

C++

Básico ao Avançado

O monstro do armário

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

1 Problemática

2 Abrindo espaços

3 Mais problemas

4 !Abrindo espaços

5 Outras situações

6 Hora de brincar

99999999...

99999999...

- Declarar um vetor de tamanho N

99999999...

- Declarar um vetor de tamanho N
- N é 9

99999999...

- Declarar um vetor de tamanho N
- N é 99

99999999...

- Declarar um vetor de tamanho N
- N é 999

99999999...

- Declarar um vetor de tamanho N
- N é 9999

99999999...

- Declarar um vetor de tamanho N
- N é 99999

99999999...

- Declarar um vetor de tamanho N
- N é 999999

99999999...

- Declarar um vetor de tamanho N
- N é 99999999

99999999...

- Declarar um vetor de tamanho N
- N é 99999999

99999999...

- Declarar um vetor de tamanho N
- N é 999999999...

99999999...

- Declarar um vetor de tamanho N
- N é 999999999...
- Vocês já tentaram fazer um vetor tão grande?

99999999...

- Declarar um vetor de tamanho N
- N é 999999999...
- Vocês já tentaram fazer um vetor tão grande?

```
int v1 [9] ;
int v2 [99] ;
int v3 [999] ;
int v4 [9999] ;
int v5 [99999] ;
int v6 [999999] ;
int v7 [9999999] ;
```

9999999...

- Declarar um vetor de tamanho N
- N é 999999999...
- Vocês já tentaram fazer um vetor tão grande?

```
int v1 [9] ;  
int v2 [99] ;  
int v3 [999] ;  
int v4 [9999] ;  
int v5 [99999] ;  
int v6 [999999] ;  
int v7 [9999999] ;
```

- Isso *deveria* dar errado

9999999...

- Declarar um vetor de tamanho N
- N é 999999999...
- Vocês já tentaram fazer um vetor tão grande?

```
int v1 [9] ;  
int v2 [99] ;  
int v3 [999] ;  
int v4 [9999] ;  
int v5 [99999] ;  
int v6 [999999] ;  
int v7 [9999999] ;
```

- Isso *deveria* dar errado (pode ser que não dê)

Quantos mesmo?

Quantos mesmo?

- Declarar um vetor de tamanho N

Quantos mesmo?

- Declarar um vetor de tamanho N
- É decidido pelo usuário em tempo de execução

Quantos mesmo?

- Declarar um vetor de tamanho N
- É decidido pelo usuário em tempo de execução
- *Naonde?*

Quantos mesmo?

- Declarar um vetor de tamanho N
- É decidido pelo usuário em tempo de execução
- *Naonde?*
- Tempo de compilação \times Tempo de execução

Quanto mesmo?

- Declarar um vetor de tamanho N
- É decidido pelo usuário em tempo de execução
- *Naonde?*
- Tempo de compilação \times Tempo de execução
- *Não é só colocar o tamanho como variável e fazer o vetor desse tamanho?*

Quantos mesmo?

- Declarar um vetor de tamanho N
- É decidido pelo usuário em tempo de execução
- *Naonde?*
- Tempo de compilação \times Tempo de execução
- *Não é só colocar o tamanho como variável e fazer o vetor desse tamanho?*

```
//...  
int N;  
//...  
int vetor[N];  
//...
```


Quantos mesmo?

- Declarar um vetor de tamanho N
- É decidido pelo usuário em tempo de execução
- *Naonde?*
- Tempo de compilação \times Tempo de execução
- *Não é só colocar o tamanho como variável e fazer o vetor desse tamanho?*

```
//...  
int N;  
//...  
int vetor[N];  
//...
```

- Então

Quantos mesmo?

- Declarar um vetor de tamanho N
- É decidido pelo usuário em tempo de execução
- *Naonde?*
- Tempo de compilação \times Tempo de execução
- *Não é só colocar o tamanho como variável e fazer o vetor desse tamanho?*

```
//...  
int N;  
//...  
int vetor[N];  
//...
```

- Então, não deveria ser...

