ALOCAÇÃO DE MEMÓRIA

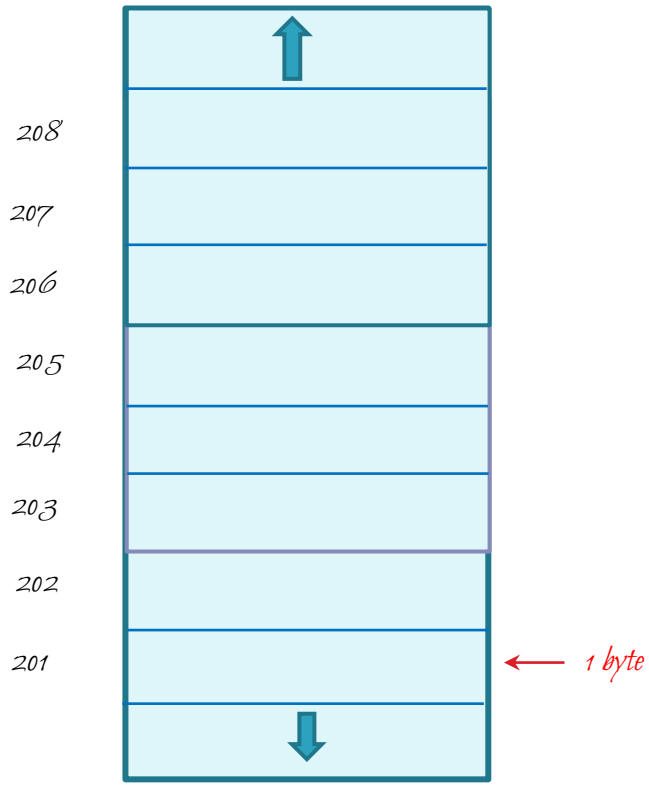
Estrutura de Dados I – ED1

Ponteiros

int 4 bytes

char 1 byte

float 4 bytes



Ponteiros

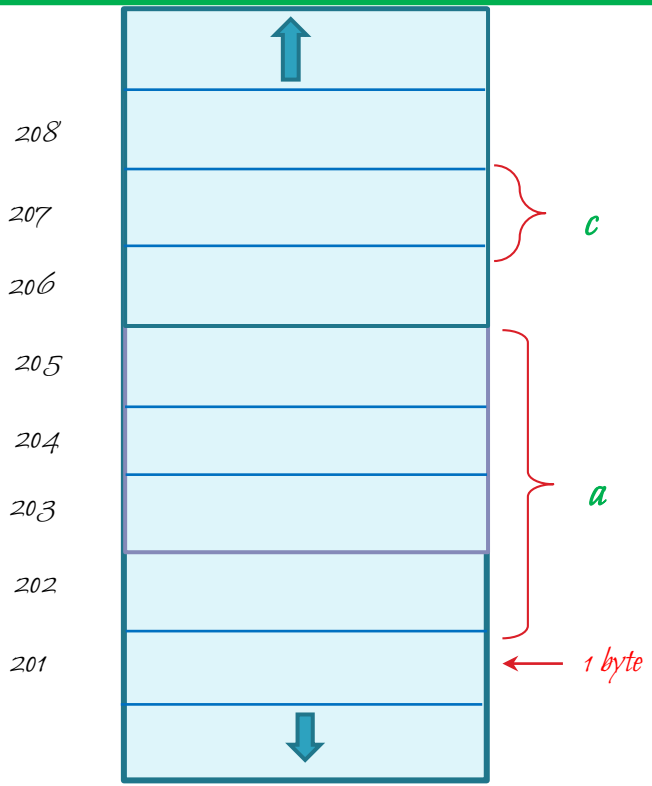
int ? 4 bytes

char ? 1 byte

float ? 4 bytes

int a;

char c;

