Ponteiro

ECM404

Definição

- ***Variável***: é um espaço reservado de memória utilizado para armazenar um ***valor*** que pode ser modificado pelo programa.
- ***Ponteiro:*** é um espaço reservado de memória utilizado para armazenar o ***endereço de memória*** de uma outra variável.
  - Um ponteiro é uma variável como qualquer outra do programa – sua diferença é que ela não armazena um valor (inteiro, char, booleano ou float). Utilizado para armazenar endereços de memória (valores inteiros sem sinal).

Declaração

- Como qualquer variável, um ponteiro também possui um tipo:

```
// declaracao de variavel
tipo_variavel nome_variavel

// declaracao de ponteiro
tipo_ponteiro *nome_ponteiro
```

- O ***asterisco (*)*** que informa ao compilador que aquela variável irá armazenar um endereço de memória, onde se encontra um valor armazenado do mesmo tipo que foi declarado, ao invés de um valor.

Declaração

- Exemplos de declaração de ponteiros:

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

  // Declara um ponteiro para int
  int *p;

  // Declara um ponteiro para float
  float *p;

  // Declara um ponteiro para char
  char *p;

  // Declara um ponteiro para struct complexo
  complexo *p;

  return 0;
}
```

Cuidado: ponteiros não inicializados apontam para um lugar indefinido

Declaração e Inicialização

- Exemplos de declaração de ponteiros:

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

  // Declara um ponteiro para int
  int *p1 = NULL;

  // Declara um ponteiro para float
  float *p2 = NULL;

  // Declara um ponteiro para char
  char *p3 = NULL;

  // Declara um ponteiro para struct complexo
  complexo *p4 = NULL;

  return 0;
}
```

Cuidado: ponteiros não inicializados apontam para um lugar indefinido.

Para resolver: podemos utilizar um valor especial (NULL) que é um endereço para nenhum lugar.

Atribuição

- Para armazenarmos o endereço de uma variável em um ponteiro utilizaremos o operador ***(&).*** Ao armazenar o endereço, o ponteiro estará apontando para aquela variável.

```
int main (int argc, char *argv[]){
  int *x, y;

  x = &y;

  return 0;
}
```

UTILIZAÇÃO

- Para acessar o ***valor*** guardado dentro de uma posição na memória apontada por um ponteiro, utilizaremos o operador ***asterisco (*)*** na frente do nome do ponteiro:

```c
int main (int argc, char *argv[]){
  int *x, y = 10;

  x = &y;

  printf("Valor y: %i\n", y);
  printf("Conteudo de onde x aponta: %i\n", *x);

  return 0;
}
```

```
Valor y: 10
Conteudo de onde x aponta: 10
```

Utilização

- Vamos olhar o conteúdo das variáveis:

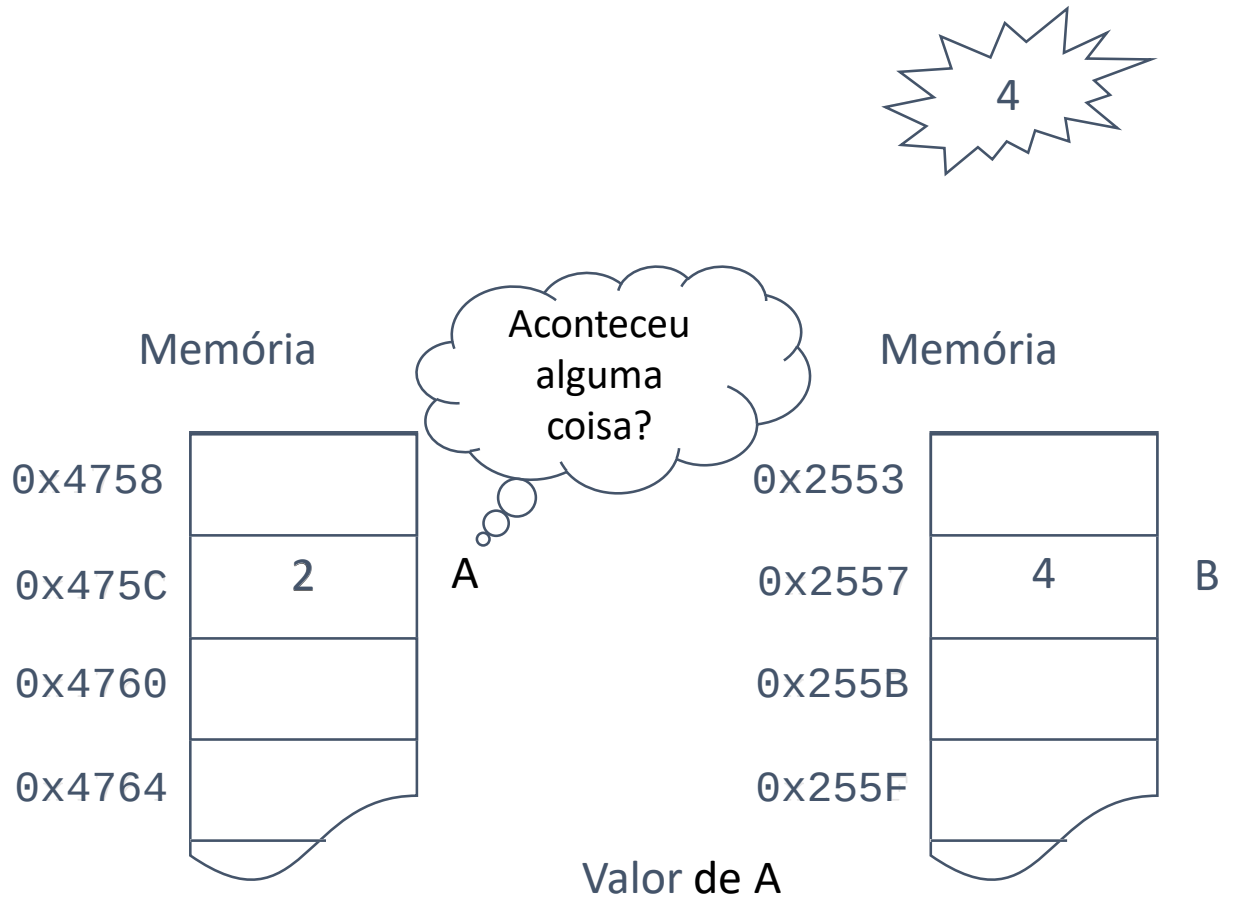
```c
int main (int argc, char *argv[]){
  int *x, y;

  printf("Digite um valor para y: ");
  scanf("%i", &y);

  x = &y;

  printf("Endereco y: %p\n", &y);
  printf("Valor y: %i\n",     y);

  printf("Endereco x: %p\n", &x);
  printf("Valor x: %p\n",     x);
  printf("Conteudo x: %i\n", *x);

  return 0;
}
```

```
Digite um valor para y: 10
Endereco y: 0x7ffcb97abb5c
Valor y: 10
Endereco x: 0x7ffcb97abb60
Valor x: 0x7ffcb97abb5c
Conteudo x: 10
```

Passagem de Parâmetros por Referência

- Utilizando ponteiros na passagem de parâmetros:

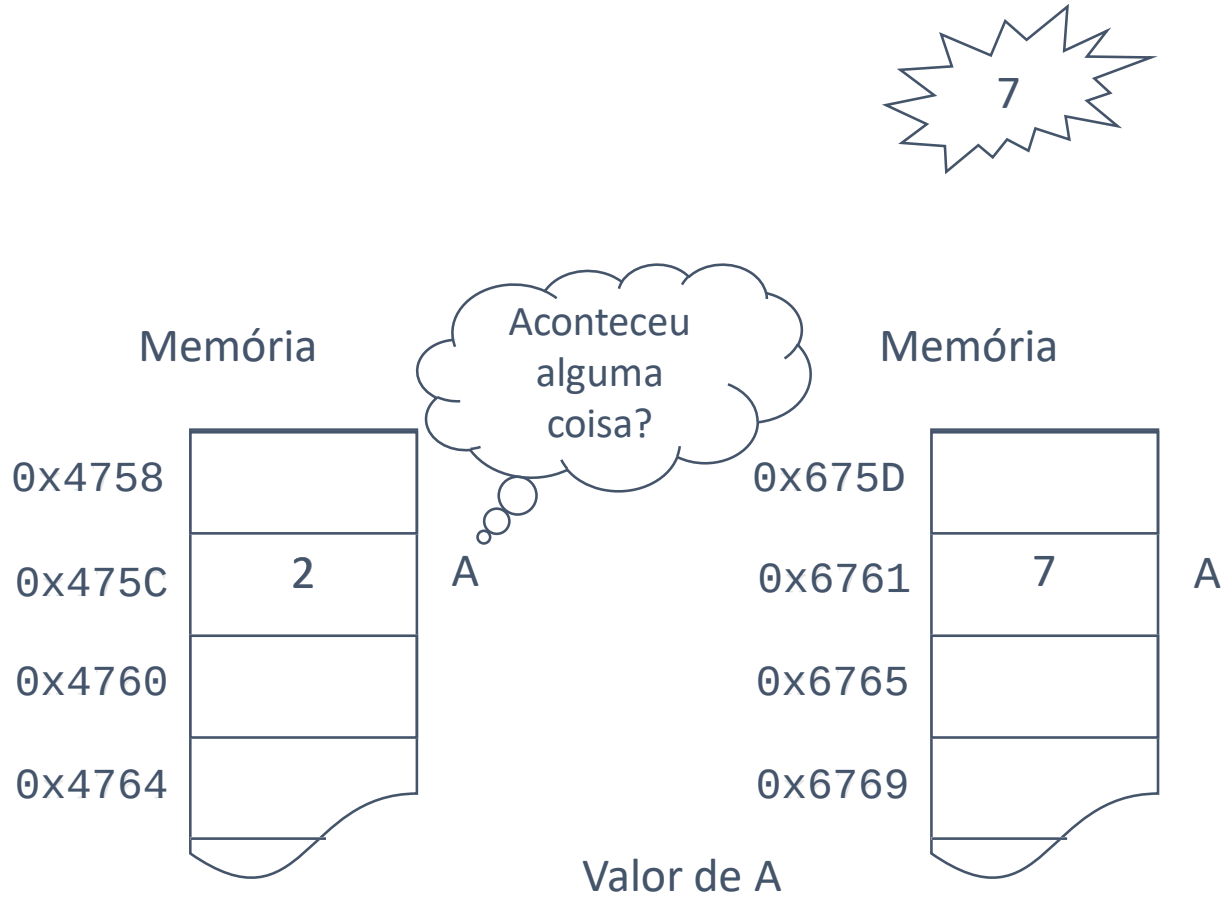
```
int main(int argc, char *argv[]){
  int A;
  scanf("%d", &A);
  ProcEx1(A);
  ...
}

void ProcEx1(int B){
  B = 2*B;
  printf("%d", B);
}
```