Quantos mesmo?

- Declarar um vetor de tamanho N
- É decidido pelo usuário em tempo de execução
- *Naonde?*
- Tempo de compilação \times Tempo de execução
- *Não é só colocar o tamanho como variável e fazer o vetor desse tamanho?*

```
//...  
int N;  
//...  
int vetor[N];  
//...
```

- Então, não deveria ser...
- O tamanho do vetor *estático* deve ser uma constante

Estático

Estático

- Quando declaramos um vetor com esta sintaxe, ele é estático

Estático

- Quando declaramos um vetor com esta sintaxe, ele é estático
- O vetor estático é declarado na *stack*

Estático

- Quando declaramos um vetor com esta sintaxe, ele é estático
- O vetor estático é declarado na *stack*
- A *stack* é o espaço de memória reservado para o programa

Estático

- Quando declaramos um vetor com esta sintaxe, ele é estático
- O vetor estático é declarado na *stack*
- A *stack* é o espaço de memória reservado para o programa
- Ela não é muito grande, então vetores grandes não cabem

Estático

- Quando declaramos um vetor com esta sintaxe, ele é estático
- O vetor estático é declarado na *stack*
- A *stack* é o espaço de memória reservado para o programa
- Ela não é muito grande, então vetores grandes não cabem
- O resto da memória do computador é chamada *heap*

Estático

- Quando declaramos um vetor com esta sintaxe, ele é estático
- O vetor estático é declarado na *stack*
- A *stack* é o espaço de memória reservado para o programa
- Ela não é muito grande, então vetores grandes não cabem
- O resto da memória do computador é chamada *heap*
- Podemos utilizar a *heap*?

Estático

- Quando declaramos um vetor com esta sintaxe, ele é estático
- O vetor estático é declarado na *stack*
- A *stack* é o espaço de memória reservado para o programa
- Ela não é muito grande, então vetores grandes não cabem
- O resto da memória do computador é chamada *heap*
- Podemos utilizar a *heap*?
- Sim!

Estático

- Quando declaramos um vetor com esta sintaxe, ele é estático
- O vetor estático é declarado na *stack*
- A *stack* é o espaço de memória reservado para o programa
- Ela não é muito grande, então vetores grandes não cabem
- O resto da memória do computador é chamada *heap*
- Podemos utilizar a *heap*?
- Sim! Mas para isso precisamos de um operador. . .

Dinâmico

Dinâmico

■ Operador unário

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)
- É fácil de usar

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)
- É fácil de usar, diferente do irmão mais velho

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)
- É fácil de usar, diferente do irmão mais velho
- O simbolo é uma palavra-chave:

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chave: `new`

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)
- É fácil de usar, diferente do irmão mais velho
- O simbolo é uma palavra-chave: `new`
- A sintaxe é amigável

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)
- É fácil de usar, diferente do irmão mais velho
- O simbolo é uma palavra-chave: `new`
- A sintaxe é amigável

```
//...  
new <tipo>;           //Alocar variável  
new <tipo>[<tamanho>]; //Alocar vetor  
//...
```

Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chave: `new`
- A sintaxe é amigável

```
//...  
new <tipo>;           //Alocar variável  
new <tipo>[<tamanho>]; //Alocar vetor  
//...
```

```
//...  
int N;  
//...  
int *vetor(new int[N]);  
//...
```


Dinâmico

- Operador unário
- Recebe o tipo a ser alocado
- Ele retorna um endereço de memória (sim, ponteiros)
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chave: `new`
- A sintaxe é amigável

```
//...  
new <tipo>;           //Alocar variável  
new <tipo>[<tamanho>]; //Alocar vetor  
//...
```

```
//...  
int N;  
//...  
int *vetor(new int[N]);  
//...
```

- Cuidado com o NULL

Glutão

Glutão

- Legal, sabemos pegar mais espaço

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba
- Se acabar acontece um problemas