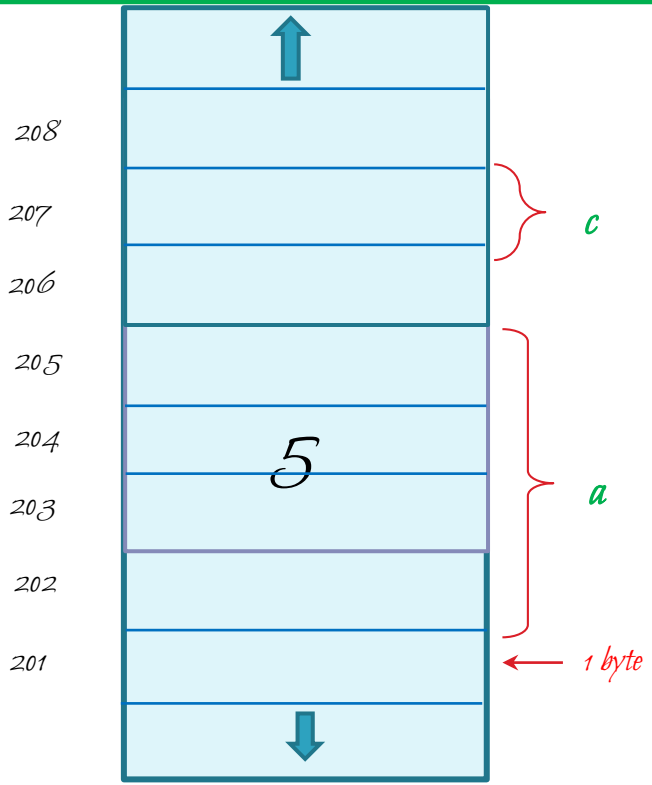


a int 202

c char 207

Ponteiros


```
int a;
char c;
a= 5;
```

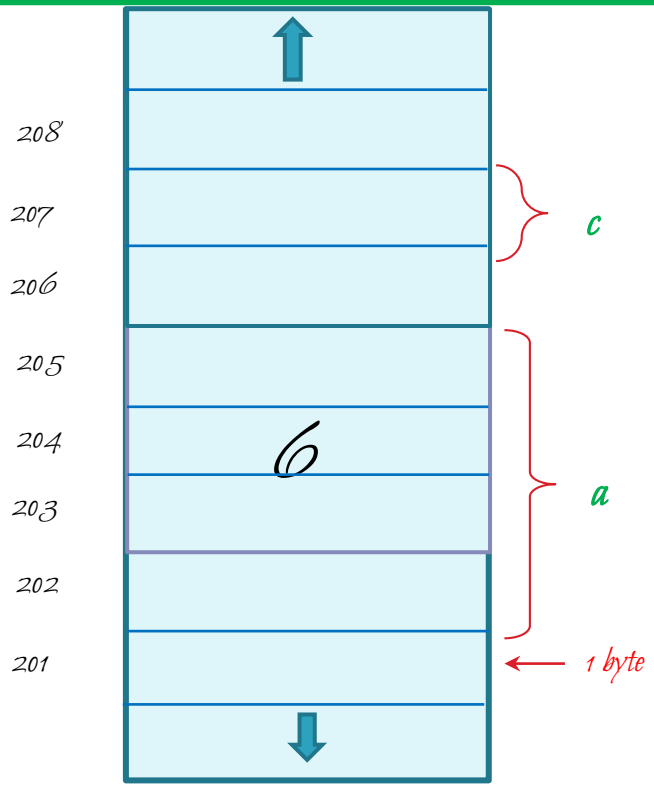


a int 202

c char 207

Ponteiros


```
int a;
char c;
a= 5;
....
a++;
```



a int 202

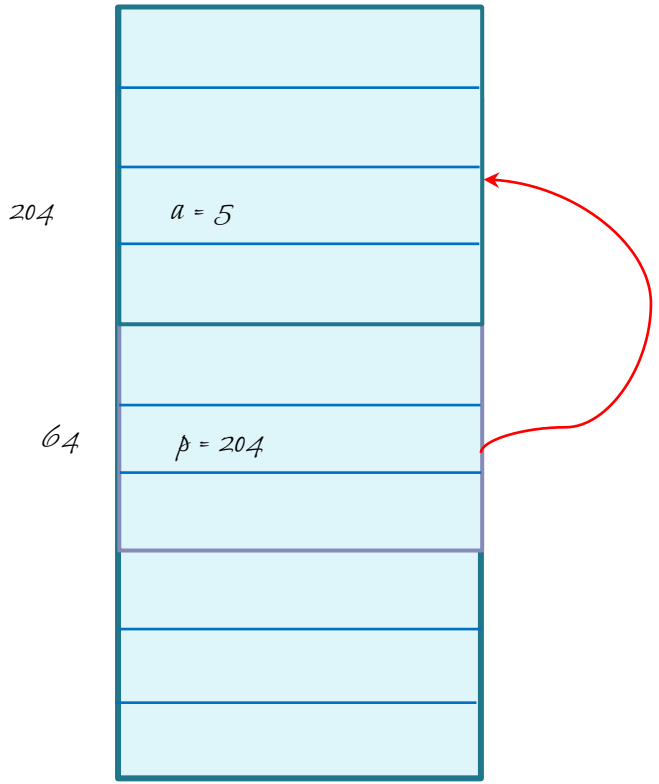
c char 207

Ponteiros

- Um ponteiro é uma variável capaz de armazenar um endereço de memória ou o endereço de outra variável
- Exemplo de declaração:
  - int *p;
  - float *pParaFloat;
- ***** : indica ao compilador que a variável guardará um endereço da memória; Neste endereço da memória haverá um valor do tipo especificado (tipo_do_ponteiro)
- **&** : fornece o endereço de memória onde está armazenada uma variável

