Programação

C++

Aula 05

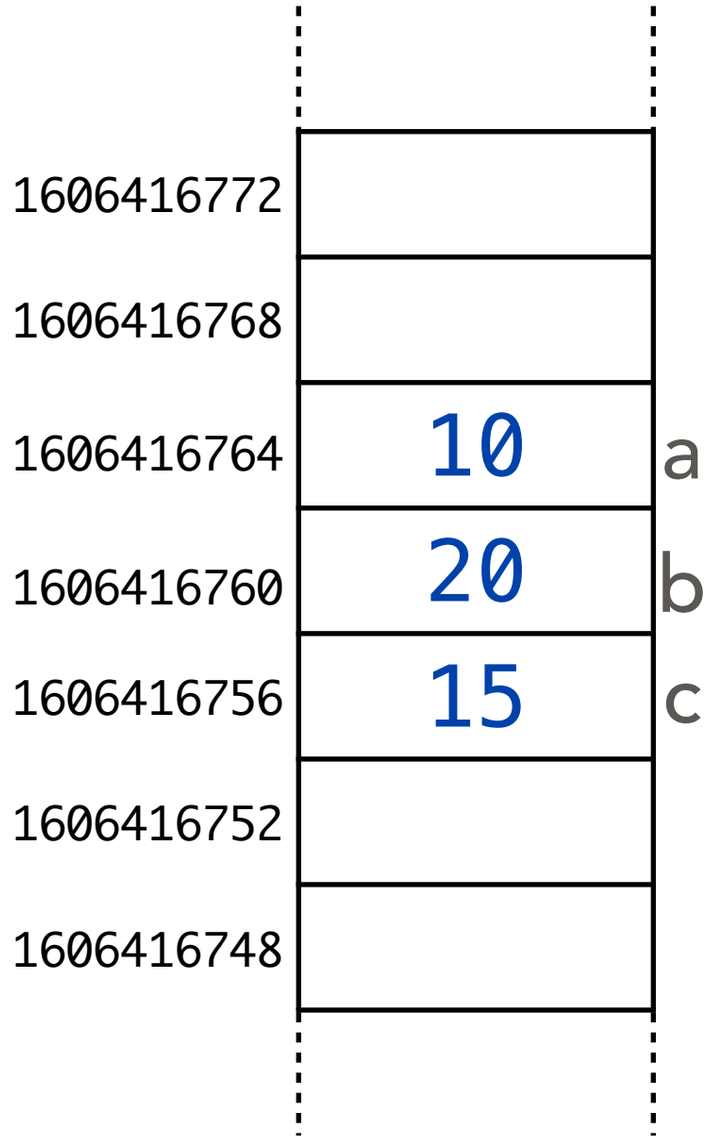
Jorgiano Vidal

Agenda

- Memória de um programa
- Ponteiros
- Organização da memória
- Alocação dinâmica
- Liberação de memória

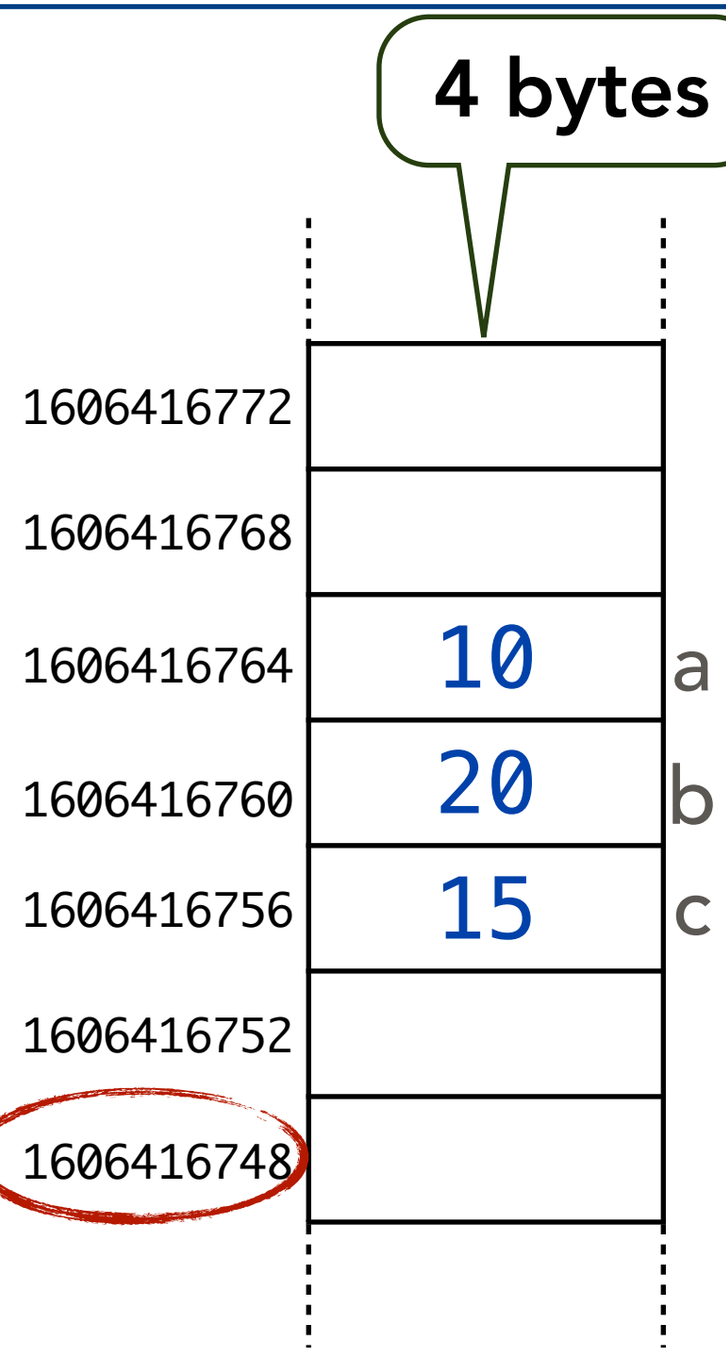
Memória

```c
#include <iostream>

int main() {
   int a,b,c;
   a = 10;
   b = 20;
   c = (a+b)/2;
   /*...*/
   return 0;
}
```



Memória

```c
#include <iostream>

int main() {
   int a,b,c;
   a = 10;
   b = 20;
   c = (a+b)/2;
   /*...*/
   return 0;
}
```



O endereçamento é em **bytes**

Endereço de memória

```cpp
#include <iostream>

int main(){
    int a;
    a = 123;
    std::cout << a << std::endl;
    std::cout << &a << std::endl;
    return 0;
}
```

Endereço de memória

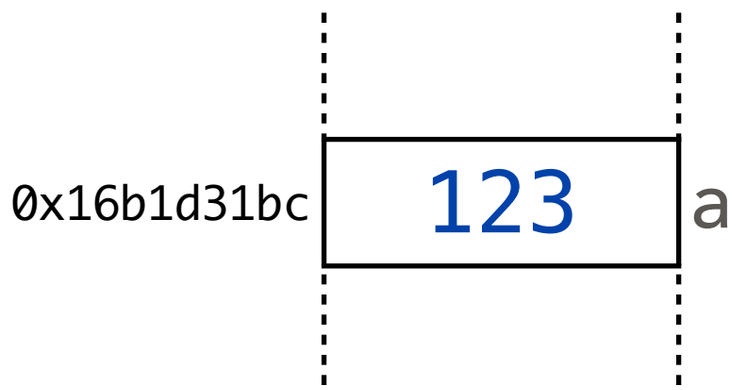
```cpp
#include <iostream>

int main(){
    int a;
    a = 123;
    std::cout << a << std::endl;
    std::cout << &a << std::endl;
    return 0;
}
```

Saída do programa:

```
123
0x16b1d31bc
```

Endereço de memória

```cpp
#include <iostream>

int main(){
    int a;
    a = 123;
    std::cout << a << std::endl;
    std::cout << &a << std::endl;
    return 0;
}
```



Saída do programa:

```
123
0x16b1d31bc
```

Endereço de memória

```cpp
#include <iostream>

int main() {
  int a=10,b=20,c;
  c = (a+b)/2;
  std::cout << "O endereço de a é "
            << reinterpret_cast<std::uintptr_t>(&a)
            << " (" << &a << ")" << std::endl;
  std::cout << "O endereço de b é "
            << reinterpret_cast<std::uintptr_t>(&b)
            << " (" << &b << ")" << std::endl;
  std::cout << "O endereço de c é "
            << reinterpret_cast<std::uintptr_t>(&c)
            << " (" << &c << ")" << std::endl;
  return 0;
}
```

Endereço de memória

```cpp
#include <iostream>

int main() {
  int a=10,b=20,c;
  c = (a+b)/2;
  std::cout << "O endereço de a é "
            << reinterpret_cast<std::uintptr_t>(&a)
            << " (" << &a << ")" << std::endl;
  std::cout << "O endereço de b é "
            << reinterpret_cast<std::uintptr_t>(&b)
            << " (" << &b << ")" << std::endl;
  std::cout << "O endereço de c é "
            << reinterpret_cast<std::uintptr_t>(&c)
            << " (" << &c << ")" << std::endl;
  return 0;
}
```

Saída do programa:

```
$> ./ponteiro03
O endereço de a é 6129922492 (0x16d5f31bc)
O endereço de b é 6129922488 (0x16d5f31b8)
O endereço de c é 6129922484 (0x16d5f31b4)
$>
```

Endereço de memória

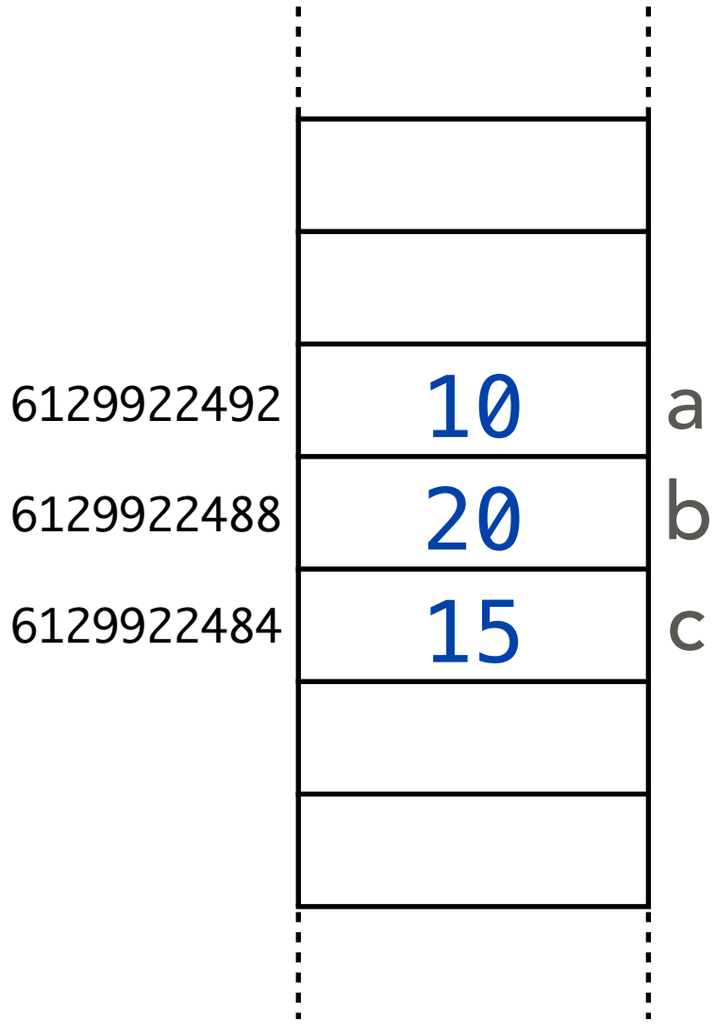
```cpp
#include <iostream>

int main() {
  int a=10,b=20,c;
  c = (a+b)/2;
  std::cout << "O endereço de a é "
            << reinterpret_cast<std::uintptr_t>(&a)
            << " (" << &a << ")" << std::endl;
  std::cout << "O endereço de b é "
            << reinterpret_cast<std::uintptr_t>(&b)
            << " (" << &b << ")" << std::endl;
  std::cout << "O endereço de c é "
            << reinterpret_cast<std::uintptr_t>(&c)
            << " (" << &c << ")" << std::endl;
  return 0;
}
```

Saída do programa:

```
$> ./ponteiro03
O endereço de a é 6129922492 (0x16d5f31bc)
O endereço de b é 6129922488 (0x16d5f31b8)
O endereço de c é 6129922484 (0x16d5f31b4)
$>
```

Endereço de memória

```cpp
#include <iostream>

int main() {
  int a=10,b=20,c;
  c = (a+b)/2;
  std::cout << "O endereço de a é "
            << reinterpret_cast<std::uintptr_t>(&a)
            << " (" << &a << ")" << std::endl;
  std::cout << "O endereço de b é "
            << reinterpret_cast<std::uintptr_t>(&b)
            << " (" << &b << ")" << std::endl;
  std::cout << "O endereço de c é "
            << reinterpret_cast<std::uintptr_t>(&c)
            << " (" << &c << ")" << std::endl;
  return 0;
}
```



Saída do programa:

```
$> ./ponteiro03
O endereço de a é 6129922492 (0x16d5f31bc)
O endereço de b é 6129922488 (0x16d5f31b8)
O endereço de c é 6129922484 (0x16d5f31b4)
$>
```

Ponteiros

Ponteiros

```
int *p;
```

Ponteiros

```
int *p;
```

O * indica que a variável é um ponteiro

Ponteiros

```
int *p;
```

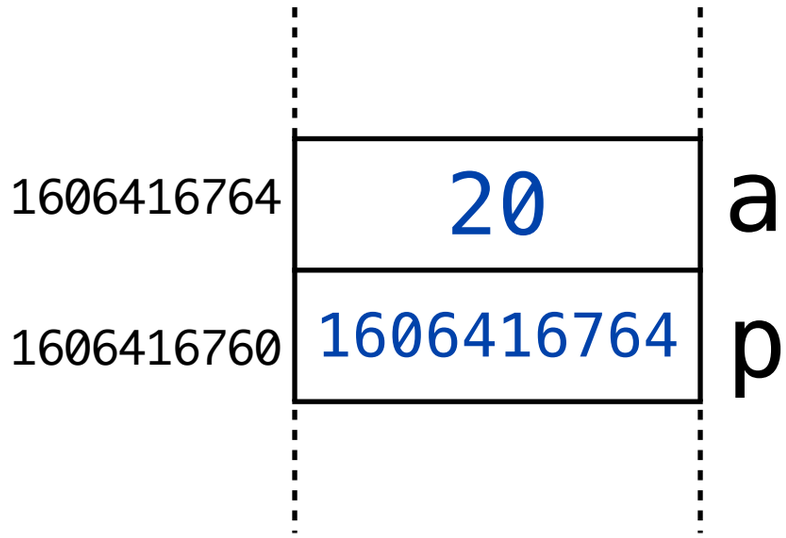
O * indica que a variável é um ponteiro

O tipo indica o tipo do conteúdo armazenado no endereços apontado pela variável ‘ponteiro’

Ponteiros

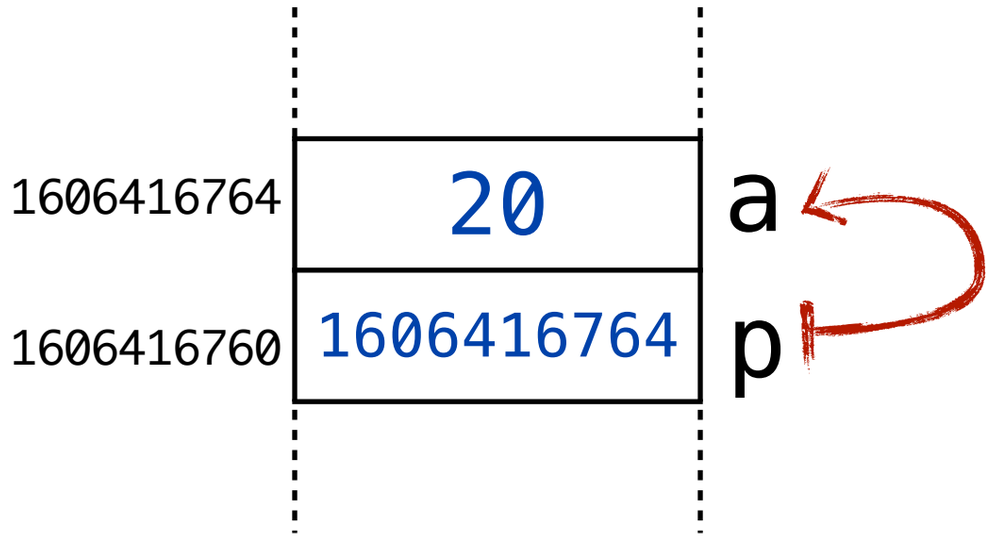
```
#include <iostream>

int main() {
   int a = 20;
   int *p;
   p = &a;
   std::cout << "Valor de *p: "
             << *p << std::endl;
   *p = 50;
   std::cout << "Valor de a: “
             << a << std::endl;
   return 0;
}
```

Ponteiros



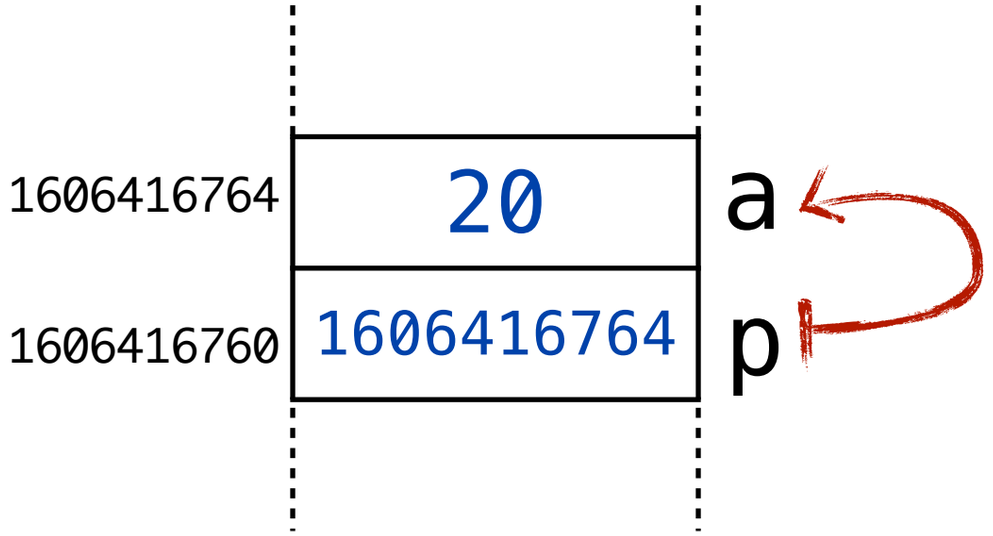

```
#include <iostream>

int main() {
   int a = 20;
   int *p;
   p = &a;
   std::cout << "Valor de *p: "
             << *p << std::endl;
   *p = 50;
   std::cout << "Valor de a: “
             << a << std::endl;
   return 0;
}
```

Ponteiros

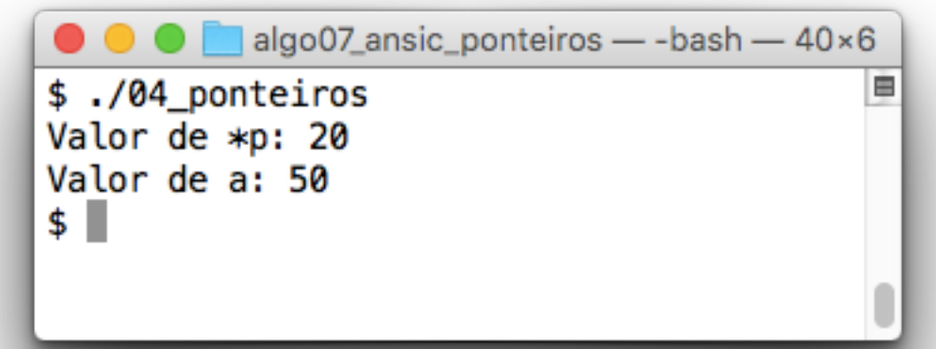



```
#include <iostream>

int main() {
   int a = 20;
   int *p;
   p = &a;
   std::cout << "Valor de *p: "
             << *p << std::endl;
   *p = 50;
   std::cout << "Valor de a: “
             << a << std::endl;
   return 0;
}
```

Ponteiros




```
#include <iostream>

int main() {
    int a = 20;
    int *p;
    p = &a;
    std::cout << "Valor de *p: "
              << *p << std::endl;
    *p = 50;
    std::cout << "Valor de a: “
              << a << std::endl;
    return 0;
}
```