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba
- Se acabar acontece um problemas
- Janelas 98 ®

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba
- Se acabar acontece um problemas
- Janelas 98 ®, alocava mais memória pra sempre

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba
- Se acabar acontece um problemas
- Janelas 98 ®, alocava mais memória pra sempre
- Resultado:

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba
- Se acabar acontece um problemas
- Janelas 98 ®, alocava mais memória pra sempre
- Resultado: Ela acabava

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba
- Se acabar acontece um problemas
- Janelas 98 ®, alocava mais memória pra sempre
- Resultado: Ela acabava, ou seja, BSoD!

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba
- Se acabar acontece um problemas
- Janelas 98 ®, alocava mais memória pra sempre
- Resultado: Ela acabava, ou seja, BSoD!
- Precisamos devolver a memória que pegamos!

Glutão

- Legal, sabemos pegar mais espaço
- Então é possível pegar todo o espaço
- Que tal testar?
- A memória do computador é finita
- Se pegar muito, acaba
- Se acabar acontece um problemas
- Janelas 98 ®, alocava mais memória pra sempre
- Resultado: Ela acabava, ou seja, BSoD!
- Precisamos devolver a memória que pegamos!
- Temos outro operador!

Ainda dinâmico

Ainda dinâmico

■ Operador unários

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno
- É fácil de usar

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno
- É fácil de usar, diferente do irmão mais velho

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chamada:

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chamada: `delete`

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chamada: `delete`
- Sintaxe amigável

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chamada: `delete`
- Sintaxe amigável

```
//...  
delete <nome>;      //Desaloca endereço apontado  
delete[] <nome>;    //Desaloca vetor alocado no endereço  
//...
```

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chamada: `delete`
- Sintaxe amigável

```
//...  
delete <nome>;      //Desaloca endereço apontado  
delete[] <nome>;    //Desaloca vetor alocado no endereço  
//...
```

```
//...  
int N;  
//...  
int *vetor(new int[N]);  
//...  
delete[] vetor;  
//...
```

Ainda dinâmico

- Operador unários
- Recebe o endereço a ser desalocado
- Não tem retorno
- É fácil de usar, diferente do irmão mais velho
- O símbolo é uma palavra-chamada: `delete`
- Sintaxe amigável

```
//...  
delete <nome>;      //Desaloca endereço apontado  
delete[] <nome>;    //Desaloca vetor alocado no endereço  
//...
```

```
//...  
int N;  
//...  
int *vetor(new int[N]);  
//...  
delete[] vetor;  
//...
```

- Cuidado para não desalocar o endereço errado

Alocação de tipos abstratos

Alocação de tipos abstratos

- Que tal um tipo abstrato que contenha um ponteiro para ele mesmo?

Alocação de tipos abstratos

- Que tal um tipo abstrato que contenha um ponteiro para ele mesmo?
- *É o que?*

Alocação de tipos abstratos

- Que tal um tipo abstrato que contenha um ponteiro para ele mesmo?
- *É o que?*

```
struct meuTipo
{
    int el;
    meuTipo *p;
};
//...
meuTipo base;
base.p = new meuTipo;
//...
```

Alocação de tipos abstratos

- Que tal um tipo abstrato que contenha um ponteiro para ele mesmo?
- *É o que?*

```
struct meuTipo  
{  
    int el;  
    meuTipo *p;  
};  
//...  
meuTipo base;  
base.p = new meuTipo;  
//...
```

- Aí que usamos o operador de acesso indireto a membro (->)

Alocação de tipos abstratos

- Que tal um tipo abstrato que contenha um ponteiro para ele mesmo?
- *É o que?*

```
struct meuTipo
{
    int el;
    meuTipo *p;
};
//...
meuTipo base;
base.p = new meuTipo;
//...
```

- Aí que usamos o operador de acesso indireto a membro (->)
- Esse código é a base para *estrutura de dados*

Vamos testar!