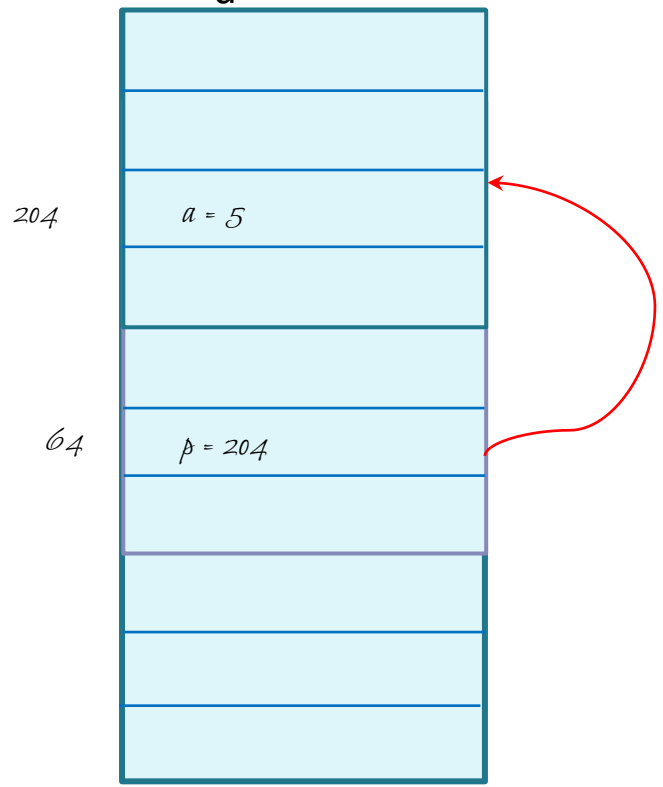
Ponteiros


```
int a;
int *p;
p = &a;
a = 5;
```



Ponteiros


```
5   int main()
6   {
7       int a;
8       int *p;
9
10      p = &a;
11      a=5;
12
13      cout << "\n p = " << p << endl; //204
14      cout << "\n &p = " << &p << endl; //64
15      cout << "\n &a = " << &a << endl; //204
16      cout << "\n *p = " << *p << endl; //5
17
18      *p = 8;
19      cout << "\n a = " << a << endl; //8
20  }
```



Ponteiros


```
 5  int main()
 6  {
 7      setlocale(LC_ALL,"Portuguese");
 8
 9      int x;
10      int *ptr;
11
12      ptr = &x;
13      cout << "\n O endereço de X é: = " << ptr << endl;
14  }
```

Ponteiros


```
 5  int main()
 6  {
 7      setlocale(LC_ALL,"Portuguese");
 8
 9      int x;
10      int *ptr;
11
12      x=5;
13      ptr = &x;
14
15      cout << "O valor da variável X é: = " << *ptr << endl;
16
17      *ptr = 10;
18      cout << "Agora, X vale: " << *ptr << endl;
19      cout << "Agora, X vale: " << x << endl;
20  }
```

Alocação de Memória

- Um programa de computador precisa utilizar memória para ser executado
- No início da sua execução, começa a solicitar memória ao sistema operacional, ou seja, faz a alocação de memória necessária para a sua execução
- A alocação de memória no computador pode ser dividida em dois grupos principais:
  - Alocação Estática
  - Alocação Dinâmica

Alocação de Memória Estática

- O espaço de memória, que as variáveis irão utilizar durante a execução do programa, é definido no processo de compilação
- Não é possível alterar o tamanho desse espaço durante a execução do programa
- Os dados tem um tamanho fixo e estão organizados sequencialmente na memória do computador
- Ex.: as variáveis globais e arrays

Alocação de Memória Estática

- Exemplos:
  - Espaço reservado para um valor do tipo char.
    - **char a;**
  - Espaço reservado para dez valores do tipo int. O int ocupa 4 bytes na memória, portanto 4x10=40 bytes.
    - **int vetor[10];**
  - Espaço reservado para nove(3x3) valores do tipo double. O double ocupa 8 bytes na memória, portanto 3x3x8=72 bytes
    - **double matriz[3][3];**