Array

Array

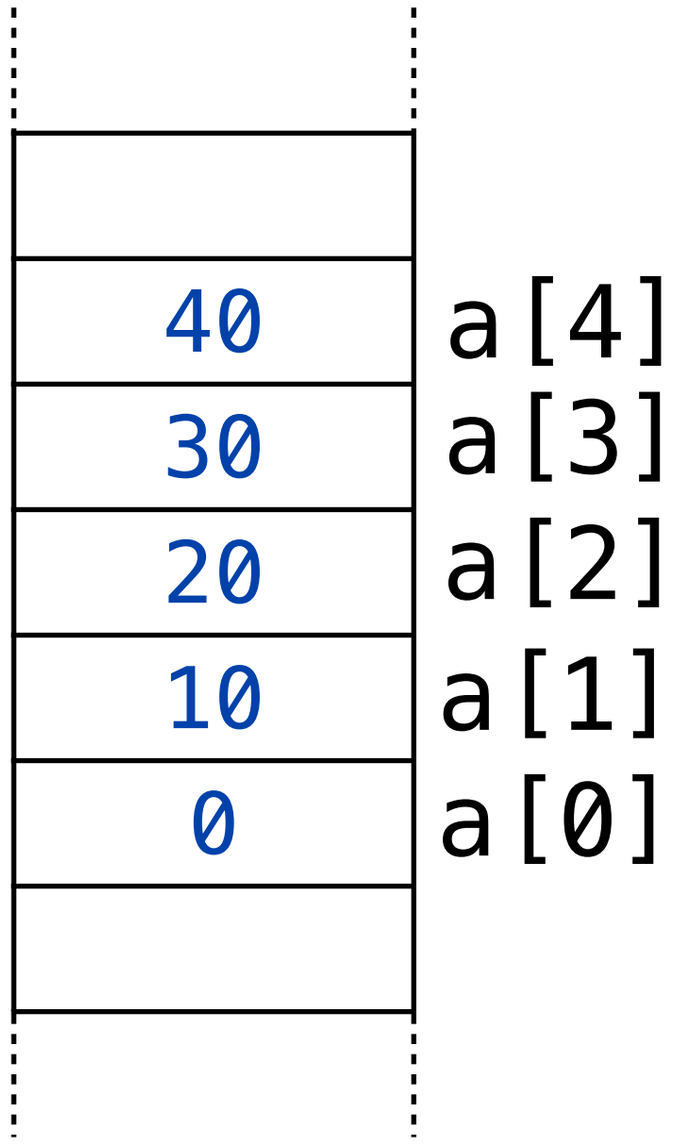
- *Arrays* (e *strings*) são ponteiros para o início de uma **área contínua de memória**
- O valor entre colchetes é o **deslocamento** a partir do endereço inicial
  - a[4] é equivalente a *(a+4)
- Considere um array

```
int a[5]
```

Array

- *Arrays* (e *strings*) são ponteiros para o início de uma **área contínua de memória**
- O valor entre colchetes é o **deslocamento** a partir do endereço inicial
  - a[4] é equivalente a *(a+4)
- Considere um array

```
int a[5]
```



Array

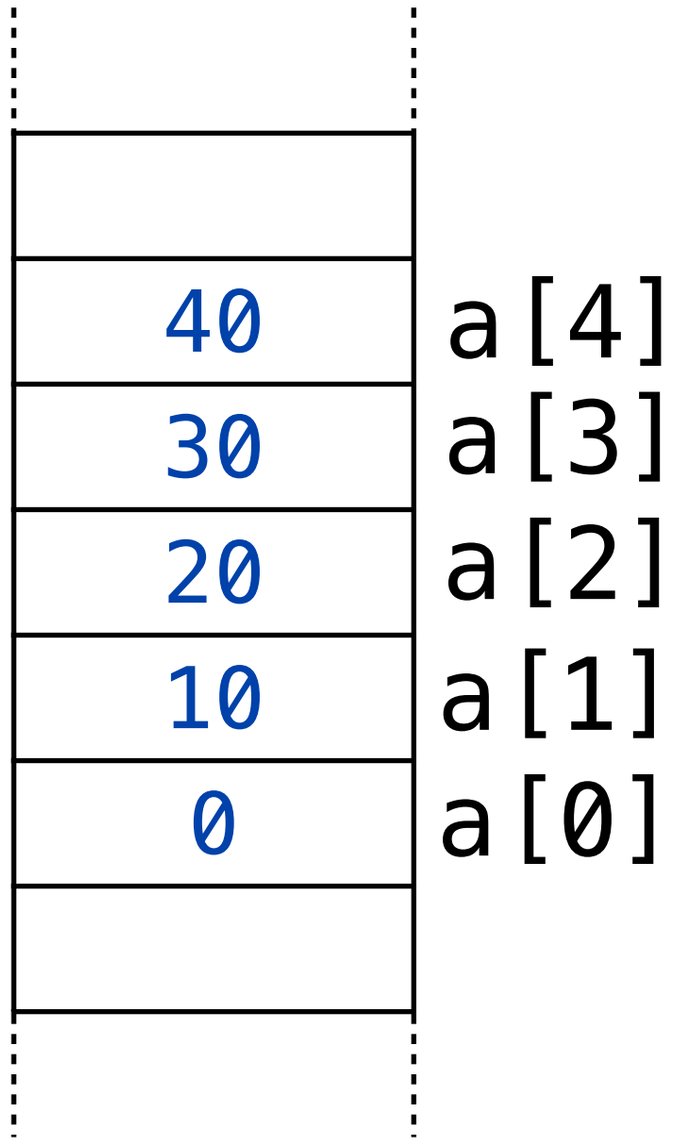
- *Arrays* (e *strings*) são ponteiros para o início de uma **área contínua de memória**
- O valor entre colchetes é o **deslocamento** a partir do endereço inicial
  - a[4] é equivalente a *(a+4)
- Considere um array

  int a[5]
- 5 espaços de memórias (int) consecutivos
- São equivalentes:

  a e &a

  *(a) e a[0]

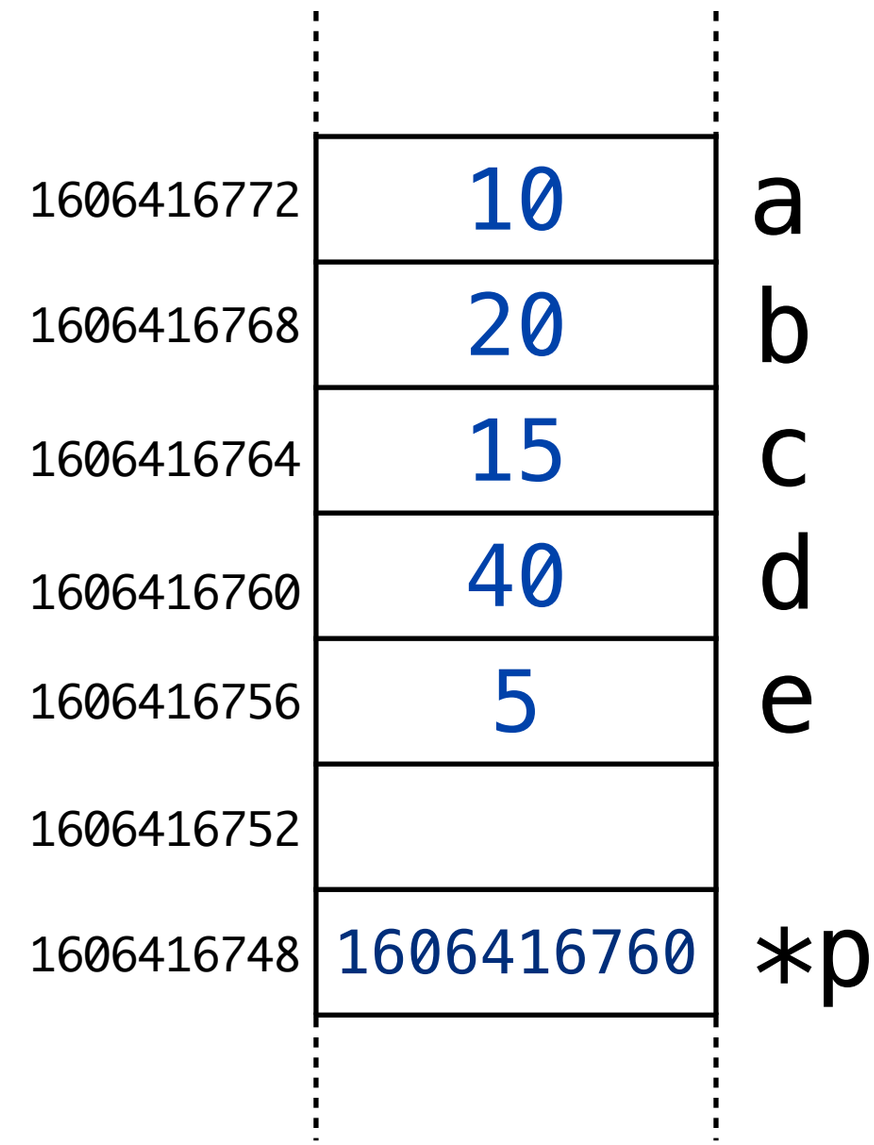
  *(a+2) e a[2]



Aritmética de ponteiros

- Como fica?

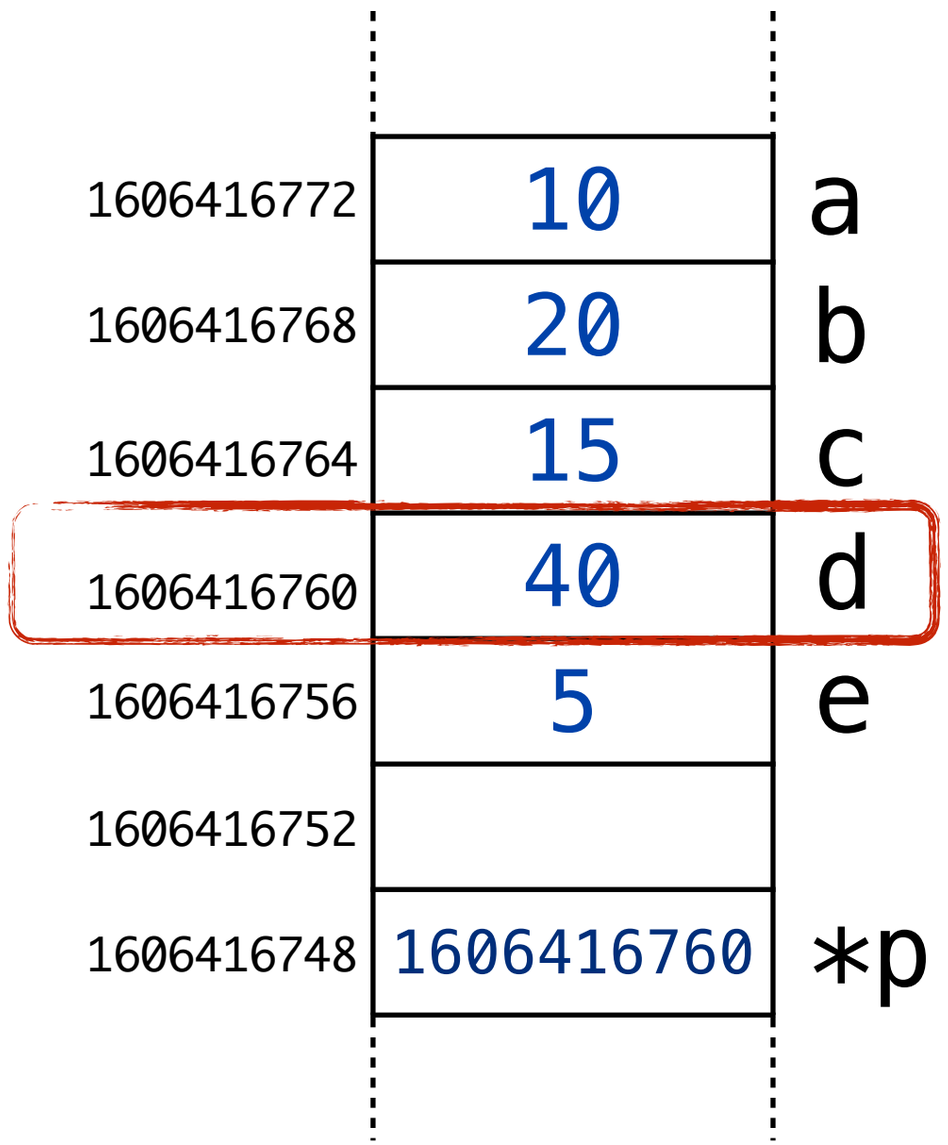
```
*p = *(p+1)-*(p+3)
```



Aritmética de ponteiros

- Como fica?

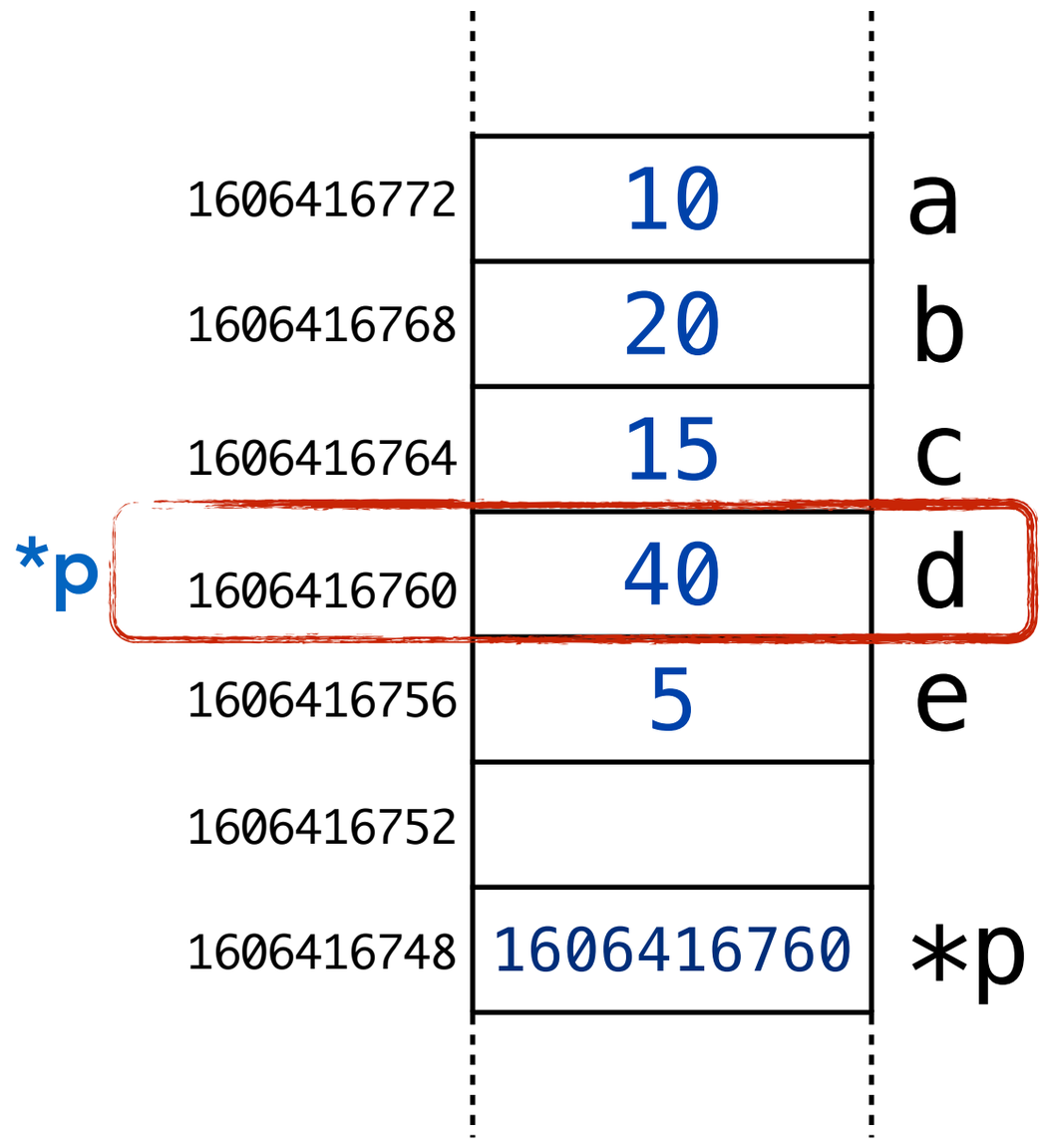
```
*p = *(p+1)-*(p+3)
```



Aritmética de ponteiros

- Como fica?

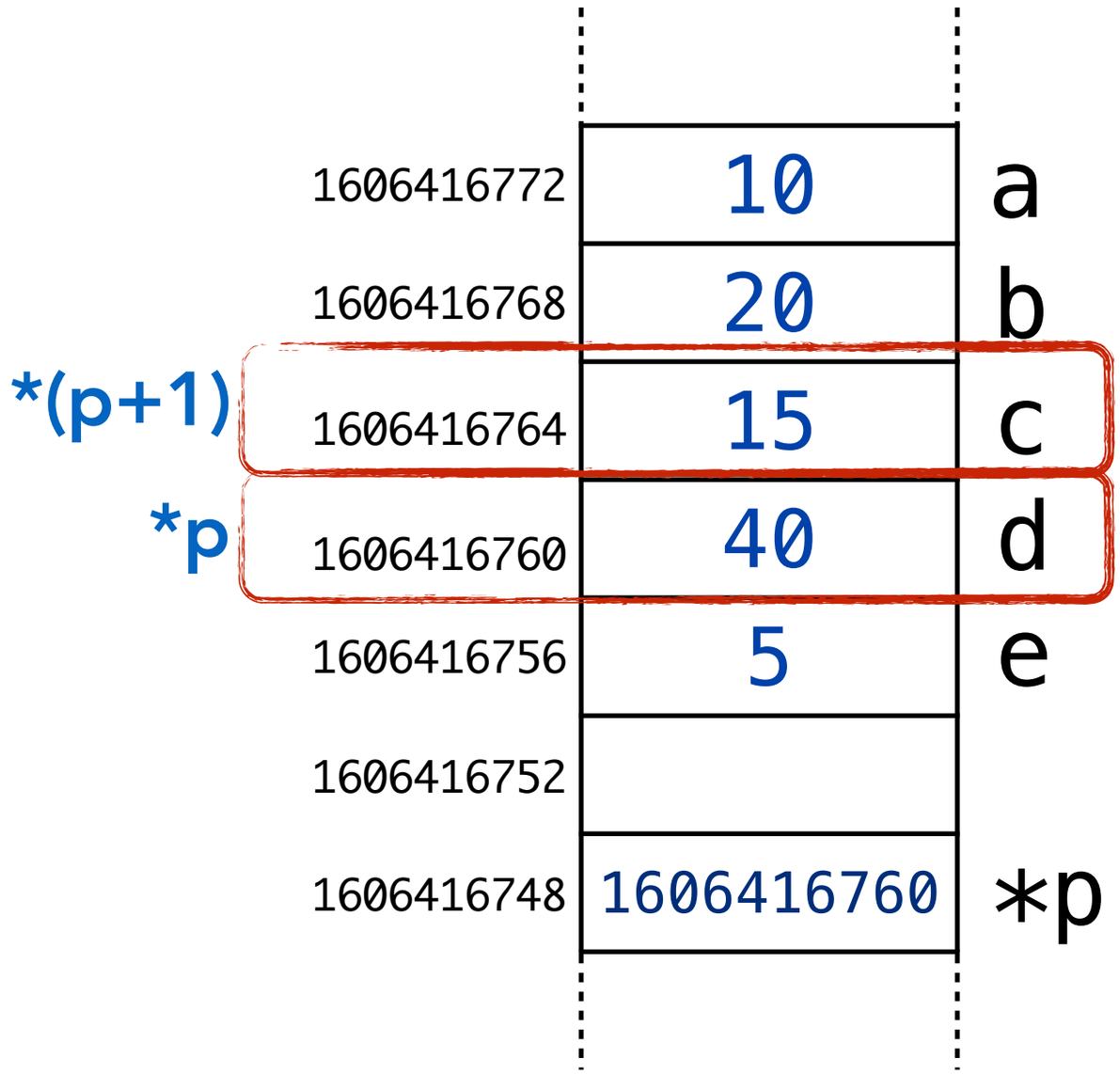
```
*p = *(p+1)-*(p+3)
```



Aritmética de ponteiros

- Como fica?