Passagem de Parâmetros por Referência

- Utilizando ponteiros na passagem de parâmetros:

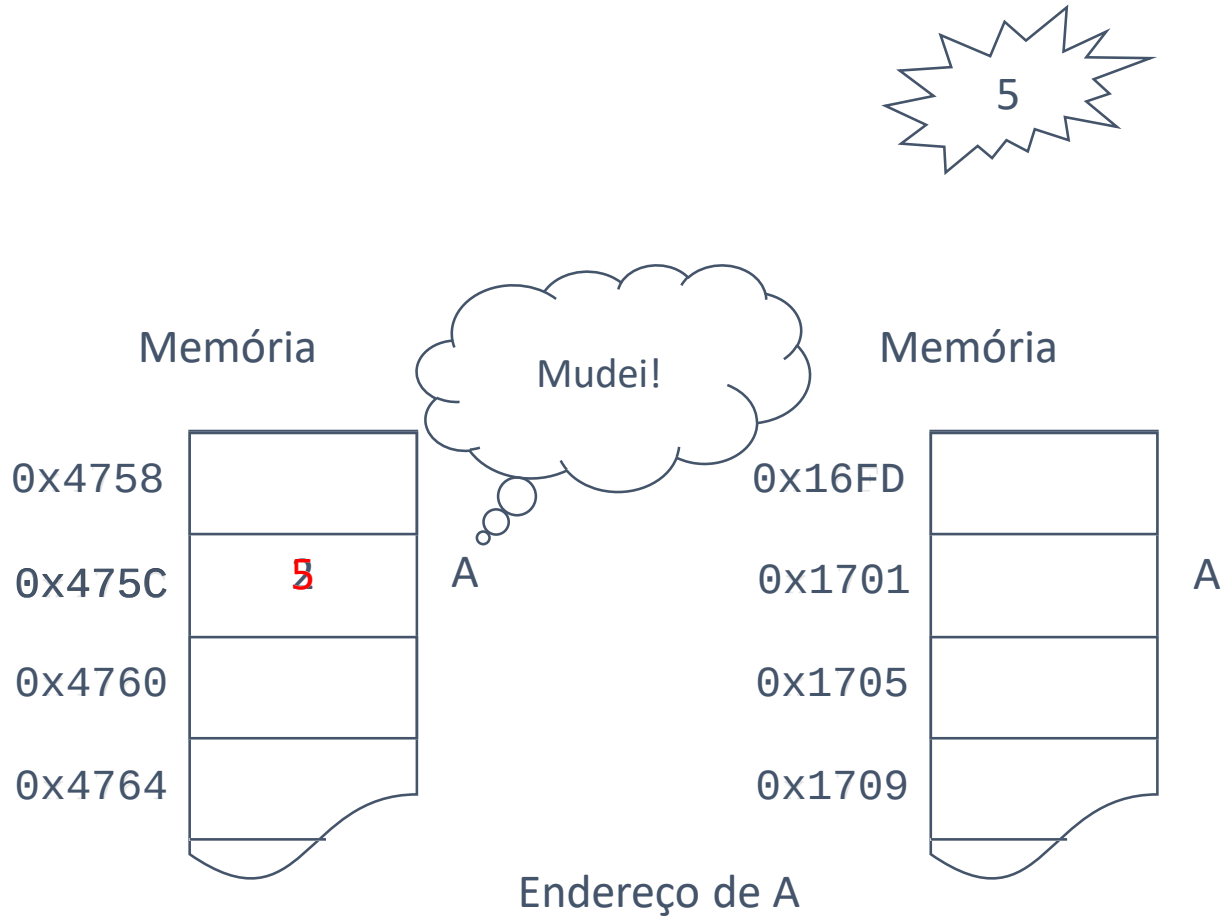
```
int main(int argc, char *argv[]){
  int A;
  scanf("%d", &A);
  ProcEx1(A);
  ProcEx2(A);
  ...
}

void ProcEx2(int A){
  A = 2*A+3;
  printf("%d", A);
}
```