Arquitetura da Memória

- 4 segmentos

Memória dinâmica(alocável) → Heap

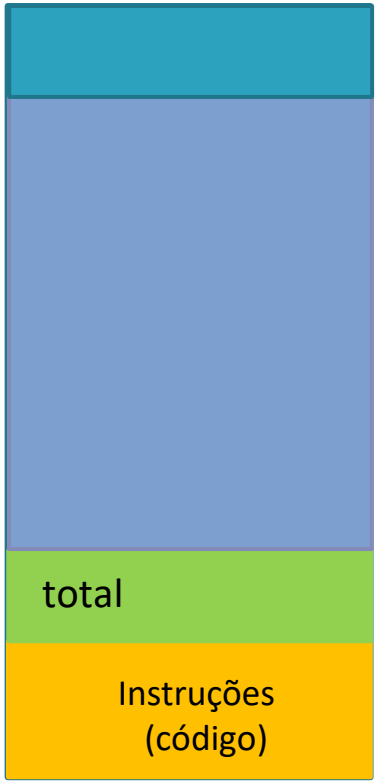
Chamadas de funções e variáveis locais
(o tamanho é calculado quando o programa é compilado) → Stack (pilha)

Variáveis globais/estáticas → Var global

Instruções do programa → Instruções (código)

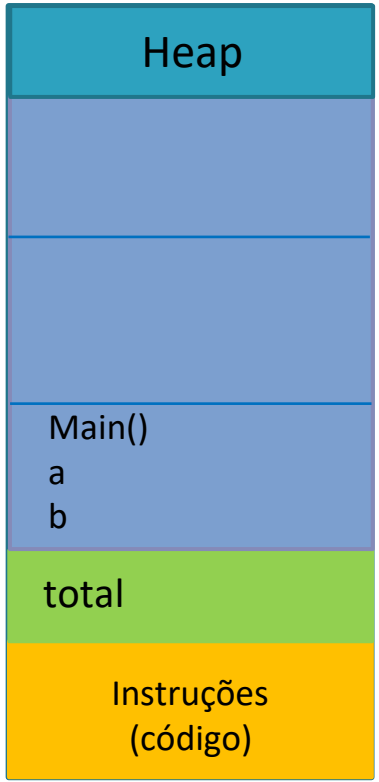
Arquitetura da Memória


```
1  #include <iostream>
2  using namespace std;
3
4  int total;
5
6  int square(int x)
7  {
8      return x*x;
9  }
10 int squareOfSum(int x, int y)
11 {
12     int z = square(x+y);
13     return z;
14 }
15 int main()
16 {
17
18     int a = 2;
19     int b = 3;
20     total = squareOfSum(a, b);
21     cout << "Total: " << total << endl;
22 }
23
```



Arquitetura da Memória

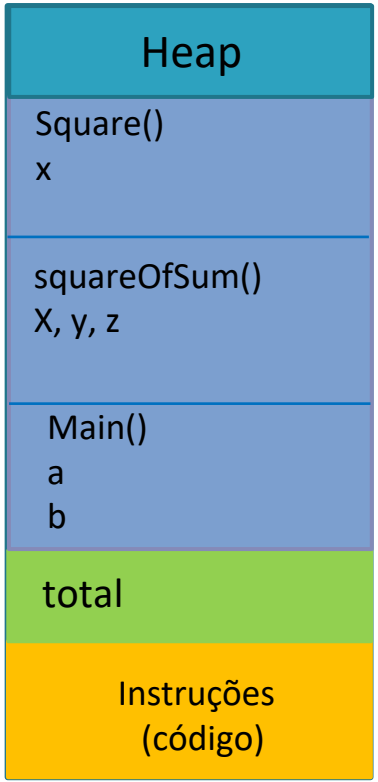

```
1  #include <iostream>
2  using namespace std;
3
4  int total;
5
6  int square(int x)
7  {
8      return x*x;
9  }
10 int squareOfSum(int x, int y)
11 {
12     int z = square(x+y);
13     return z;
14 }
15 int main()
16 {
17
18     int a = 2;
19     int b = 3;
20     total = squareOfSum(a, b);
21     cout << "Total: " << total << endl;
22 }
23
```



global

Arquitetura da Memória


```
1  #include <iostream>
2  using namespace std;
3
4  int total;
5
6  int square(int x)
7  {
8      return x*x;
9  }
10 int squareOfSum(int x, int y)
11 {
12     int z = square(x+y);
13     return z;
14 }
15 int main()
16 {
17
18     int a = 2;
19     int b = 3;
20     total = squareOfSum(a, b);
21     cout << "Total: " << total << endl;
22 }
23
```