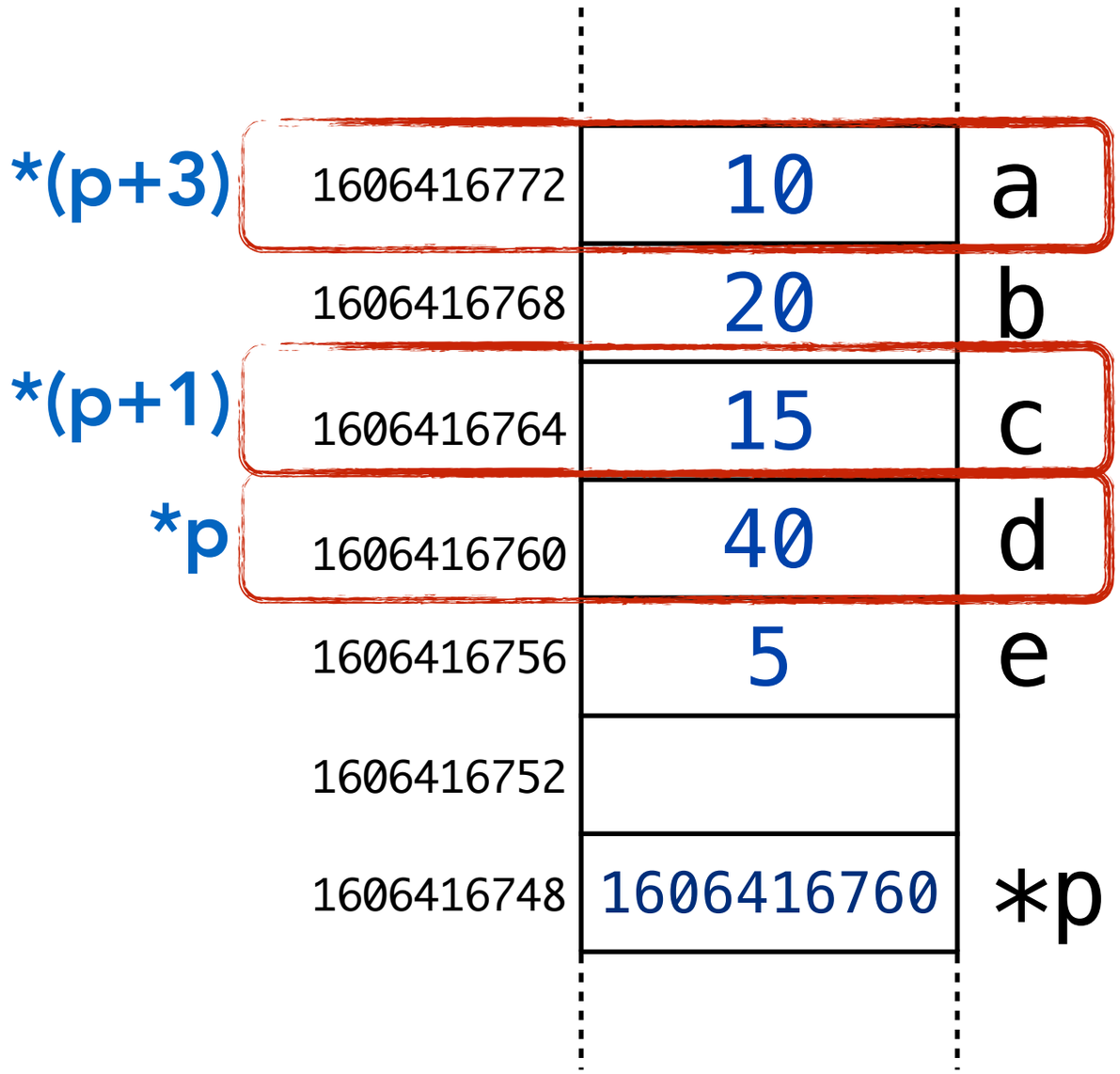
```
*p = *(p+1)-*(p+3)
```



Aritmética de ponteiros

- Como fica?

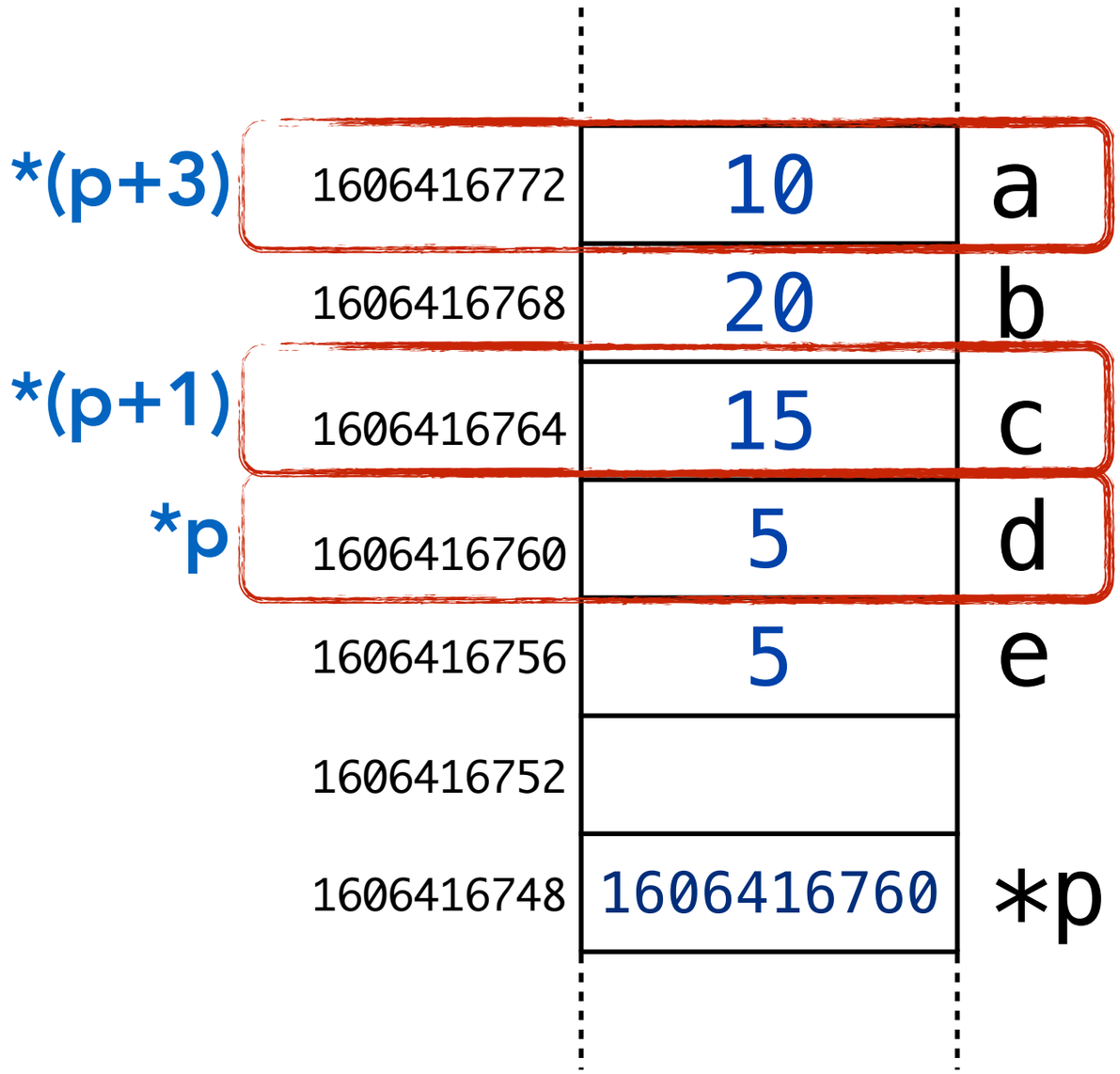
```
*p = *(p+1)-*(p+3)
```



Aritmética de ponteiros

- Como fica?

```
*p = *(p+1)-*(p+3)
```



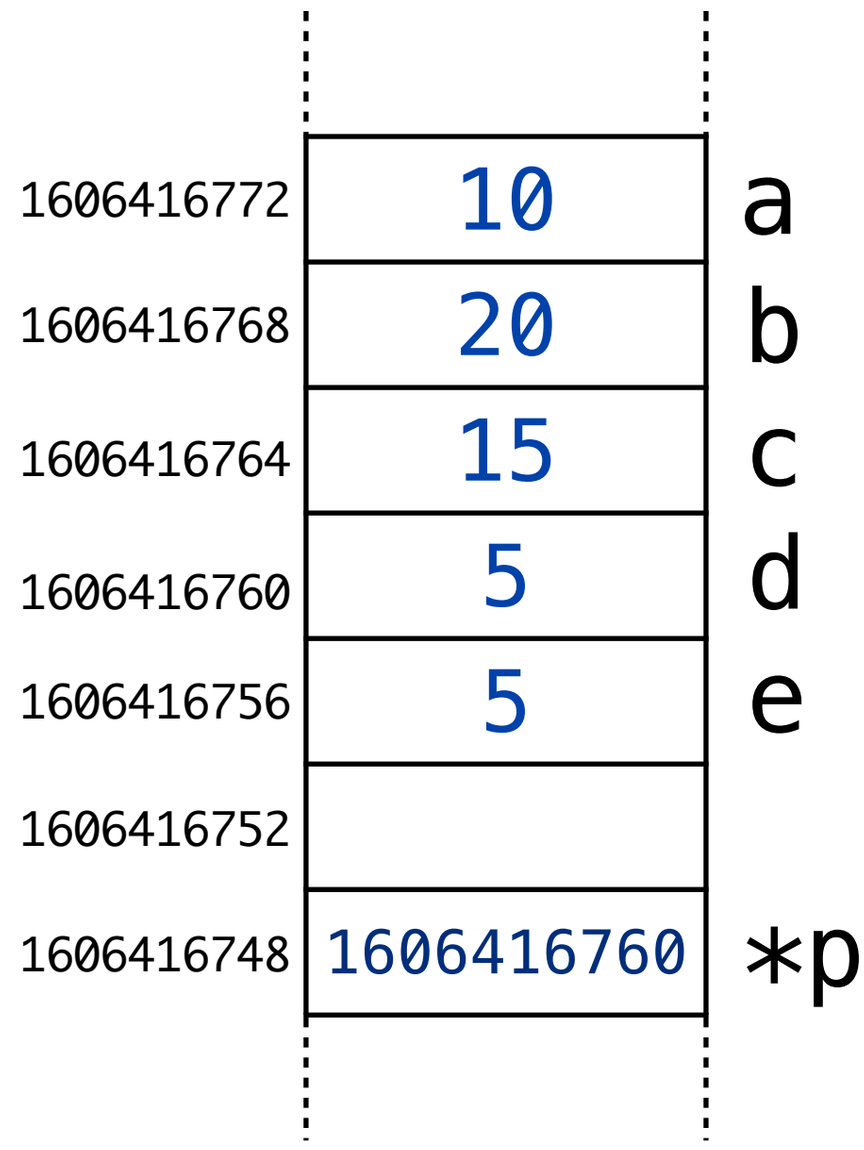
O valor de d passa a ser 5

Aritmética de ponteiros

- Como fica?

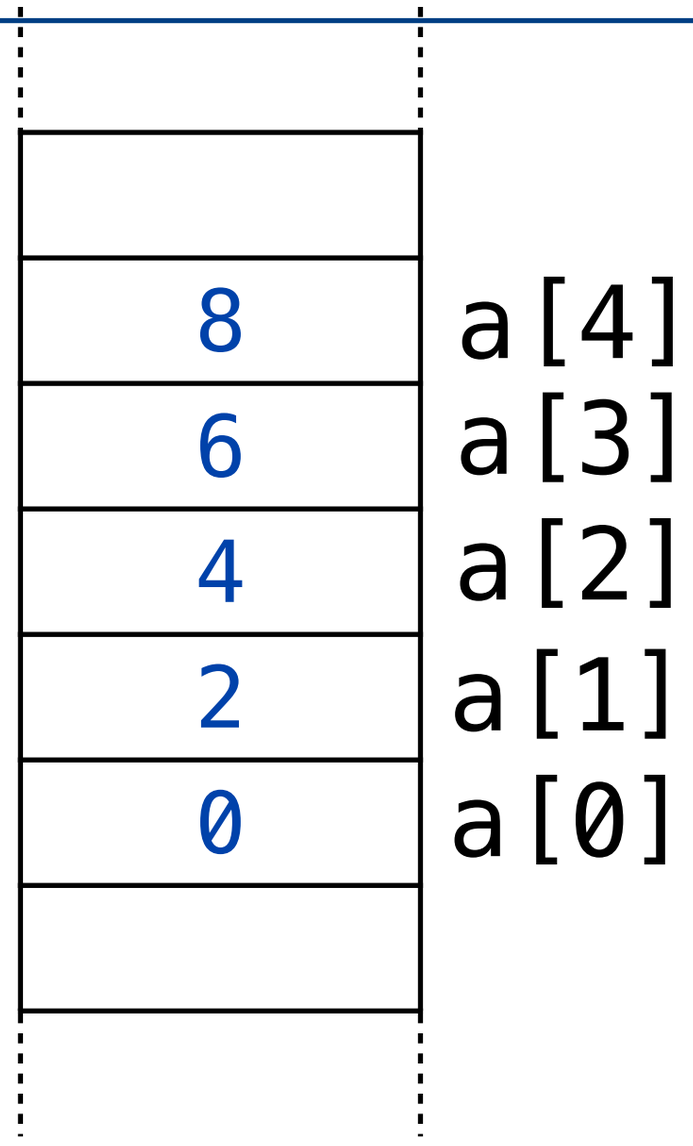
```
*p = *(p+1)-*(p+3)
```

O valor de d passa a ser 5



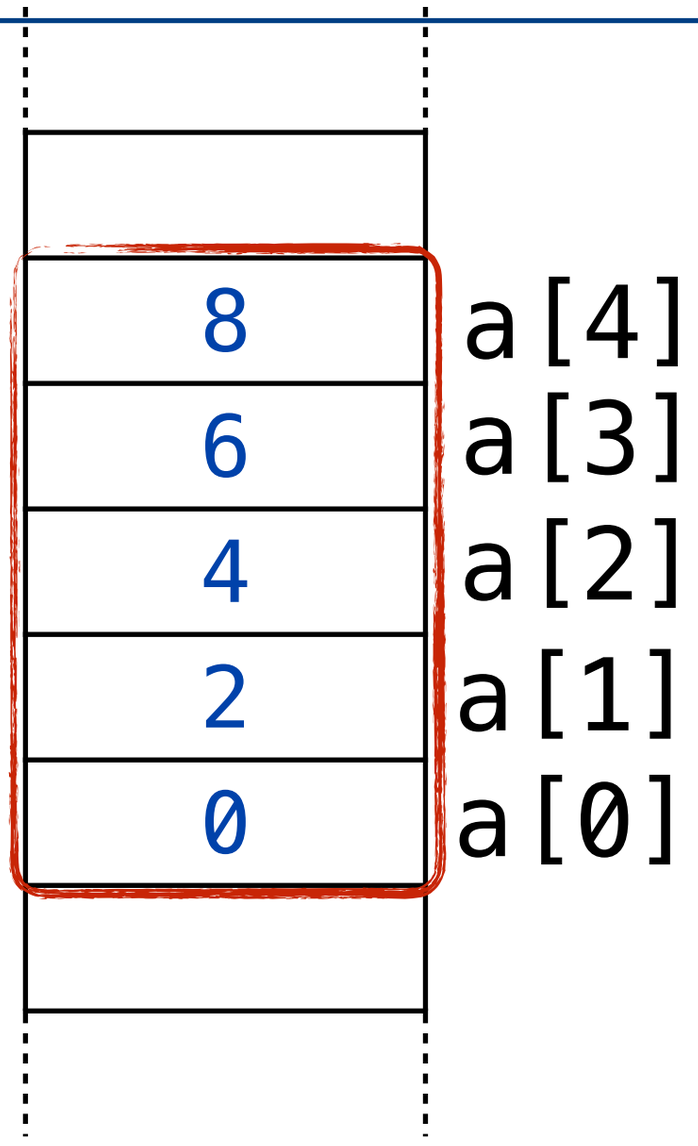
Array

```cpp
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



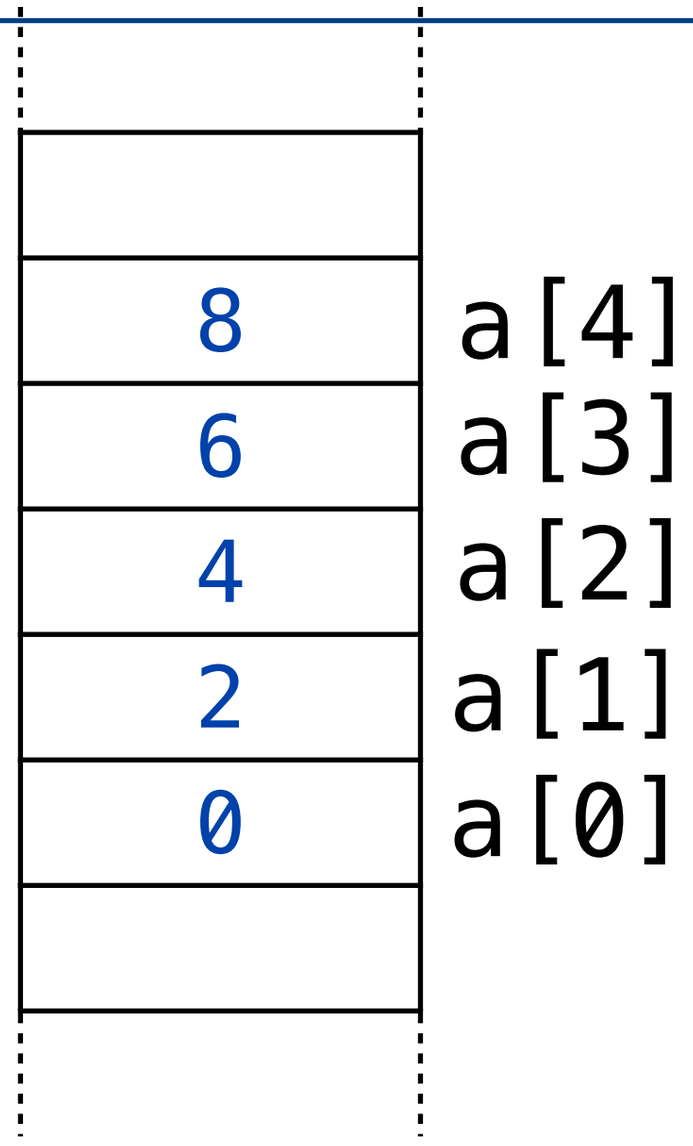
Array

