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

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?

- Declarar um vetor de tamanho N
- É decidido pelo usário 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

- 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

- 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 amigáve

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

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

- Cuidado com o `NULL`

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

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

- 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!