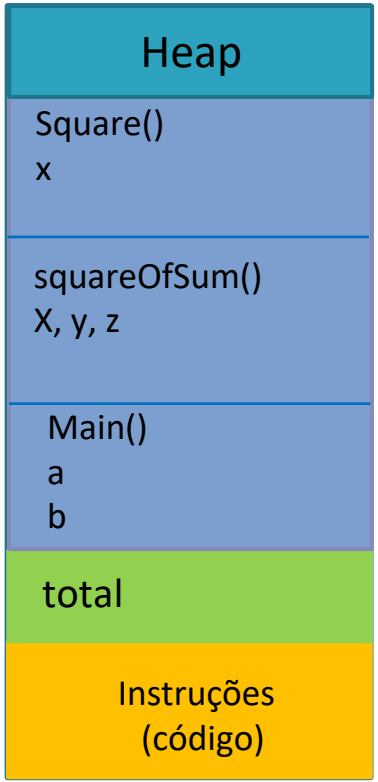
heap

stack

global

Arquitetura da Memória


```
1  #include <iostream>
2  using namespace std;
3
4  int total;
5
6  int square(int x)
7  {
8      return x*x;
9  }
10 int squareOfSum(int x, int y)
11 {
12     int z = square(x+y);
13     return z;
14 }
15 int main()
16 {
17
18     int a = 2;
19     int b = 3;
20     total = squareOfSum(a, b);
21     cout << "Total: " << total << endl;
22 }
23
```



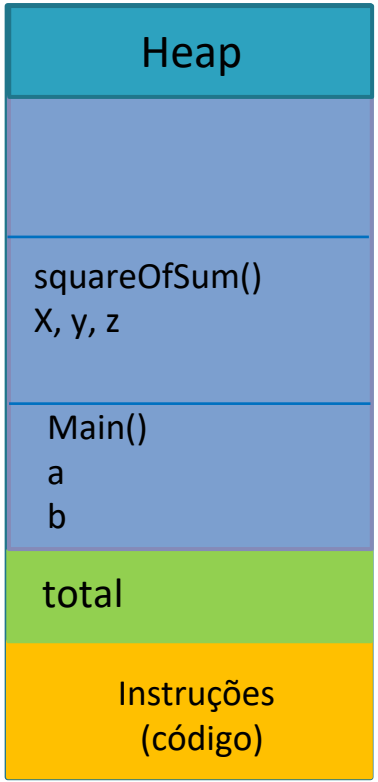
heap

stack

global

Arquitetura da Memória


```
1  #include <iostream>
2  using namespace std;
3
4  int total;
5
6  int square(int x)
7  {
8      return x*x;
9  }
10 int squareOfSum(int x, int y)
11 {
12     int z = square(x+y);
13     return z;
14 }
15 int main()
16 {
17
18     int a = 2;
19     int b = 3;
20     total = squareOfSum(a, b);
21     cout << "Total: " << total << endl;
22 }
23
```



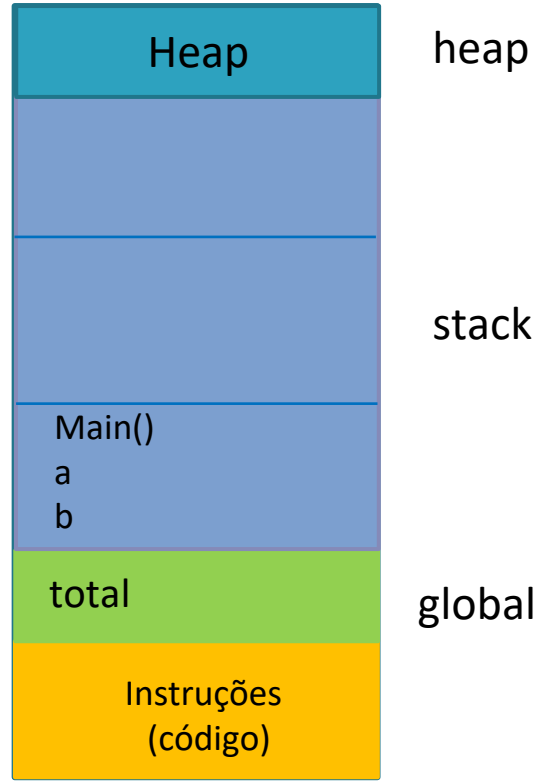
heap

stack

global

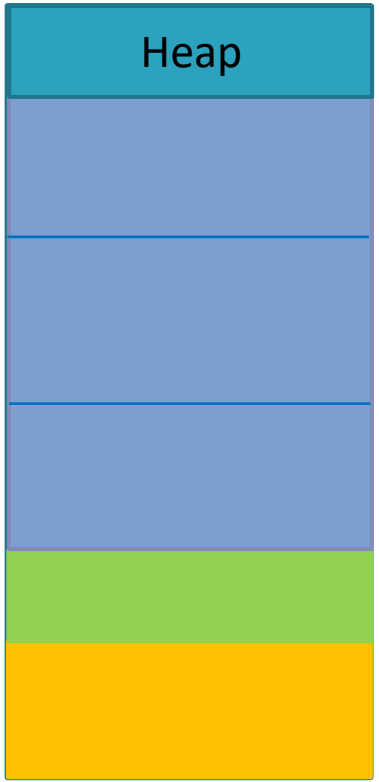
Arquitetura da Memória


```
1  #include <iostream>
2  using namespace std;
3
4  int total;
5
6  int square(int x)
7  {
8      return x*x;
9  }
10 int squareOfSum(int x, int y)
11 {
12     int z = square(x+y);
13     return z;
14 }
15 int main()
16 {
17
18     int a = 2;
19     int b = 3;
20     total = squareOfSum(a, b);
21     cout << "Total: " << total << endl;
22 }
23
```