```cpp
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



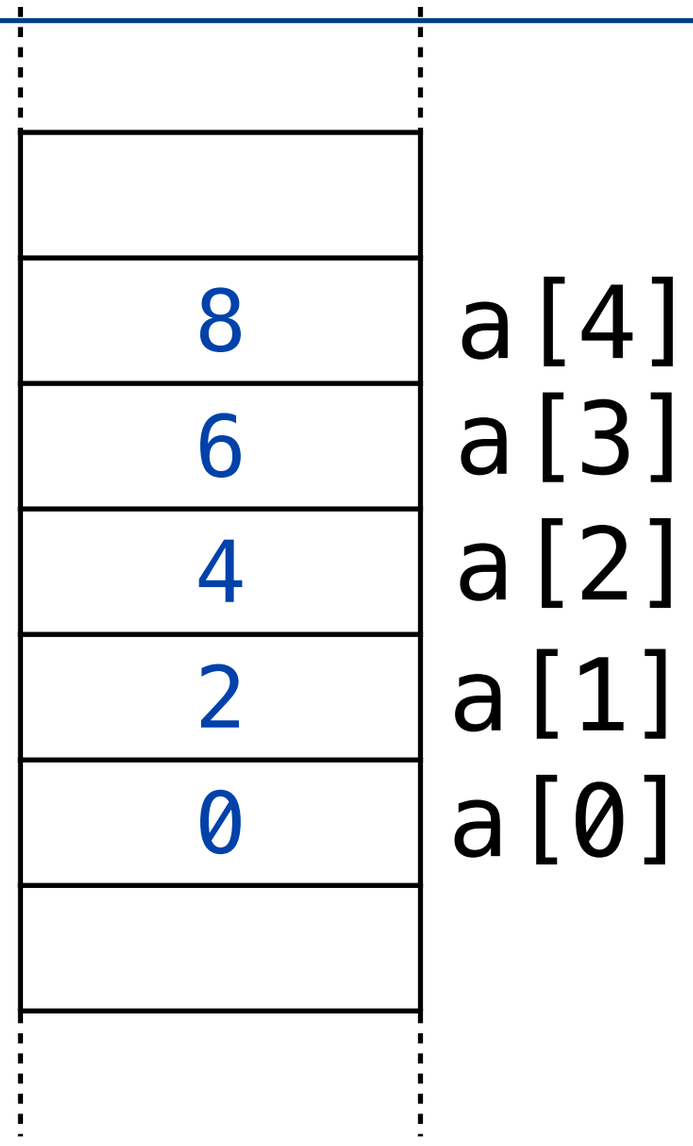
Array

```cpp
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



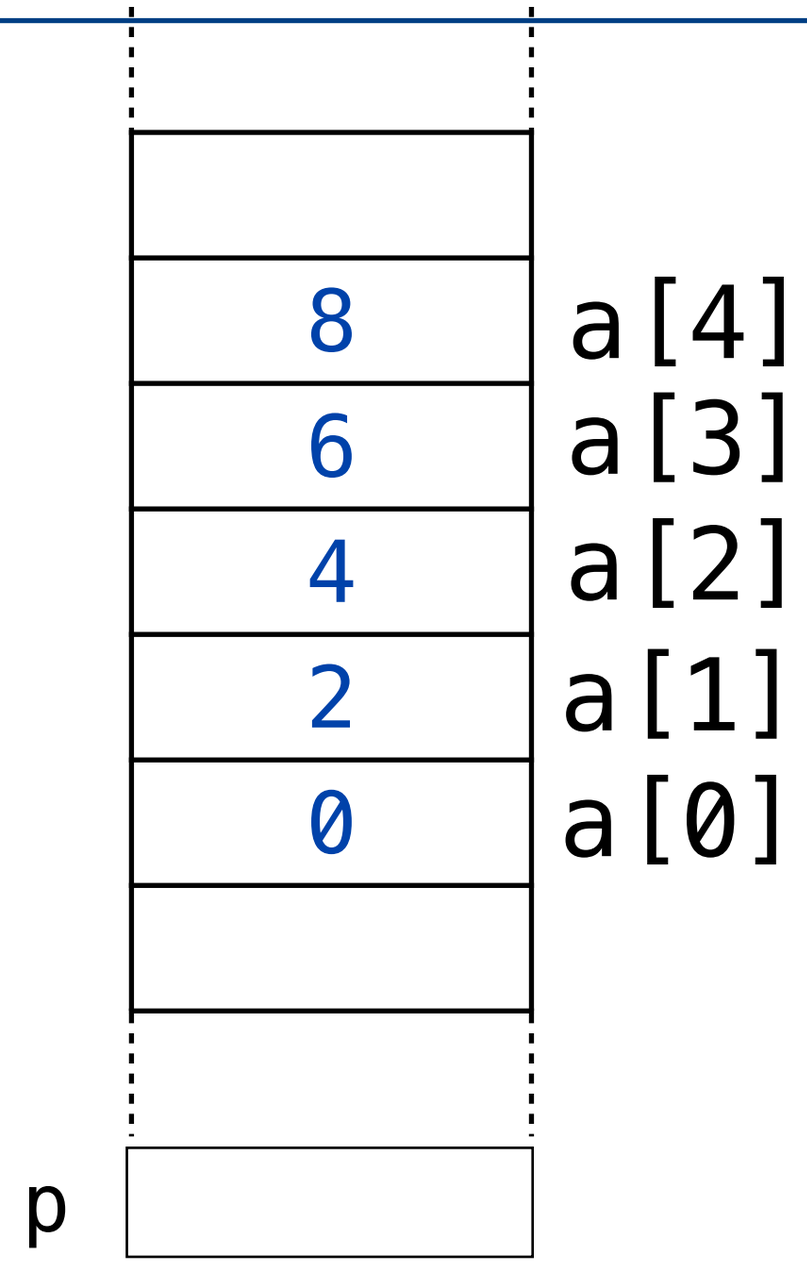
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



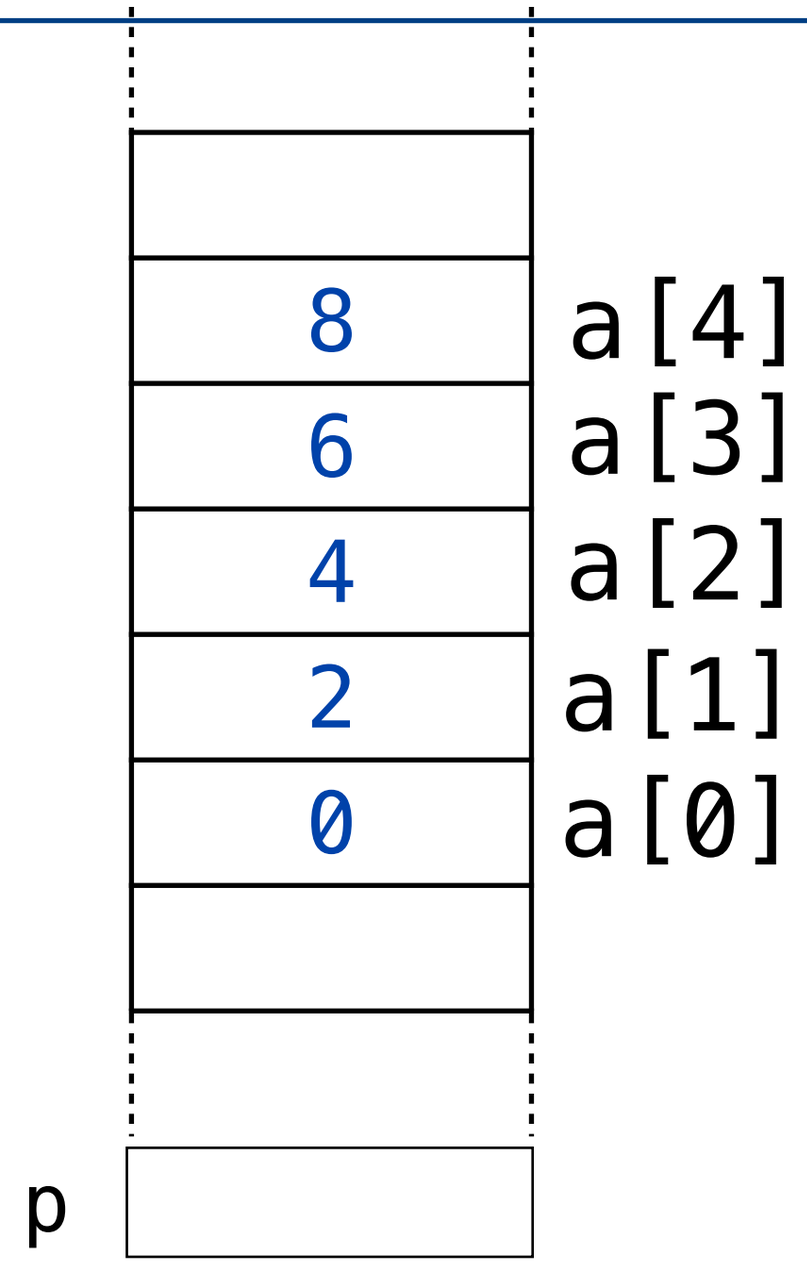
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



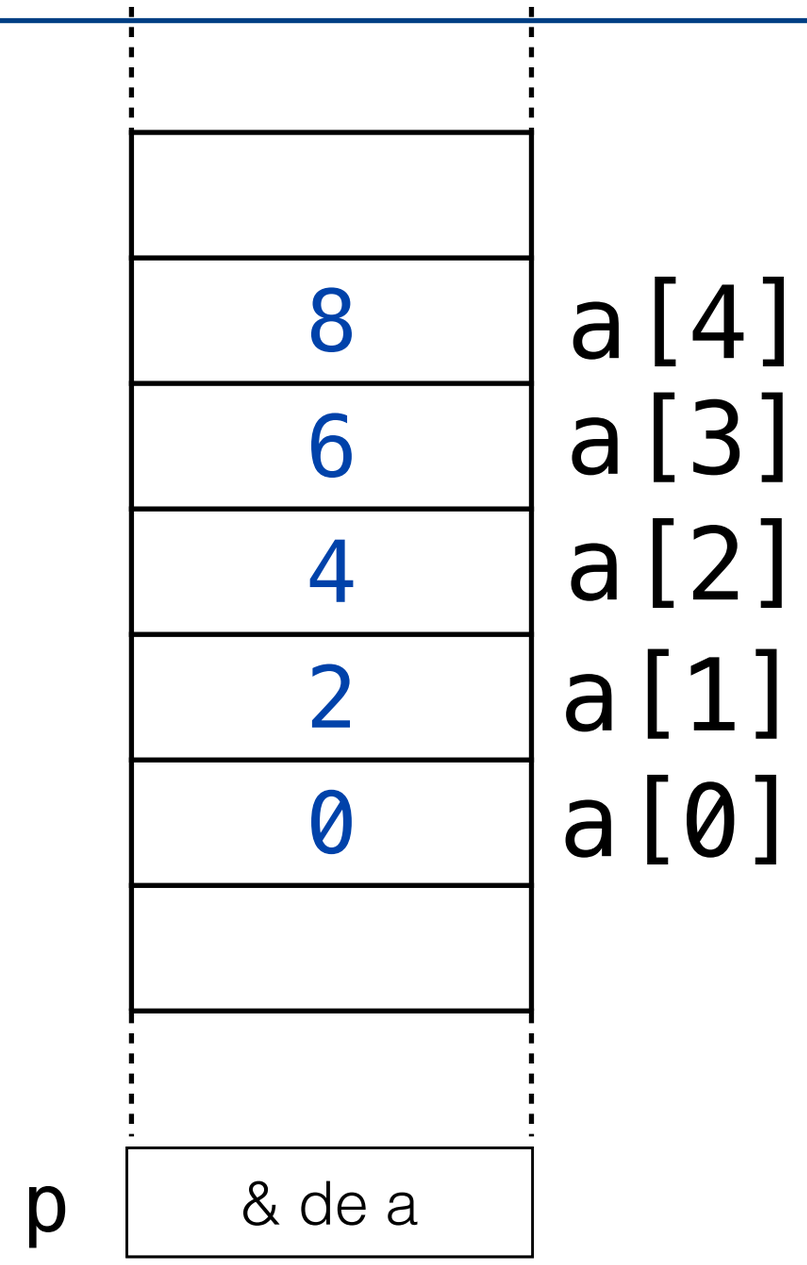
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



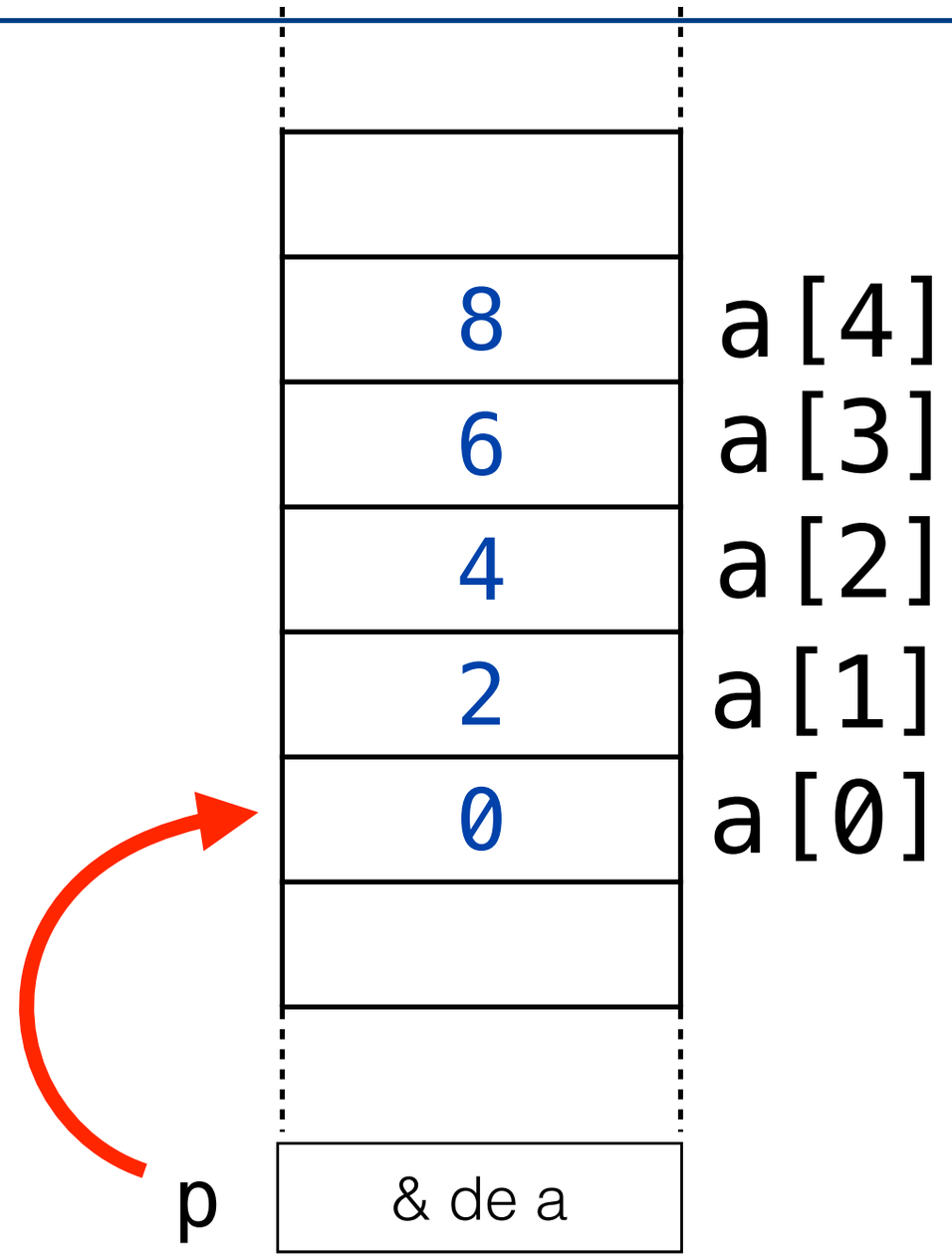
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



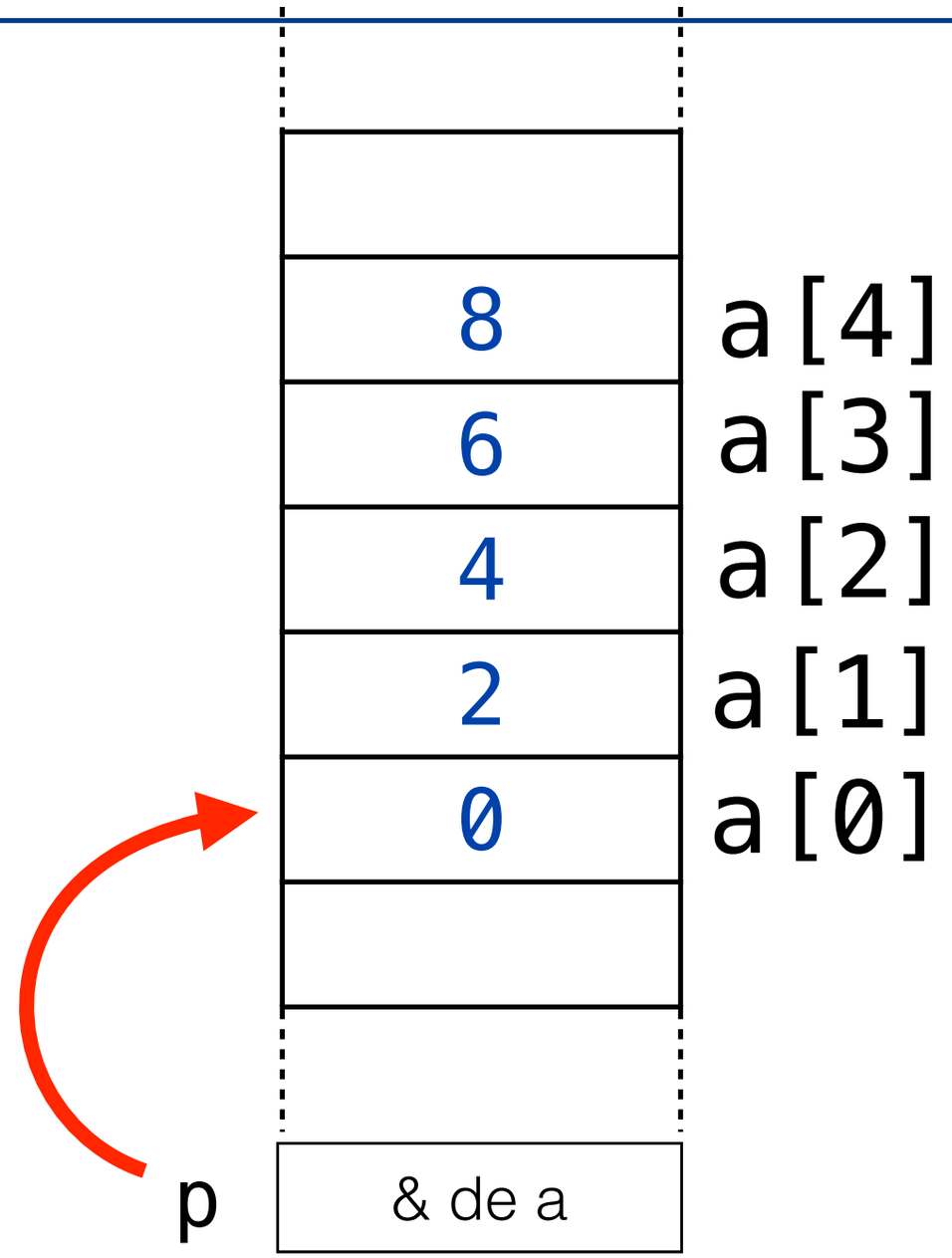
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



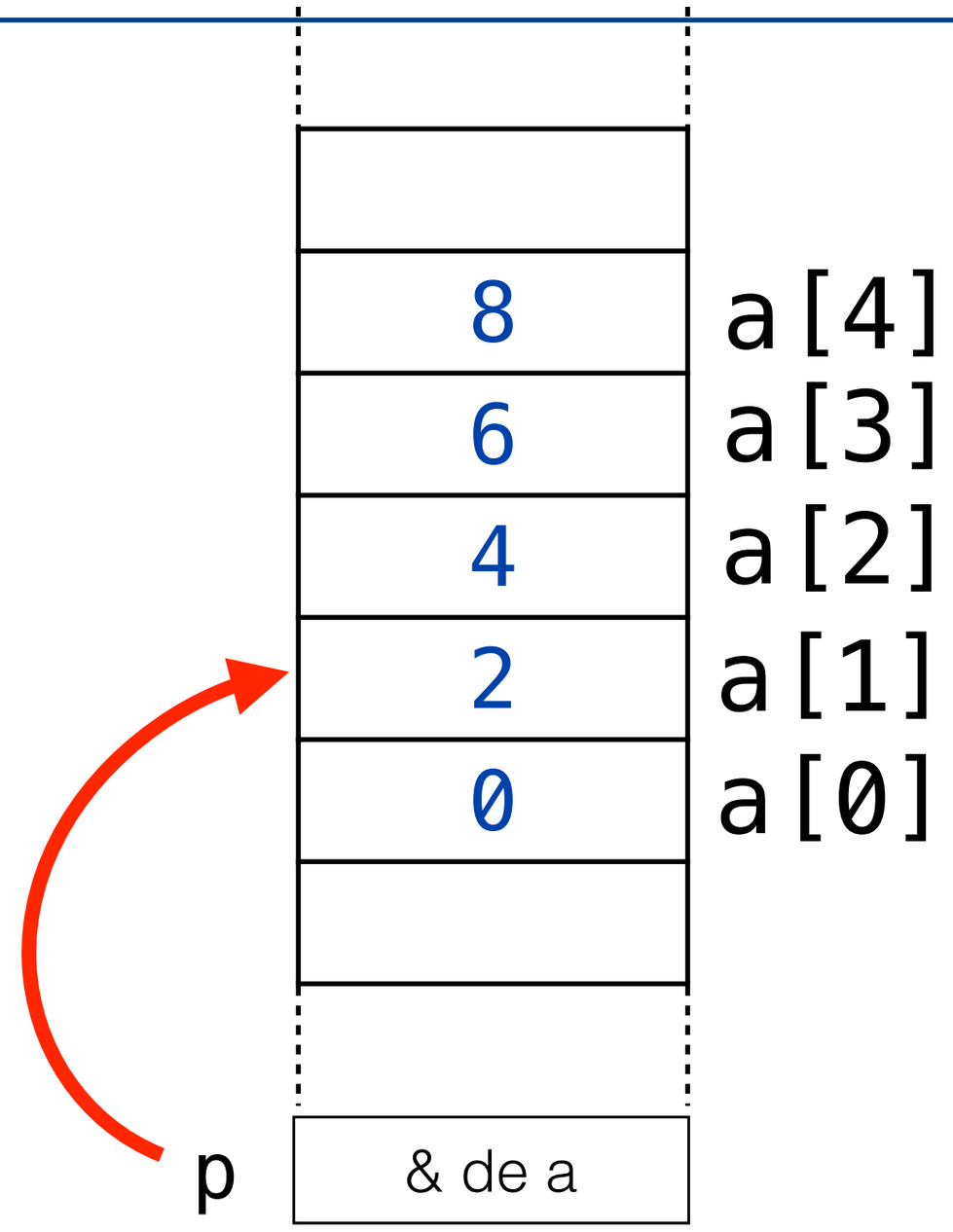
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



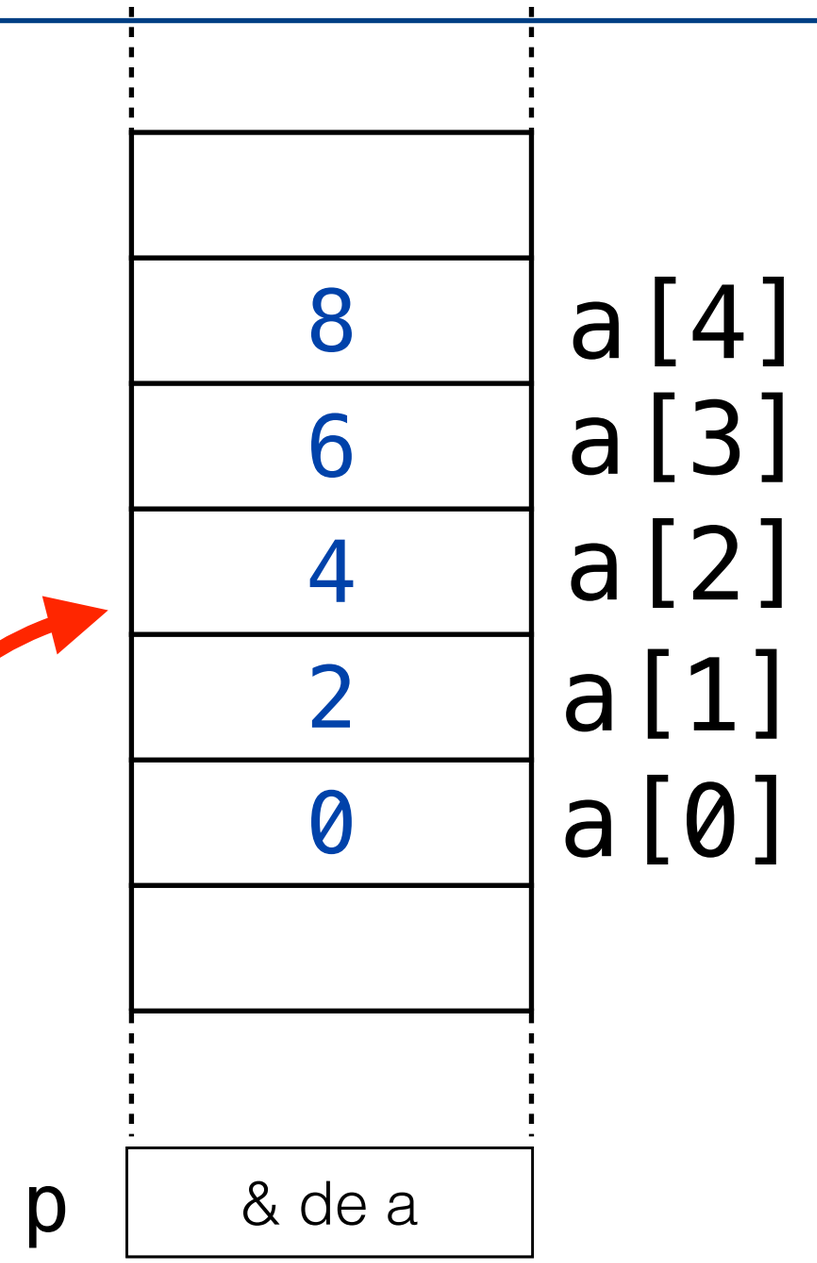
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



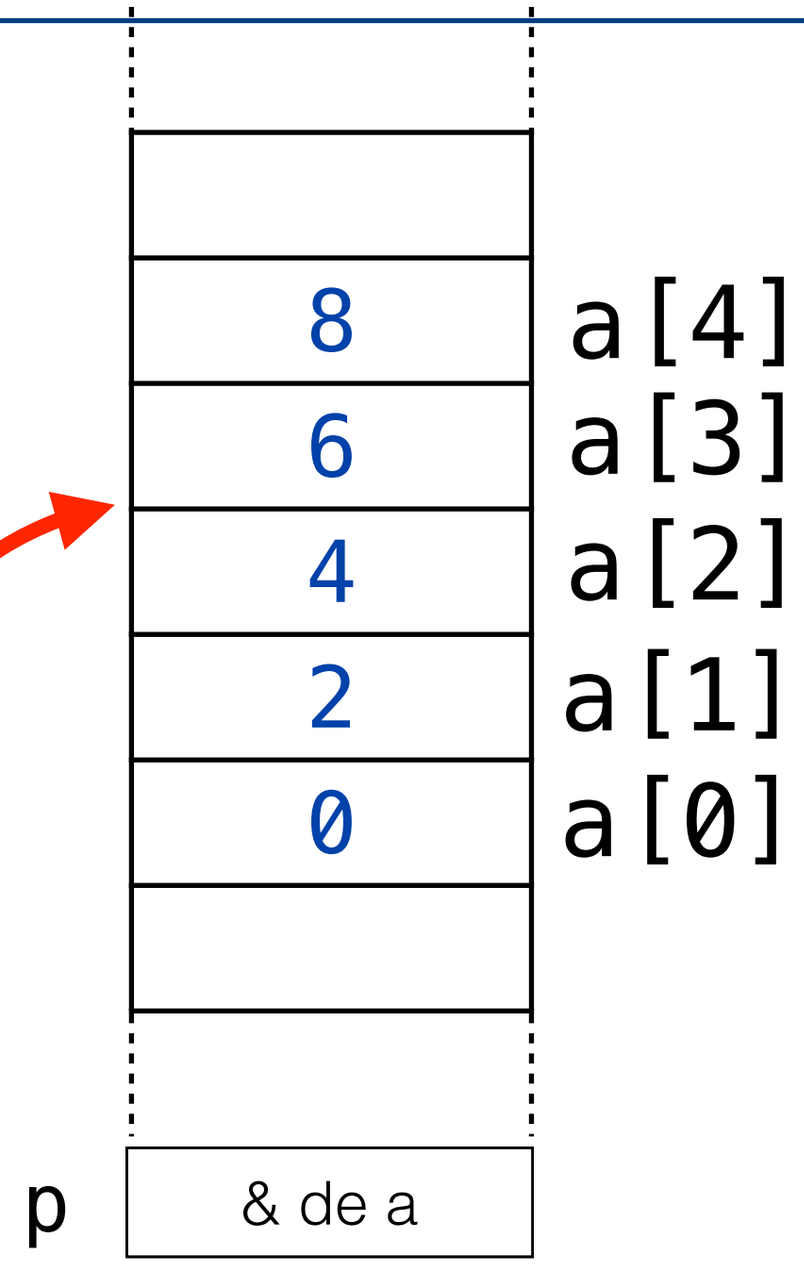
Array

