar uma única **struct**
  - Um ponteiro para **struct** receberá o **new**
  - Utilizamos o **operador seta** para acessar o conteúdo
  - Usamos **delete** para liberar a memória alocada

```
#include<iostream>
#include<windows.h>
using namespace std;
struct Cadastro{
    string nome;
    int idade;
};
int main(){
    SetConsoleCP(1252);
    SetConsoleOutputCP(1252);

    Cadastro *cad = new Cadastro;

    cad->nome="João";
    cad[0].idade = 10;
    cout << cad->nome << endl;
    cout << cad->idade << endl;

    delete cad;
    return 0;
```

Alocação de struct

- Alocar uma struct: um ponteiro para **struct** receberá o **new**
- Para acessar o conteúdo:
  - **colchetes [ ] e o ponto .**
  - **Seta**
- Usa-se **delete** para liberar a memória alocada

Pode-se usar da mesma forma o vetor de struct.

```
struct Cadastro{
    string nome;
    string doc;
};

int main(){
    Cadastro *cad1 = new Cadastro;
    cad1->nome = "Maria";
    cad1[0].doc = "1234";

    cout << "Informações do Cadastro 1\n";
    cout << cad1->nome << endl;
    cout << cad1->doc << endl;

    delete cad1;
```

```cpp
struct Cadastro{
    string nome;
    string doc;
};
int main(){
    Cadastro *cad2 = new Cadastro[3];
    cout << "\nCadastro:\n";
    for(int i=0;i<3;i++){
        cout << "Nome: "; getline(cin,cad2->nome);
        cout << "Doc: ";  getline(cin,cad2->doc);
        cad2++;
        cout<<endl;
    }
    cout << "\nExibindo informações\n";
    cad2-=3;
    for(int i=0;i<3;i++){
        cout << i+1 << "º Cadastro:\n"
        cout<< cad2[i].nome << endl;
        cout << cad2[i].doc << endl;
        cout<<endl;
    }
    delete[] cad2;
```

Funções e structs

```
struct Cadastro{
    string nome;
    int idade;
};

void cadastroAluno(Cadastro *aluno, int num);
void exibeCadastro(Cadastro *aluno, int num);

int main(){
    SetConsoleCP(1252);
    SetConsoleOutputCP(1252);
    int n=3;
    Cadastro *aluno = new Cadastro[n];
    cadastroAluno(aluno, n);
    exibeCadastro(aluno,n);
    delete[] aluno;

    return 0;
}
```

```
void cadastroAluno(Cadastro *aluno, int num){
    cout << "\nCadastro:\n";
    for(int i=0;i<num;i++){
        cout << "Nome: "; getline(cin,aluno[i].nome);
        cout << "Idade: ";  cin>> aluno[i].idade;
        cin.ignore();
        cout<<endl;
    }
}

void exibeCadastro(Cadastro *aluno, int num){
     cout << "\nExibindo informações\n";
    for(int i=0;i<num;i++){
        cout << i+1 << "º Cadastro:\n";
        cout<< aluno[i].nome << endl;
        cout<< aluno[i].idade << endl;
        cout<<endl;
    }
}
```

Exercícios

1. Elabore um programa para armazenar números (de 0 a 10) em dois vetores. Inicialmente necessário saber o tamanho dos vetores (peça para o usuário). Aloque dinamicamente os vetores e peça ao usuário que informe os valores de cada um deles. Crie e construa um terceiro vetor que seja o resultado da soma dos anteriores. Exiba os números contidos em cada um dos vetores.

2. Faça o exercício anterior usando função, deverá existir uma função para preencher, outra para somar e uma para exibir os valores.

3. Elabore um programa em C++ que seja capaz de criar uma struct que contenha o nome, ano inicial válido , cargo e salário de um funcionário de uma empresa. Crie dinamicamente o registro, preencha as informações (válidas) e depois as exiba.

4. Faça o exercício 3 utilizando função: uma para cadastro, outra para exibir as informações.

5. Elabore um programa que seja baseado nos exercícios 3 e 4, porém para o cadastro de 5 funcionários da empresa Crie uma função que exiba o nome e o cargo do funcionário que possua maior salário.

Referências

EDELWEISS, N,; GALANTE, R.. Estruturas de dados. Porto Alegre: Bookman, 2009.

SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3. ed. Rio de Janeiro: LTC, 2010.

ZIVIANI, N. Projeto de algoritmos: com implementações em Pascal e C. 3. ed. rev. e ampl. São Paulo: Cengage Learning, 2011.

Aulas e vídeo aulas do professor André Backes: https://www.facom.ufu.br/~backes/