Arquitetura da Memória


```
1  #include <iostream>
2  using namespace std;
3
4  int total;
5
6  int square(int x)
7  {
8      return x*x;
9  }
10 int squareOfSum(int x, int y)
11 {
12     int z = square(x+y);
13     return z;
14 }
15 int main()
16 {
17
18     int a = 2;
19     int b = 3;
20     total = squareOfSum(a, b);
21     cout << "Total: " << total << endl;
22 }
23
```



Alocação Dinâmica de Memória

- Nem sempre é possível saber, em tempo de execução, o quanto de memória um programa irá precisar
- Na alocação dinâmica o espaço de memória, que as variáveis irão utilizar durante a execução do programa, é definido enquanto o programa está em execução (sabe-se exatamente a quantidade necessária)
- Isso permite otimizar o uso da memória

Alocação Dinâmica de Memória

- C
  - Malloc
  - Realloc
  - Calloc
  - Free

- C++
  - New
  - delete

Alocação Dinâmica – C++

new

- retorna o endereço onde começa o bloco de memória que foi reservado
- Exemplos

```
Int *px;
Px = new int;
```

  - Outra forma

```
int *px= new int;
```

  - Alocando espaço e inicializando com o valor 4

```
Int *px= new int(4);
```

  - Alocando espaço e inicializando com o valor A

```
char *c= new char('A');
```

Alocação Dinâmica – C++

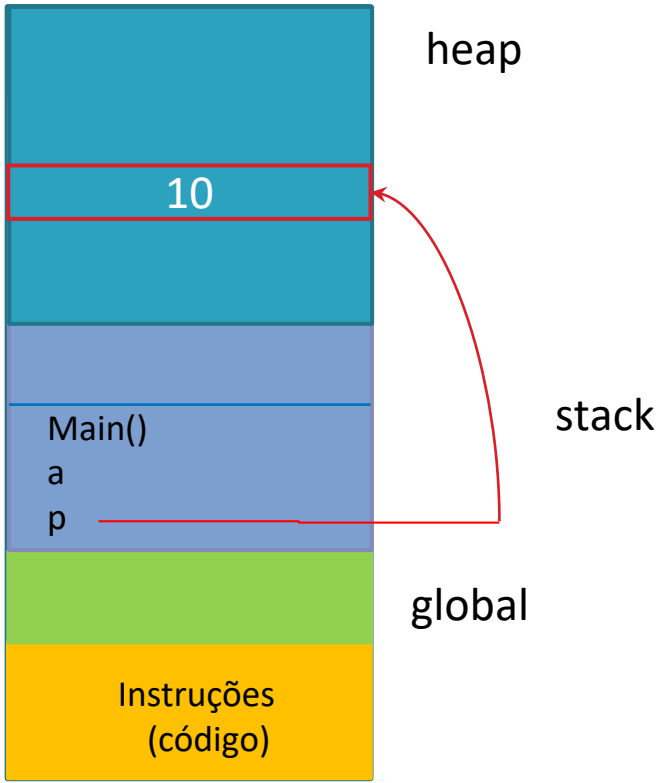
delete

- este operador entrega ao sistema operacional ou sistema de gerenciamento de memória os espaços de memória reservados dinamicamente
- Ex:

```
delete px;
```

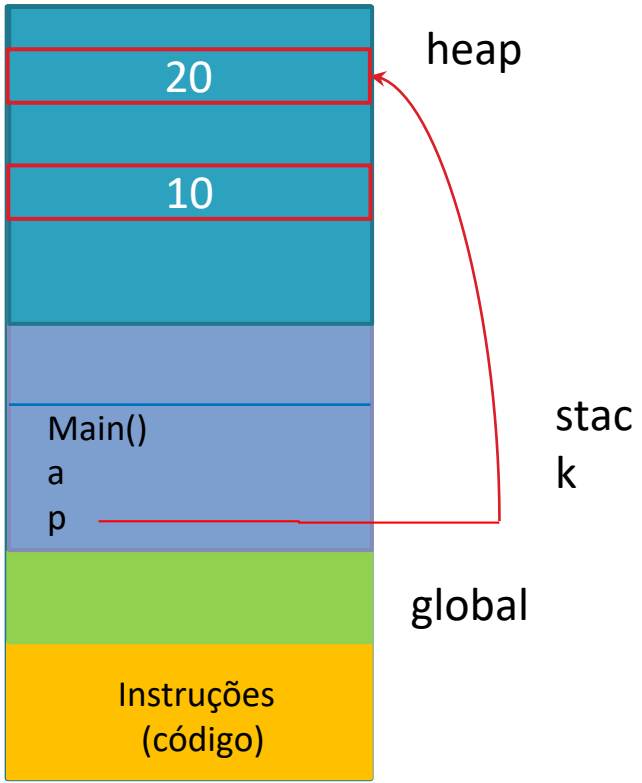
Arquitetura da Memória


```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6
7       int a;
8       int *p;
9       p = new int;
10      *p = 10;
11  }
```