```cpp
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



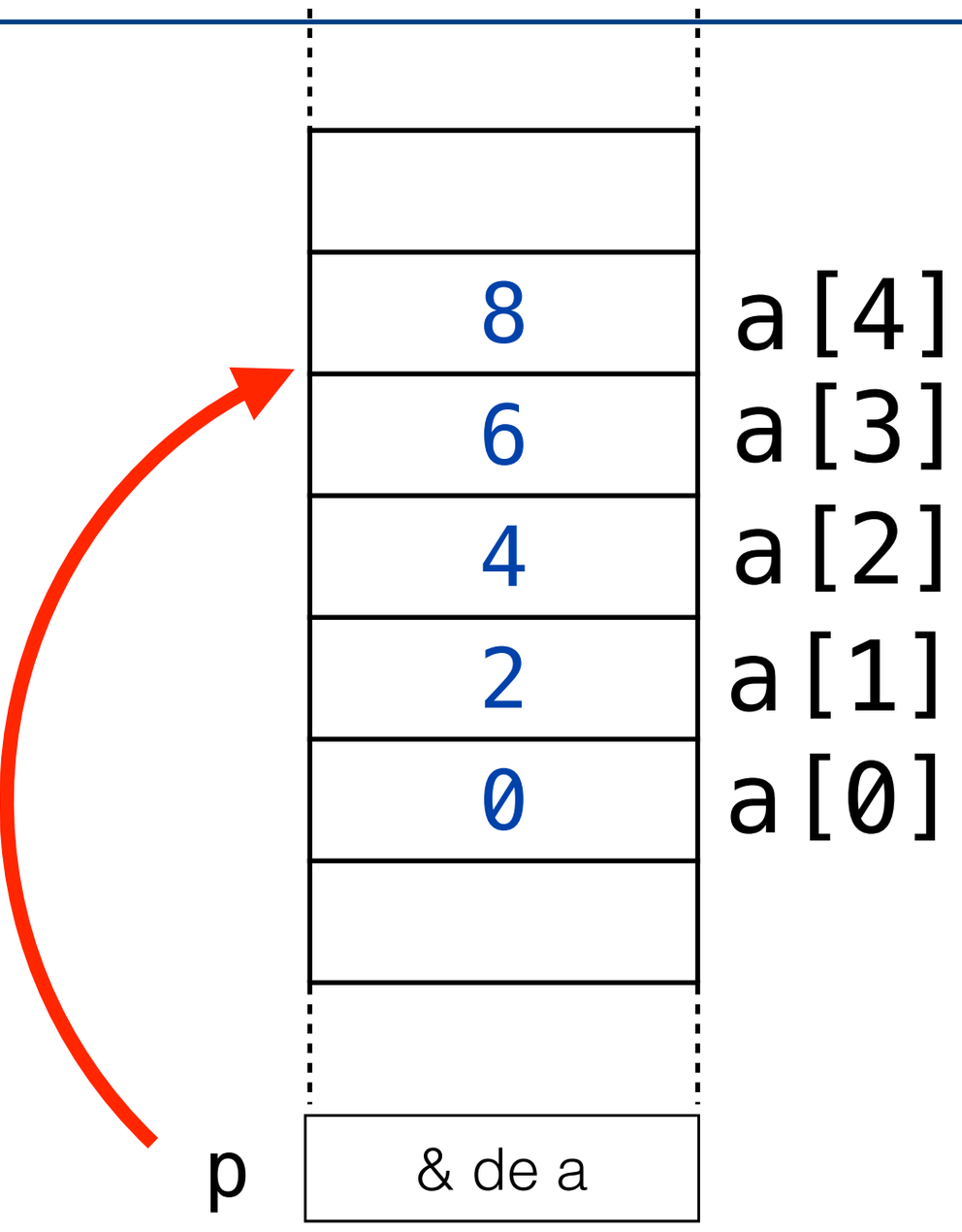
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



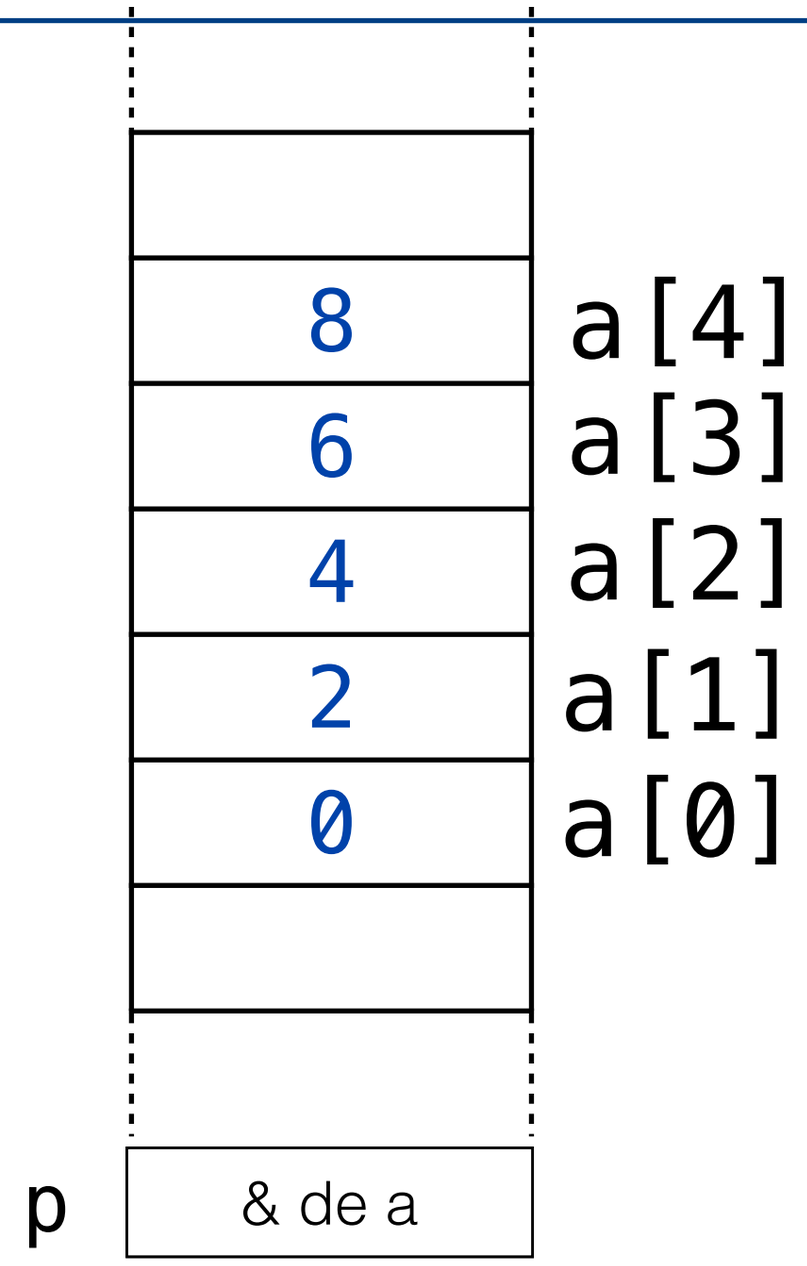
Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```



Array

```
#include <iostream>

int main(){
   int i,a[5];
   int *p;
   for (i=0 ; i<5 ; i++){
      a[i]=i*2;
   }
   p = a;
   for (i=0 ; i<5 ; i++){
      std::cout << *p << std::endl;
      p++;
   }
   return 0;
}
```

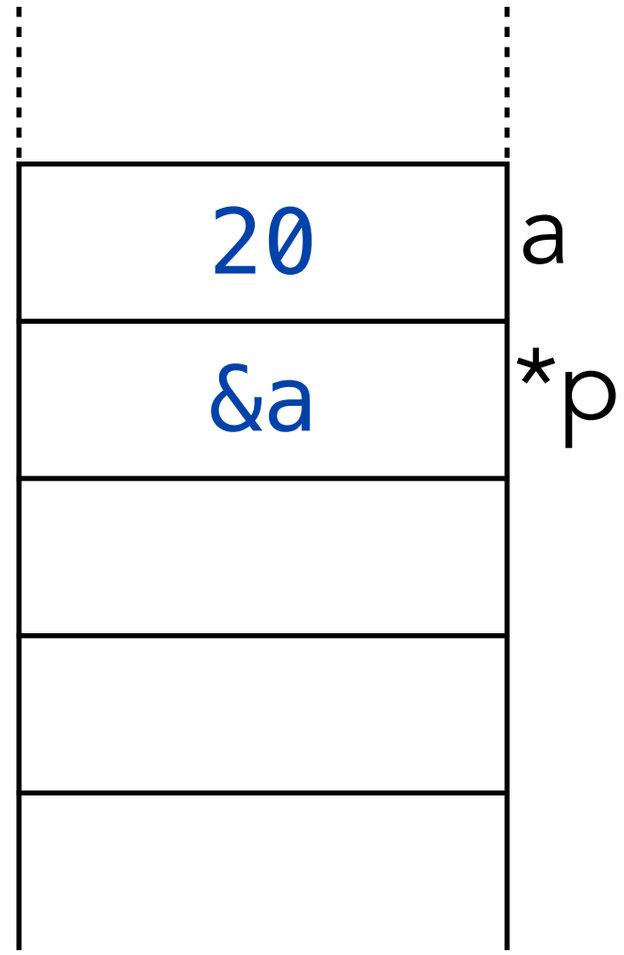


Observações

- O uso de ponteiros favorece ao aparecimento de **erros**
  - Um acesso a um conteúdo de uma área de memória não autorizada causa erro
  - ***Segmentation fault***
- Muito cuidado no uso de ponteiros!

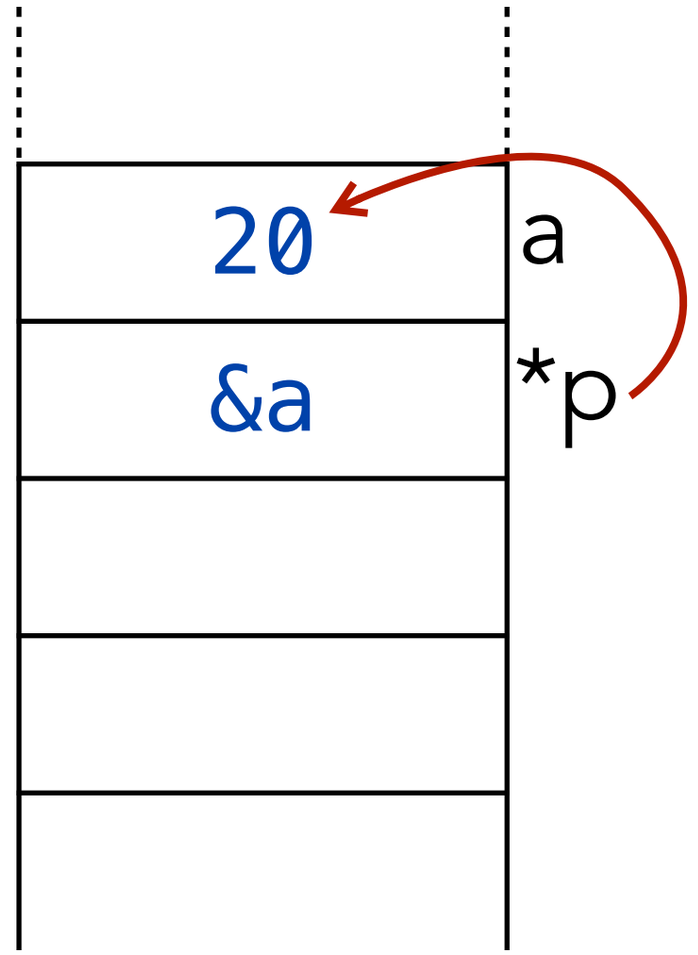
Memória

- Declaração de variável
  - Memória alocada
  - **Endereço fixo**
- Ponteiros armazenam endereços
  - E se quisermos apontar para endereços que **não sejam variáveis** do programa?



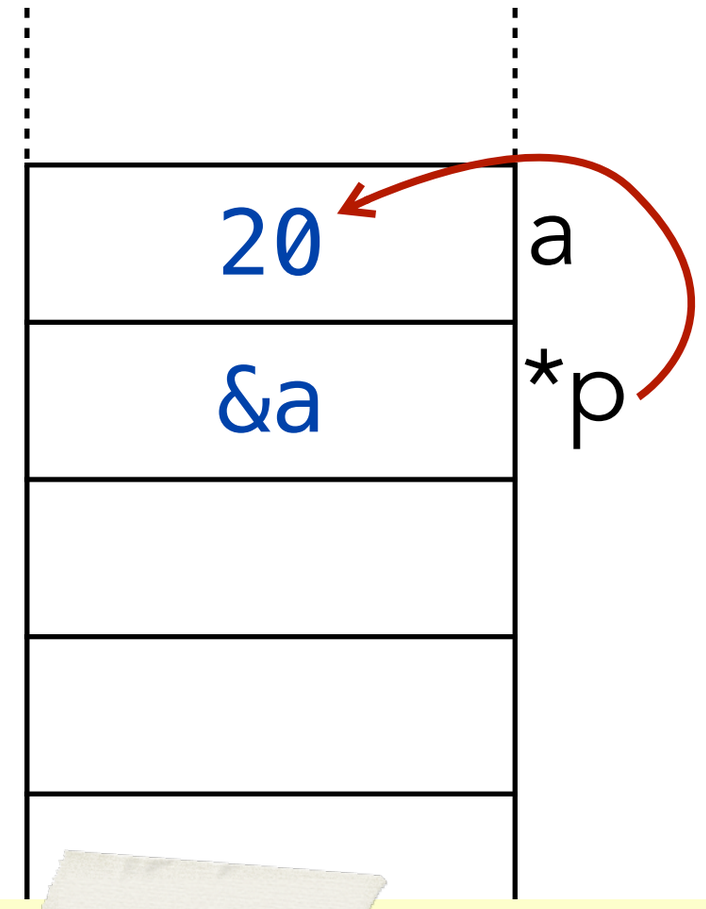
Memória

- Declaração de variável
  - Memória alocada
  - **Endereço fixo**
- Ponteiros armazenam endereços
  - E se quisermos apontar para endereços que **não sejam variáveis** do programa?