Arquitetura da Memória

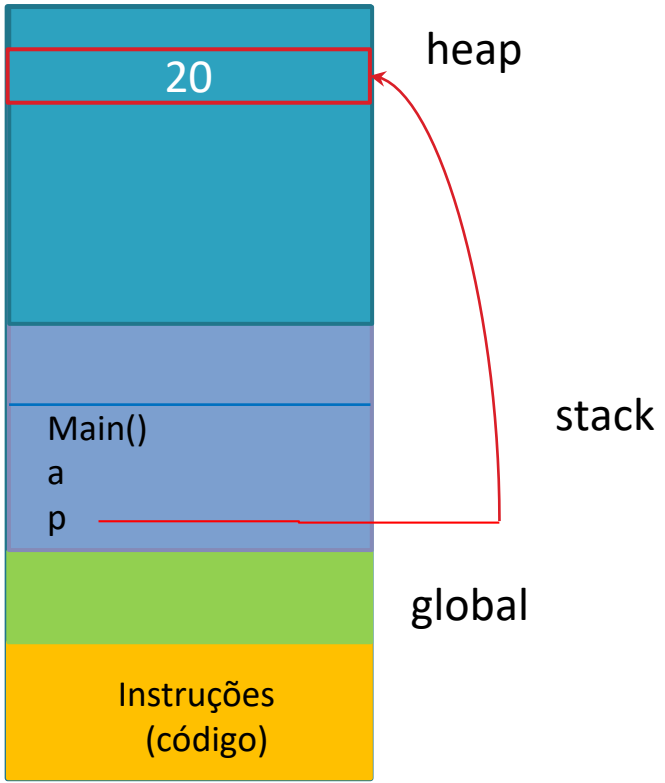

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6
7       int a;
8       int *p;
9
10      p = new int;
11      *p = 10;
12      cout << "p: " << p << endl;
13
14      p = new int;
15      *p = 20;
16      cout << "p: " << p << endl;
17  }
```



Arquitetura da Memória


```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6
7      int a;
8      int *p;
9
10     p = new int;
11     *p = 10;
12     cout << "p: " << p << endl;
13     delete p;
14
15     p = new int;
16     *p = 20;
17     cout << "p: " << p << endl;
18 }
```

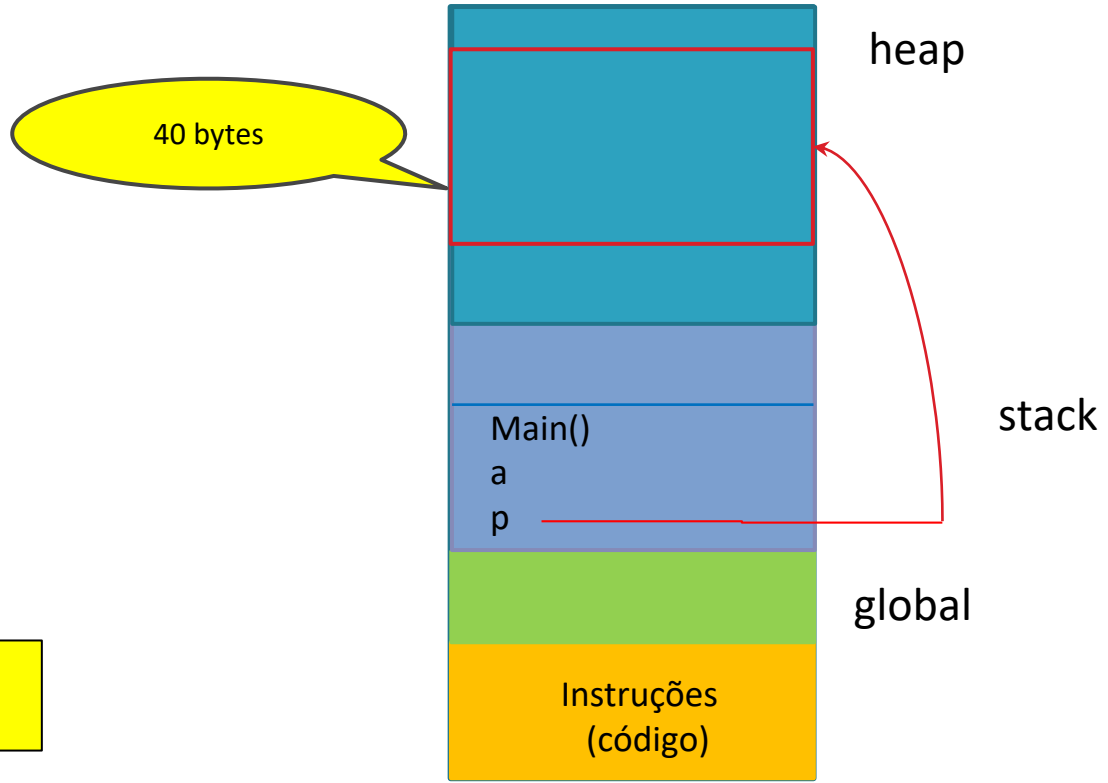
Liberando memória



Arquitetura da Memória


```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6
7       int a;
8       int *p;
9
10      p = new int;
11      *p = 10;
12      delete p;
13
14      p = new int[10];
15      delete[] p;
16  }
```

p[0] p[1] p[2]
*p *(p+1) *(p+2)



Alocação de Memória

Estática

- espaço de memória é definido durante o processo de compilação
- não é possível alterar o tamanho do espaço de memória que foi definido durante a compilação
- manter os dados organizados na memória, dispostos de forma linear e sequencial

Dinâmica

- espaço de memória é reservado durante a execução do programa
- espaço pode ser alterado dinamicamente durante a execução
- Dados estão são armazenados de forma fragmentada

IMPLEMENTAÇÃO

Chamada por valor


```
1  #include <iostream>
2  using namespace std;
3
4  void adiciona(int a)
5  {
6      a = a + 1;
7
8      cout << "Endereço de a (função): " << &a << endl;
9  }
10 int main()
11 {
12     setlocale(LC_ALL,"Portuguese");
13     int a = 10;
14     adiciona(a);
15
16     cout << "Endereço de a (main): " << &a << endl;
17     cout << "Valor de a: " << a << endl;
18 }
```

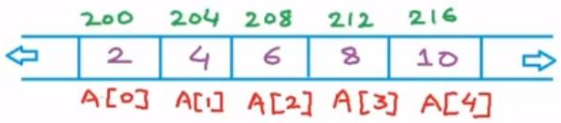
Chamada por referência


```
1   #include <iostream>
2   using namespace std;
3
4   void adiciona(int *p)
5   {
6       *p = *p + 1;
7
8       cout << "Endereço de a (função): " << p << endl;
9   }
10  int main()
11  {
12      setlocale(LC_ALL,"Portuguese");
13      int a = 10;
14      adiciona(&a);
15
16      cout << "Endereço de a (main): " << &a << endl;
17      cout << "Valor de a: " << a  << endl;
18  }
```

Ponteiros - Array


```
1  #include <iostream>
2  using namespace std;
3
4
5  int main()
6  {
7      int A[] = {2,4,6,8,10};
8
9      cout << "A: " << A << endl; //200
10     cout << "&A[0]: " << &A[0] << endl;//200
11     cout << "A[0]: " << A[0] << endl; //2
12     cout << "*A: " << *A << endl; //2
13     cout << endl;
14
15     for(int i=0; i<5; i++)
16     {
17         cout << "i = " << i << endl;
18
19         cout << "&A[i]: " << &A[i] << endl;
20         cout << "A[i]: " << A[i] << endl;
21         cout << "A+i: " << A+i << endl;
22         cout << "*(A+i): " << *(A+i) << endl;
23
24         cout << endl;
25     }
26 }
```

int A[5]



Alocação Dinâmica – vetor


```
1  #include <iostream>
2  #include <ctime>
3  using namespace std;
4
5
6  int main()
7  {
8      srand(time(NULL));
9
10     int tam;
11     cout << "Informe o tamanho do vetor: ";
12     cin >> tam;
13
14     int *V = new int[tam];
15
16     for(int i=0; i<tam; i++)
17     {
18         V[i] = rand()%50;
19         cout << "V[" << i << "] = " << V[i] <<  endl;
20     }
21
22     delete[] V;
23 }
```

Alocação Dinâmica – matriz


```
int l, c;
cout << "Informe o número de linhas: ";
cin >> l;
cout << "Informe o número de colunas: ";
cin >> c;

int **mat = new int*[l];
for(int i=0; i<l; i++)
{
    mat[i] = new int[c];
}

for(int i=0; i<l; i++)
{
    for(int j=0; j<c; j++)
    {
        mat[i][j] = rand()%50;

        cout << mat[i][j] << "\t";
    }
    cout << endl;
}

for(int i=0; i<l; i++)
{
    delete[] mat[i];
}
delete[] mat;
```