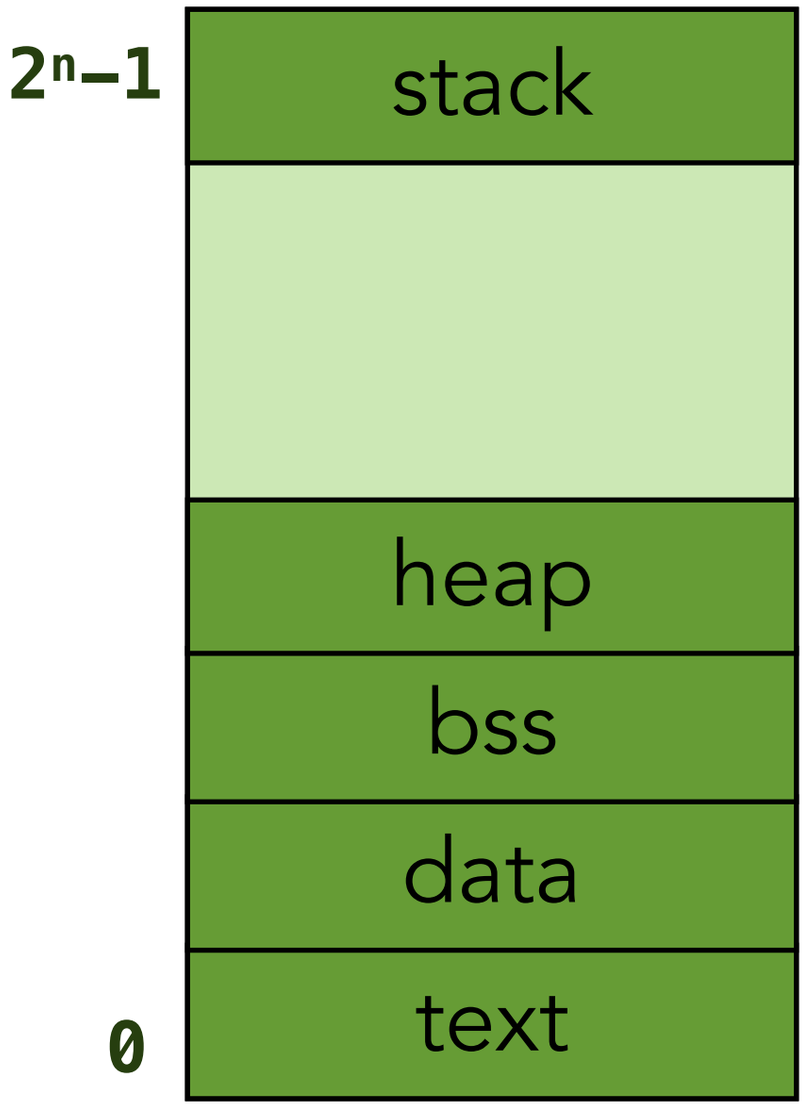
Memória

- Declaração de variável
  - Memória alocada
  - **Endereço fixo**
- Ponteiros armazenam endereços
  - E se quisermos apontar para endereços que **não sejam variáveis** do programa?

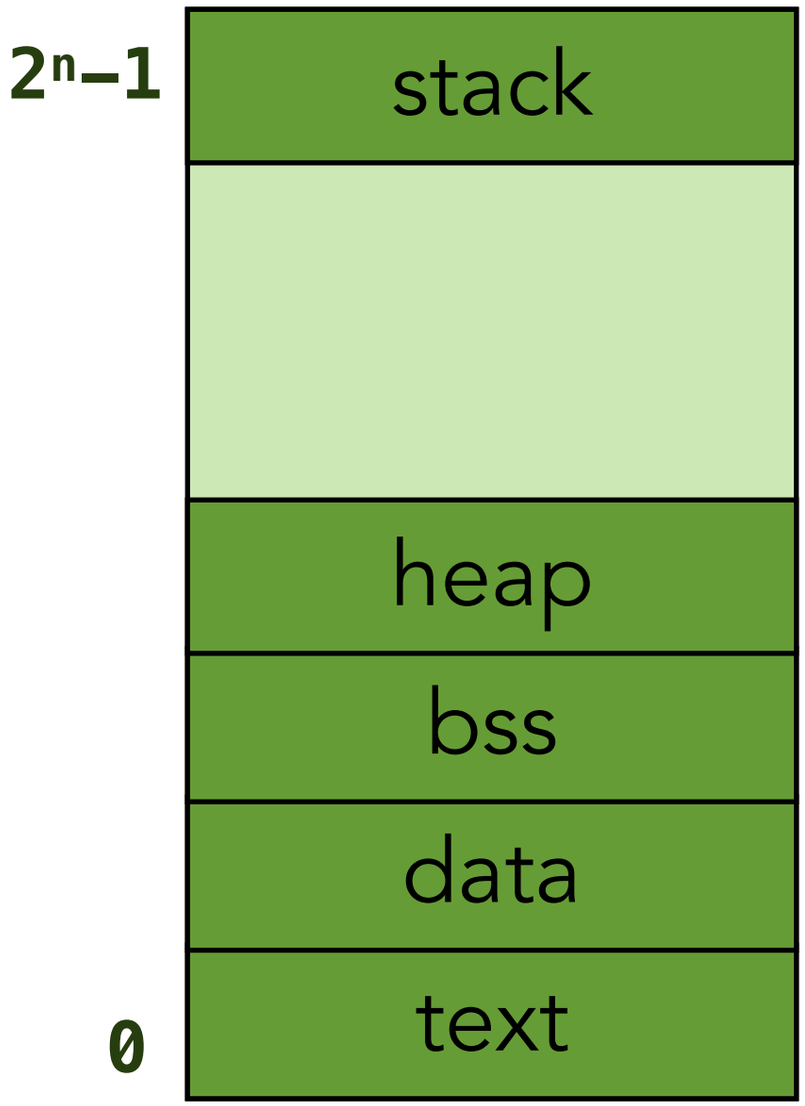


p pode apontar para **QUALQUER** endereço

Memória

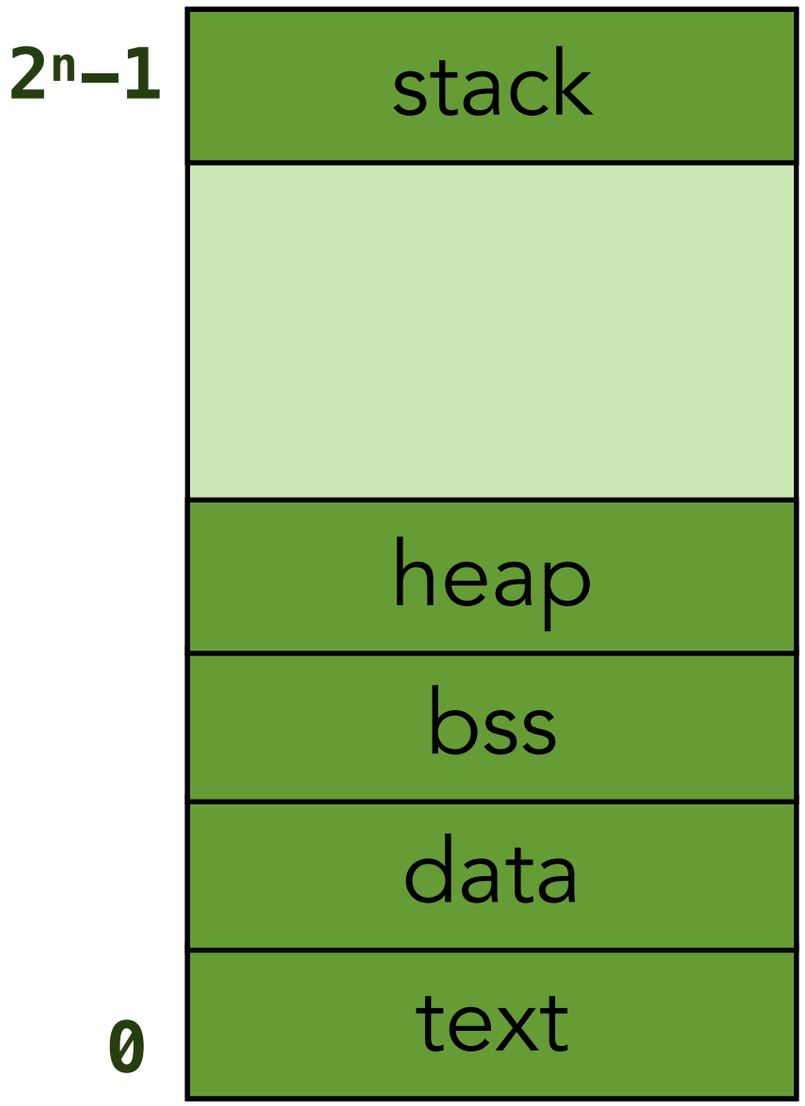


Memória



Código do programa

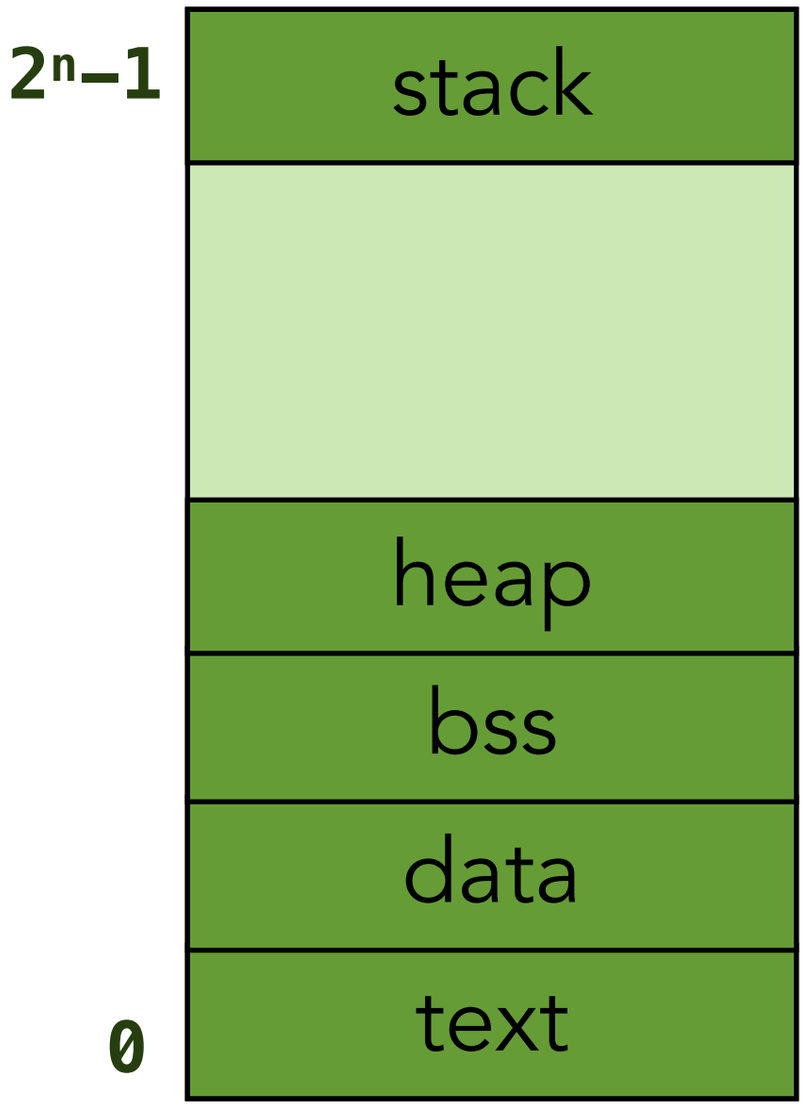
Memória



Variáveis globais inicializadas

Código do programa

Memória

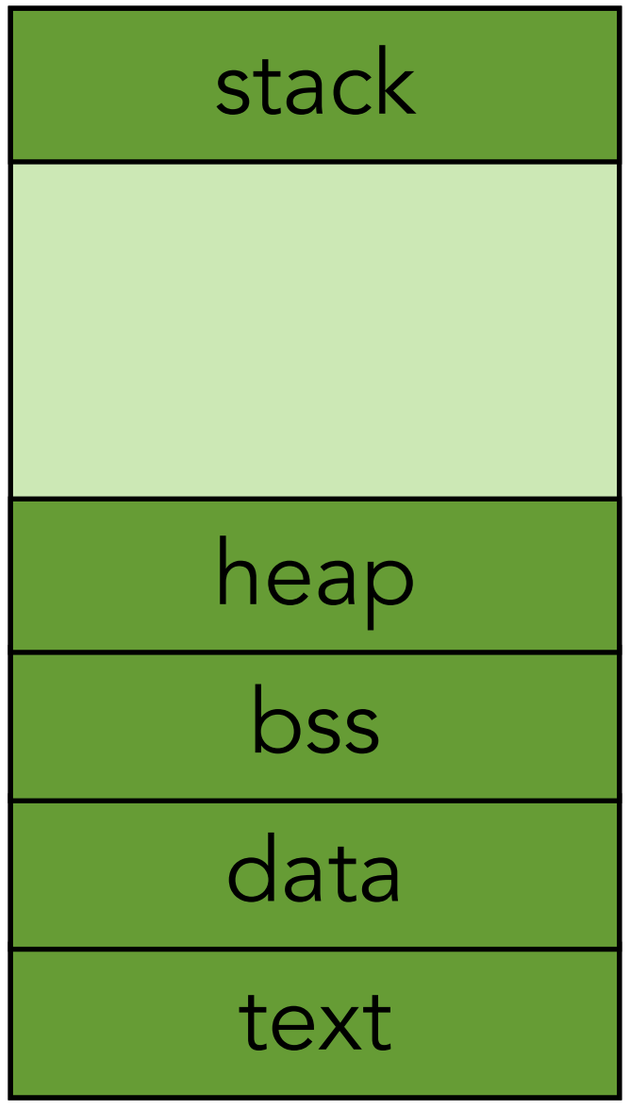


Variáveis globais não inicializadas

Variáveis globais inicializadas

Código do programa

Memória



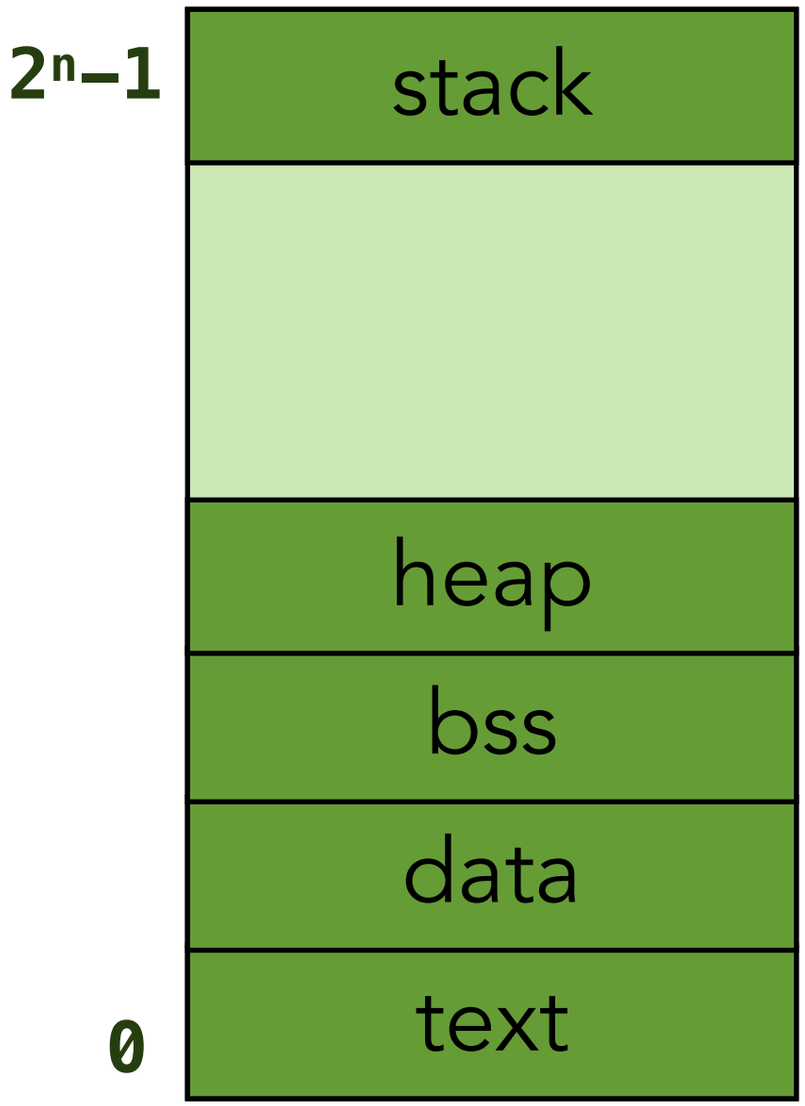
Pilha de dados locais

Variáveis globais não inicializadas

Variáveis globais inicializadas

Código do programa

Memória



Pilha de dados locais

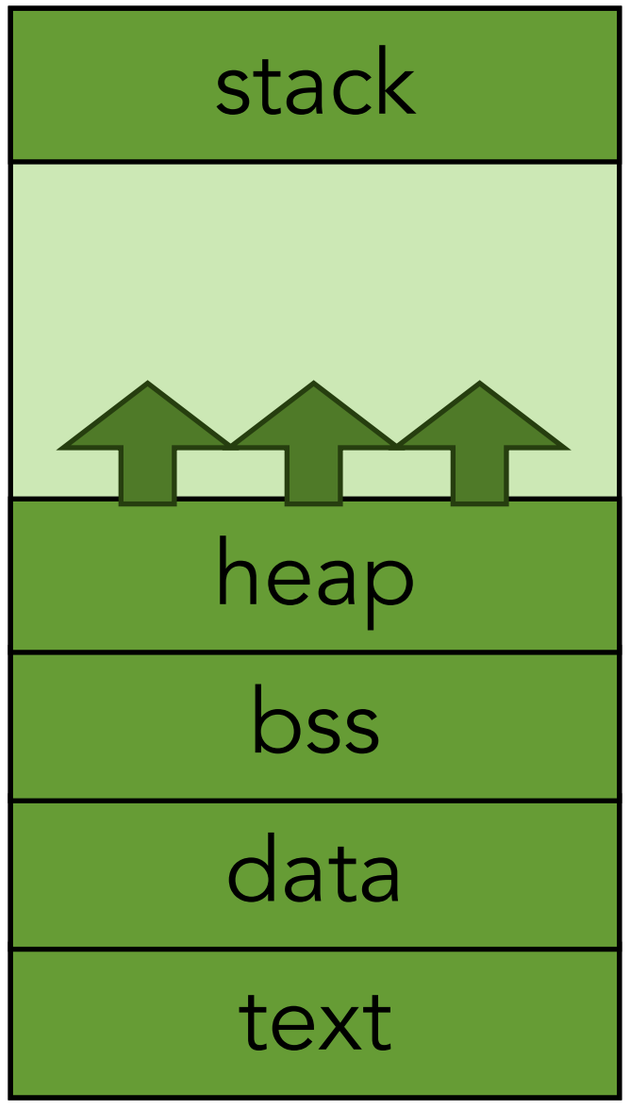
Área alocada dinamicamente

Variáveis globais não inicializadas

Variáveis globais inicializadas

Código do programa

Memória



Pilha de dados locais

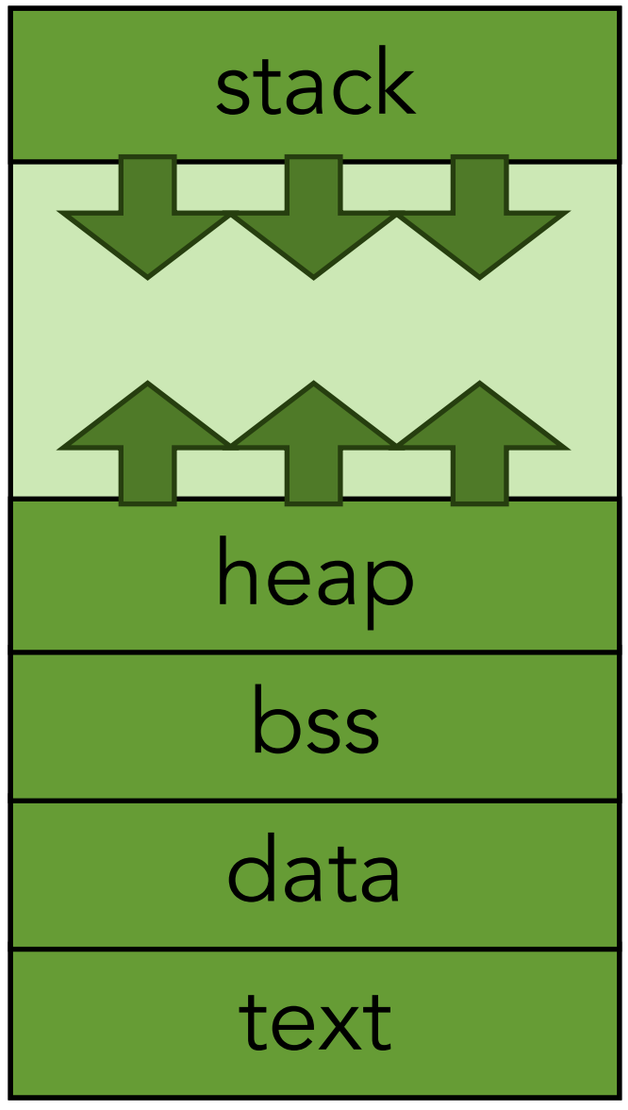
Área alocada dinamicamente

Variáveis globais não inicializadas

Variáveis globais inicializadas

Código do programa

Memória



Pilha de dados locais

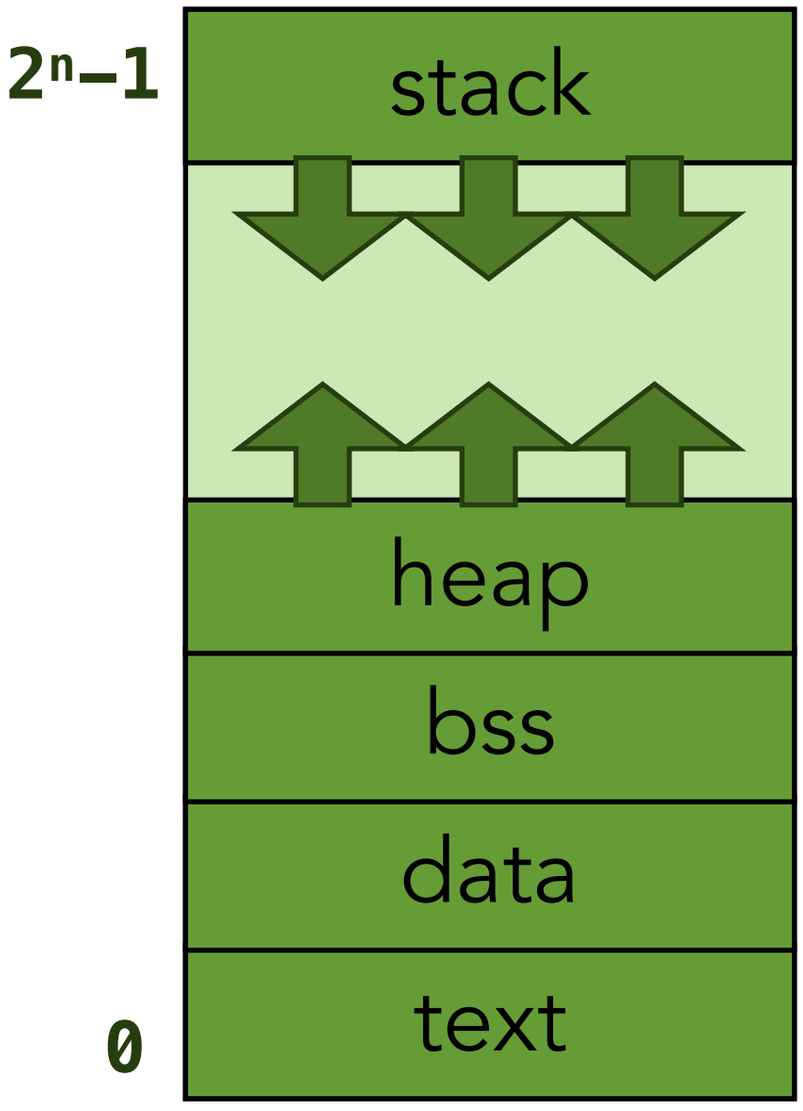
Área alocada dinamicamente

Variáveis globais não inicializadas

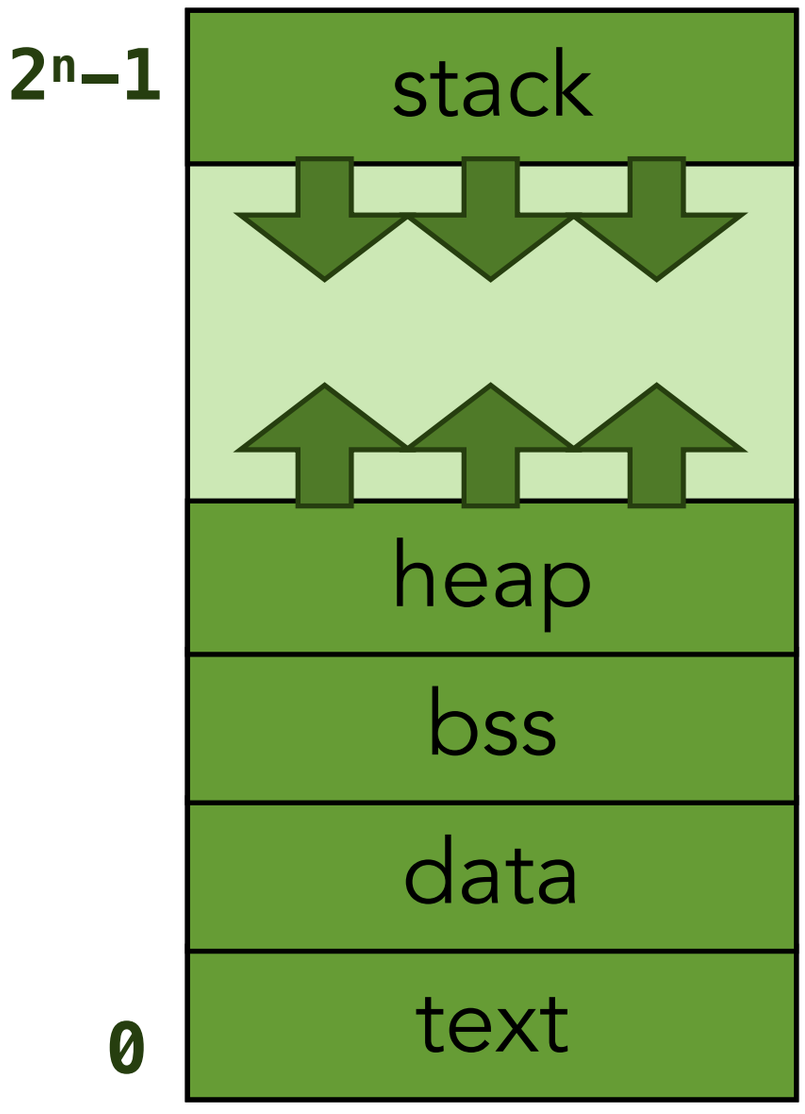
Variáveis globais inicializadas

Código do programa

Memória

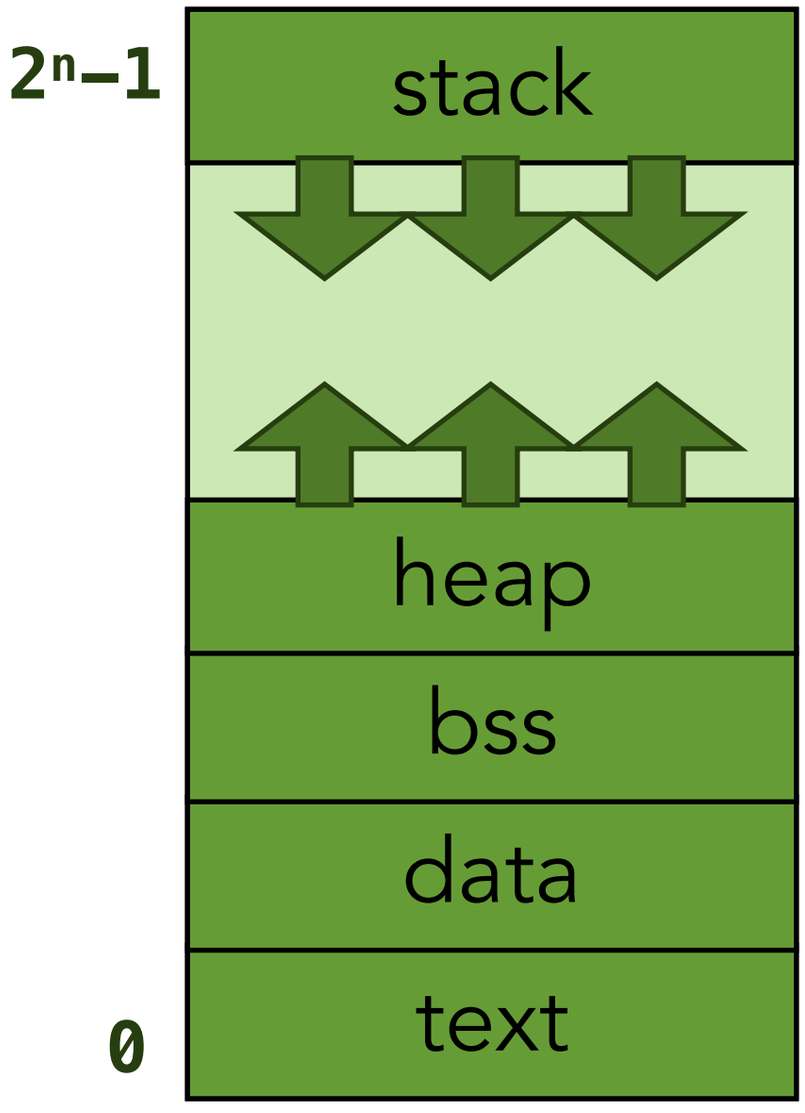


Memória



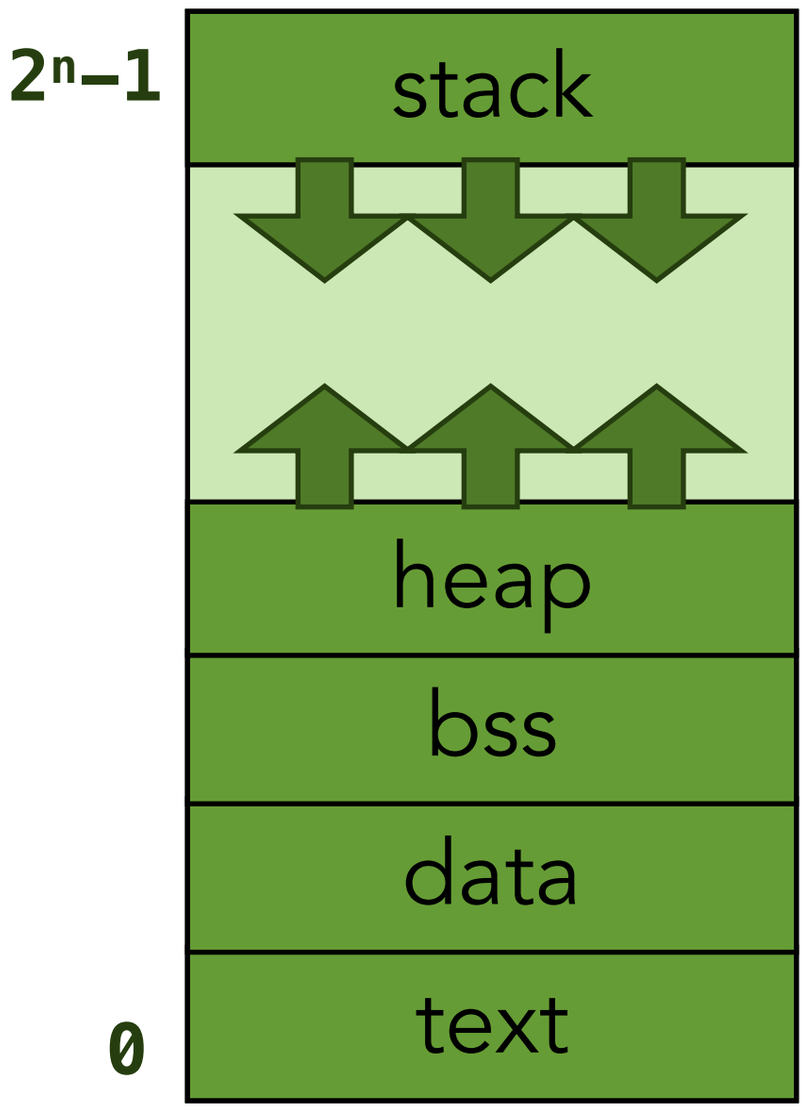

```c
#include <iostream>

int x;
int y=15;

int main(int argc, char *argv[]){

  int *values;
  int i;

  values = new int[5] ;

  for ( i=0 ; i<5 ; ++i )
    values[i] = i;

  free(values);

  return 0;
}
```

Memória



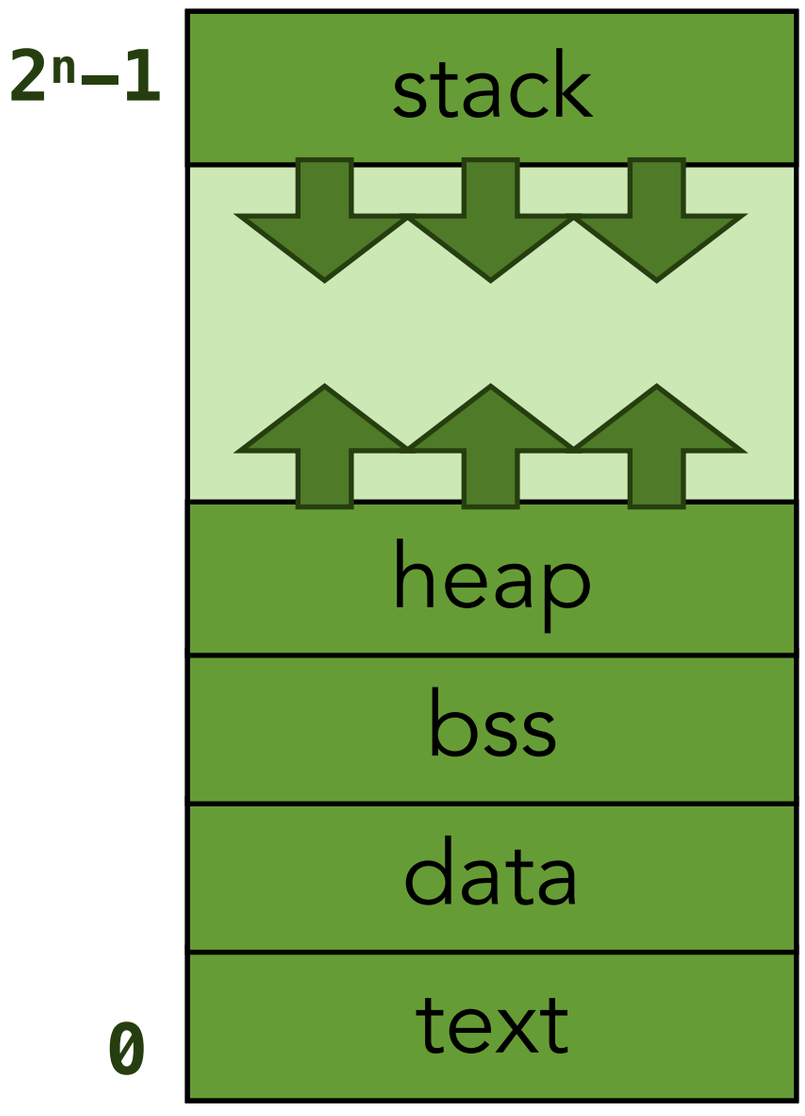

```c
#include <iostream>

int x;
int y=15;

int main(int argc, char *argv[]){

  int *values;
  int i;

  values = new int[5] ;

  for ( i=0 ; i<5 ; ++i )
    values[i] = i;

  free(values);

  return 0;
}
```

Memória



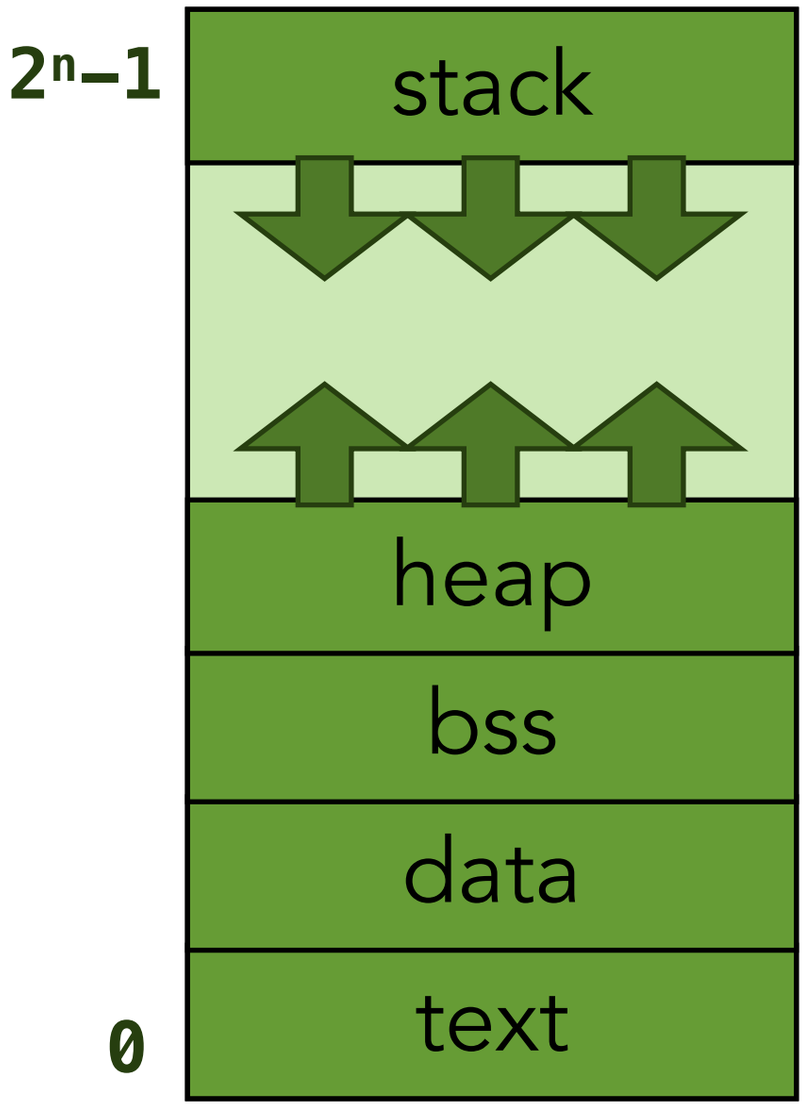

```c
#include <iostream>

int x;
int y=15;

int main(int argc, char *argv[]){

  int *values;
  int i;

  values = new int[5] ;

  for ( i=0 ; i<5 ; ++i )
    values[i] = i;

  free(values);

  return 0;
}
```

Memória



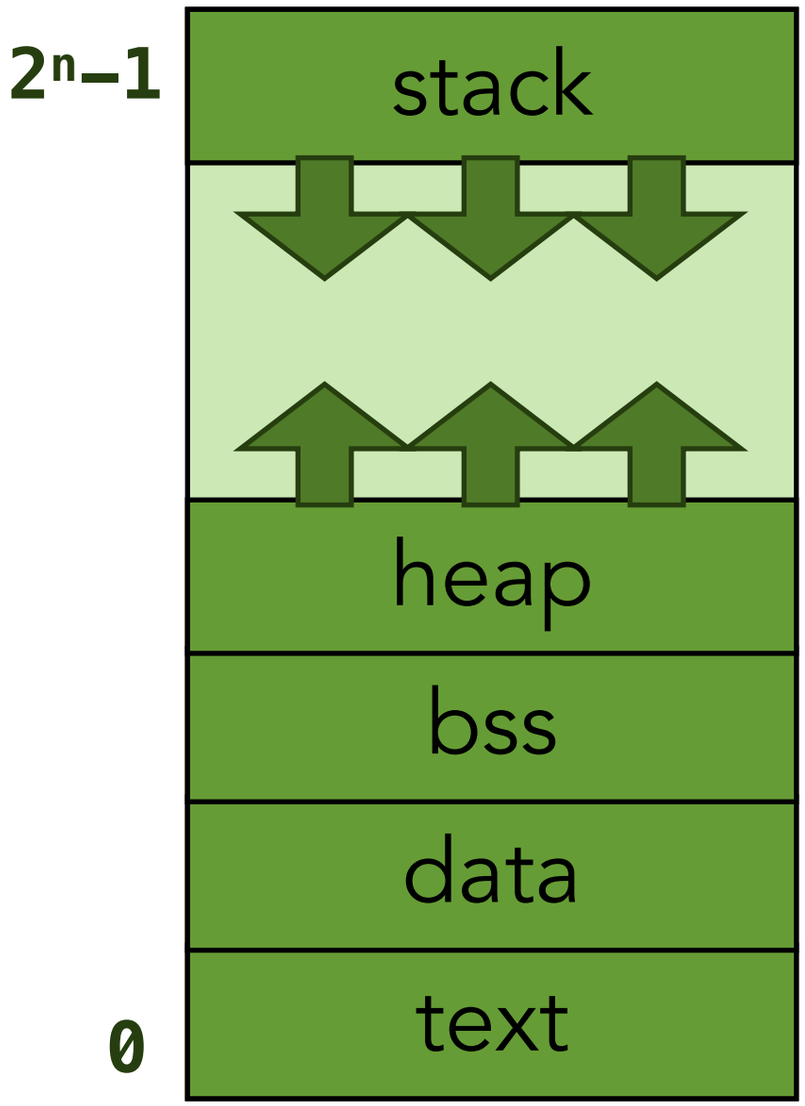

```
#include <iostream>

int x;
int y=15;

int main(int argc, char *argv[]){

  int *values;
  int i;

  values = new int[5] ;

  for ( i=0 ; i<5 ; ++i )
    values[i] = i;

  free(values);

  return 0;
}
```

Memória




```
#include <iostream>

int x;
int y=15;

int main(int argc, char *argv[]){

  int *values;
  int i;

  values = new int[5] ;

  for ( i=0 ; i<5 ; ++i )
    values[i] = i;

  free(values);

  return 0;
}
```

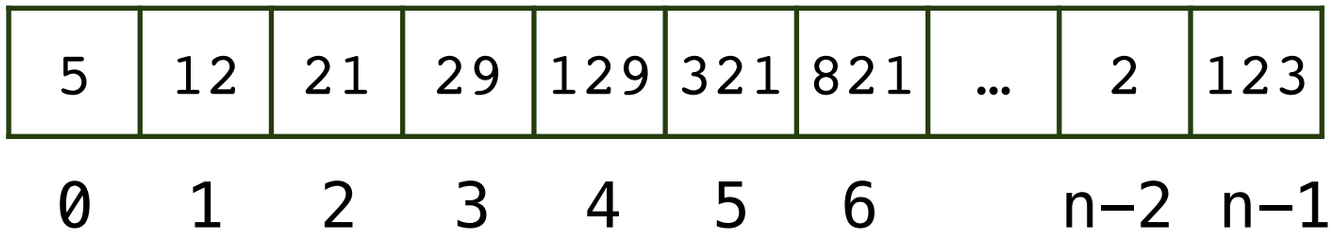
Memória




```
#include <iostream>

int x;
int y=15;

int main(int argc, char *argv[]){

  int *values;
  int i;

  values = new int[5] ;

  for ( i=0 ; i<5 ; ++i )
    values[i] = i;

  free(values);

  return 0;
}
```

Alocação dinâmica

- Programa pede mais memória ao S.O.
  - *Exemplo 1*: Tamanho do *array* só é conhecido durante a **execução** do programa



- *Exemplo 2*: Listas ligadas (veremos adiante)
- *Exemplo 3*: Estruturas de dados dinâmicas (várias - disciplina de Estrutura de dados)

Gerenciamento de memória

- Operadores definidos na especificação C++:
  - `new`: aloca memória
  - `new[]`: aloca memória (*array*)
  - `delete`: libera memória
  - `delete[]`: libera memória (*array*)

O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

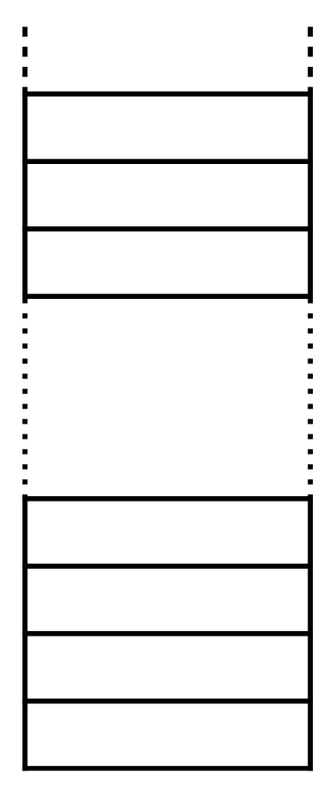
- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```

O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```



- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```

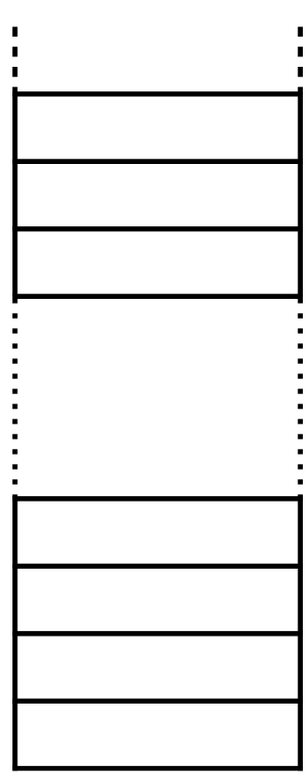
O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```



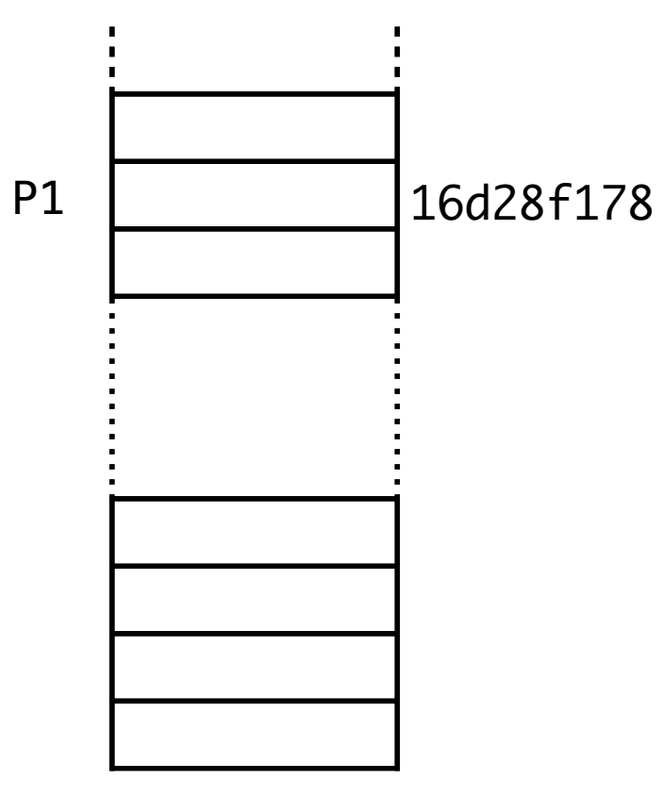
O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

- Exemplo

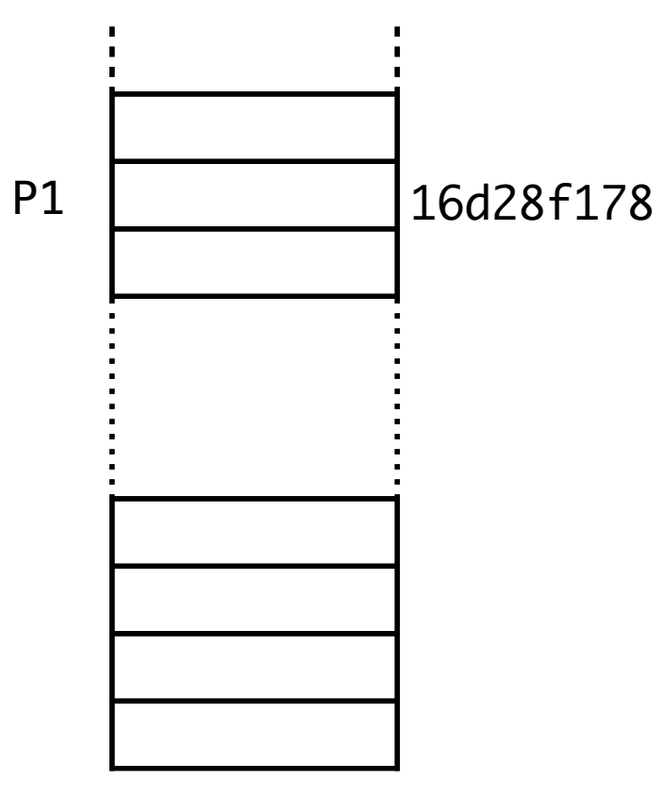
```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```



O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```



- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```

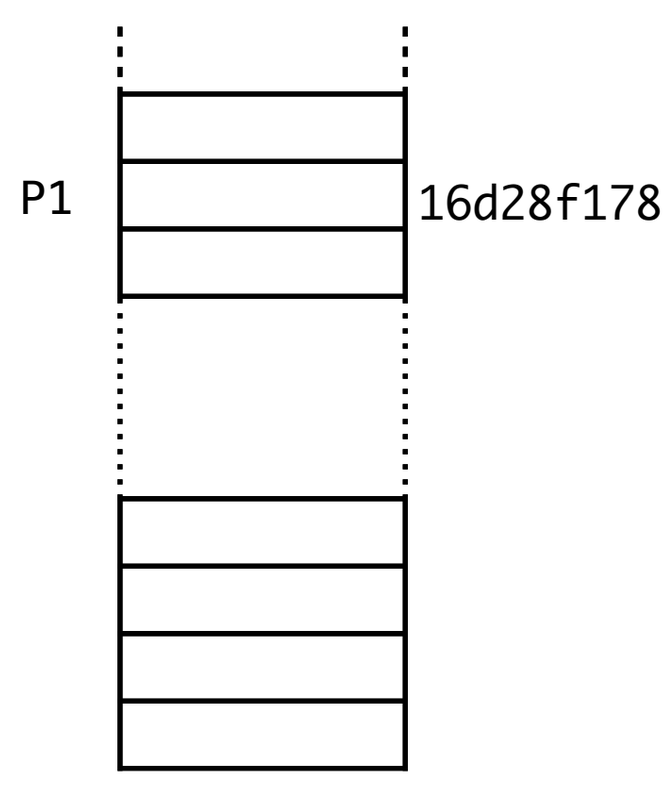
O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```



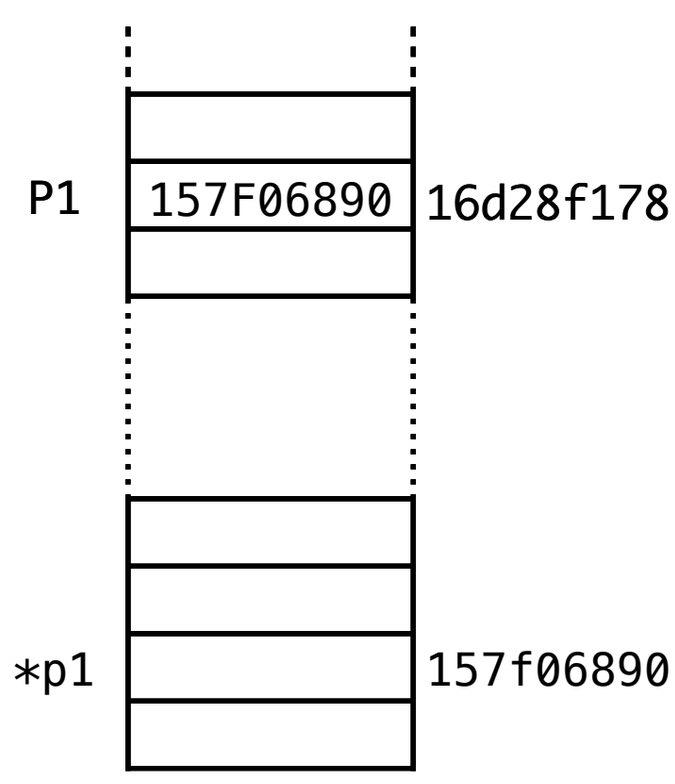
O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```



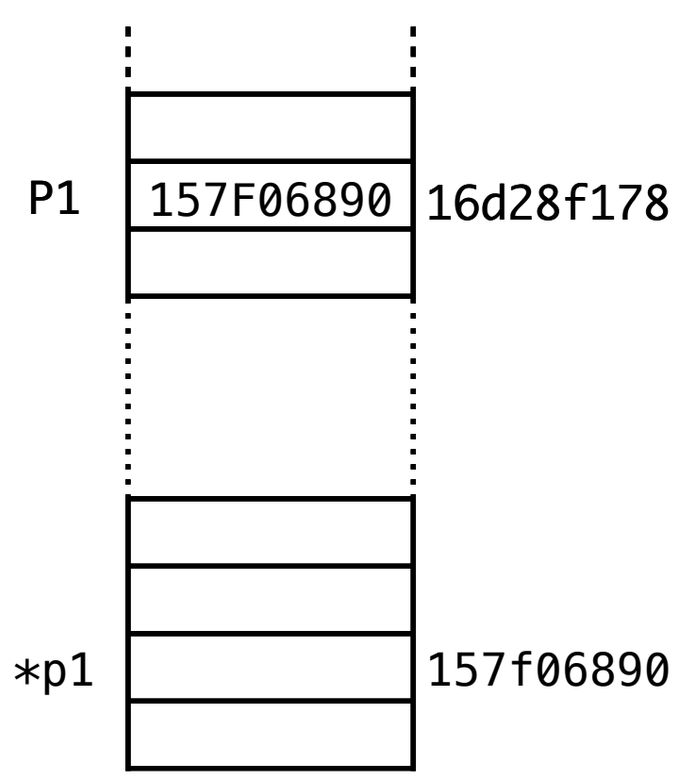
O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```



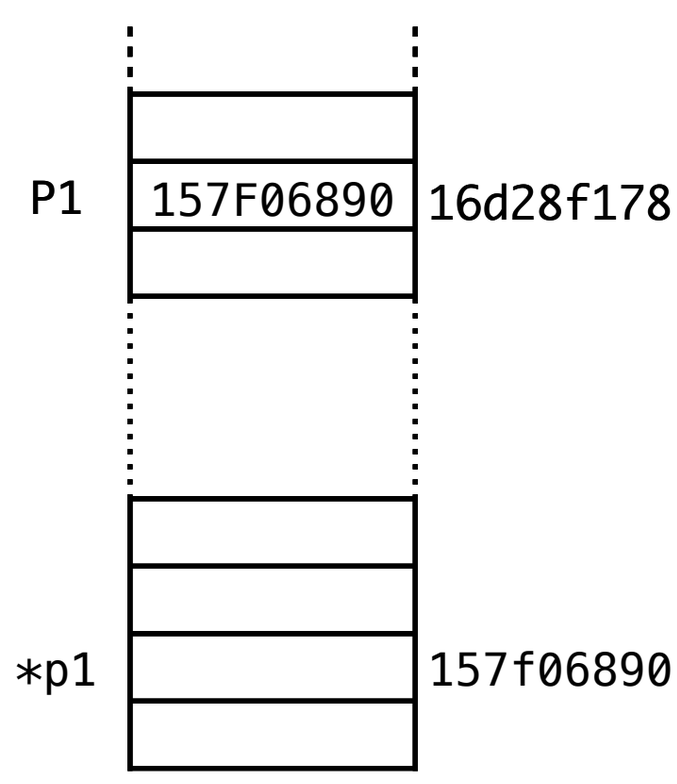
O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```



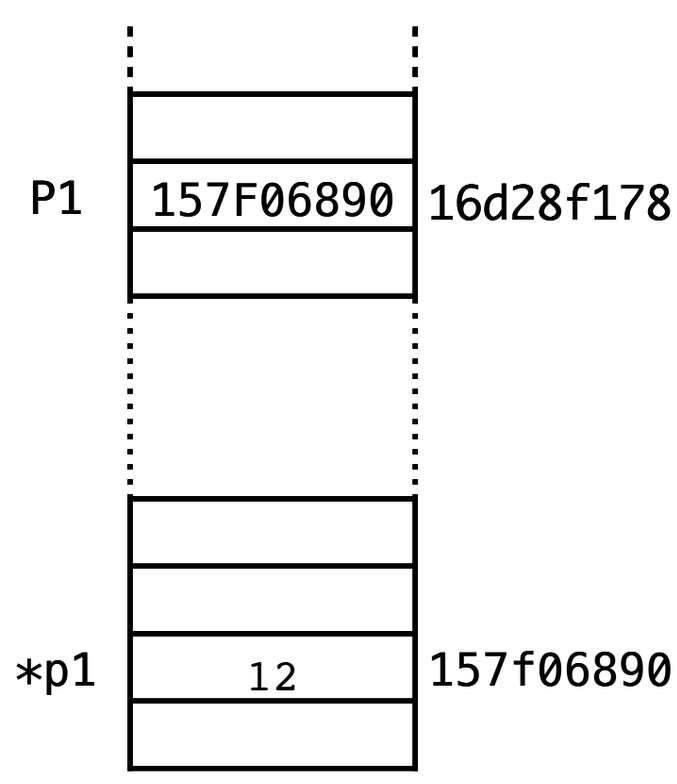
O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```



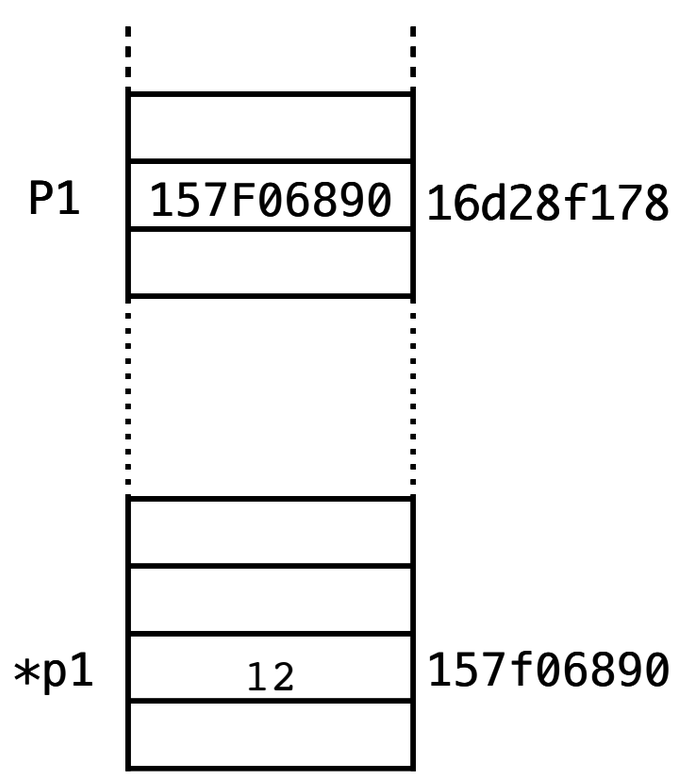
O operador new

- Solicita memória ao S.O.
  - O tamanho da memória é solicitado pelo tipo

```
new TIPO;
int *p1 = new int;
```

- Exemplo

```
int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    return 0;
}
```



Memória precisa ser liberada

- Operador delete

```
delete ENDE;
```

```
#include <iostream>

int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    std::cout << "O valor " << *p1
              << " está no endereço " << p1 << std::endl;
    delete p1;
    return 0;
}
```

Memória precisa ser liberada

- Operador delete

```
delete ENDE;
```

```
#include <iostream>

int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    std::cout << "O valor " << *p1
              << " está no endereço " << p1 << std::endl;
    delete p1;
    return 0;
}
```

Memória precisa ser liberada

- Operador delete

```
delete ENDE;
```

```
#include <iostream>

int main() {
    int *p1;
    p1 = new int;
    std::cin >> *p1; // 12
    std::cout << *p1 << std::endl;
    std::cout << "O valor " << *p1
              << " está no endereço " << p1 << std::endl;
    delete p1;
    return 0;
}
```

O operador new[]

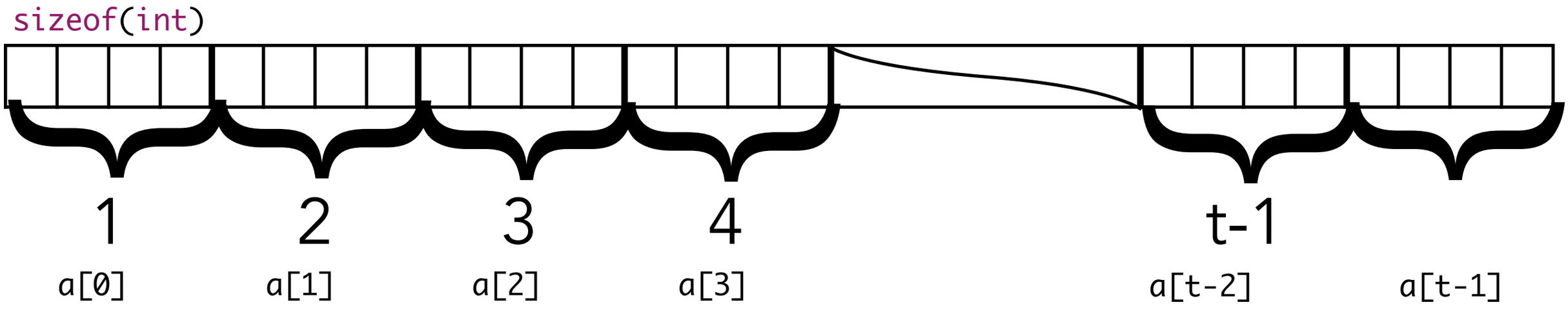
- Solicita área contígua de memória (*array*)

```
int *a, t, i;
std::cin >> t;
a = new int[t];
```

O operador new[]

- Solicita área contígua de memória (*array*)

```
int *a, t, i;
std::cin >> t;
a = new int[t];
```



O operador new[]

```cpp
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
   std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/

  return 0;
}
```

O operador new[]

```
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
   std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/

  return 0;
}
```

O operador new[]

```cpp
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
   std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/

  return 0;
}
```

O operador new[]

```cpp
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
   std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/

  return 0;
}
```

O operador new[]

```cpp
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
   std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/

  return 0;
}
```

O operador new[]

```
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
   std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/

  return 0;
}
```

Uso da memória alocada

O operador new[]

```cpp
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
   std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/

  return 0;
}
```

O operador delete[]

- Libera memória alocada

```
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
    std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/
  delete [] a;
  return 0;
}
```

O operador delete[]

- Libera memória alocada

```
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
    std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/
  delete [] a;
  return 0;
}
```

O operador delete[]

- Libera memória alocada

```
#include <iostream>

int main() {
  int *a, t, i;
  std::cin >> t;
  a = new int[t];
  if (a == nullptr) {
    std::cerr << "ERRO ao alocar memória" << std::endl;
    exit(1);
  }
  for (i=0; i<t; i++)
    std::cin >> a[i];
  /*...*/
  for (i=0; i<t; i++){
    std::cout << a[i] << " ";
  }
  std::cout << std::endl;
  /*...*/
  delete [] a;
  return 0;
}
```

Prática

- Dúvidas?

Programação

C++

Aula 05

Jorgiano Vidal