Passagem de Parâmetros por Referência

- Utilizando ponteiros na passagem de parâmetros:

```
int main(int argc, char *argv[]){
  int A;
  scanf("%d", &A);
  ProcEx1(A);
  ProcEx2(A);
  ProcEx3(&A);
  ...
}

void ProcEx3(int *A){
  *A = 3*(*A)-1;
  printf("%d", *A);
}
```



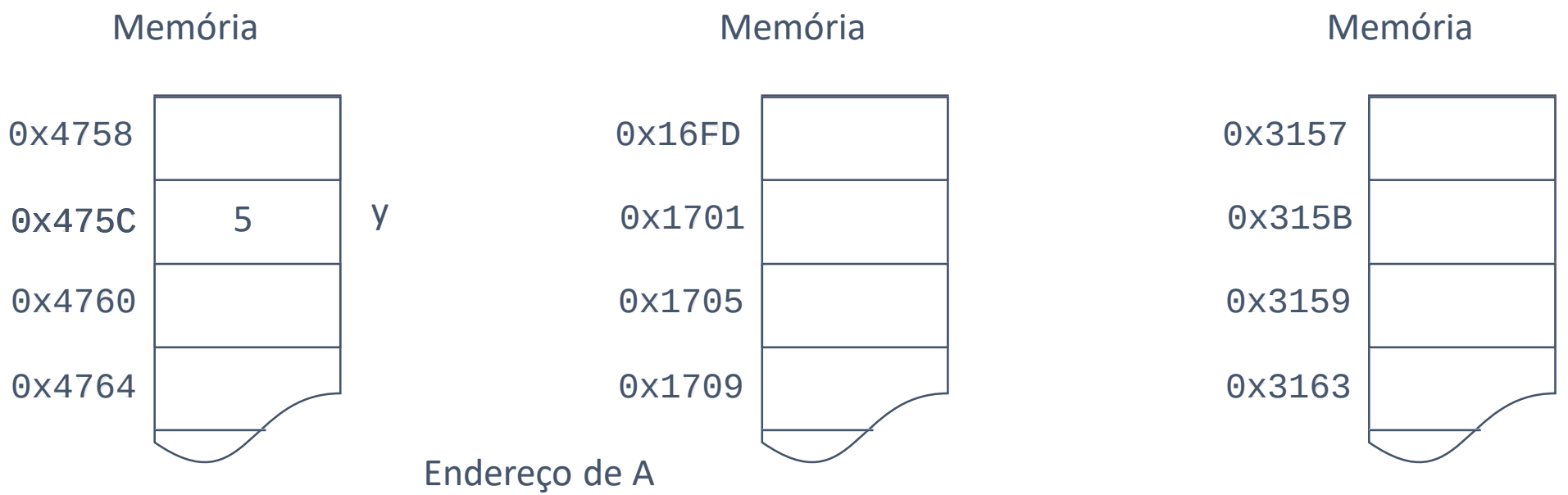
Passagem de Parâmetros por Referência

- ***Cuidado*** ao passar um ponteiro como parâmetro dentro de uma função que recebeu este ponteiro como parâmetro.

```
int main(int argc, char *argv[])
    int y = 5;
    printf("Valor y: %i\n", y);
    algo1(&y);
    printf("Valor y: %i\n", y);

void algo1(int *a){
    *a = *a + 5;
    algo2(a);
}

void algo2(int *b){
    *b = *b * 3;
}
```

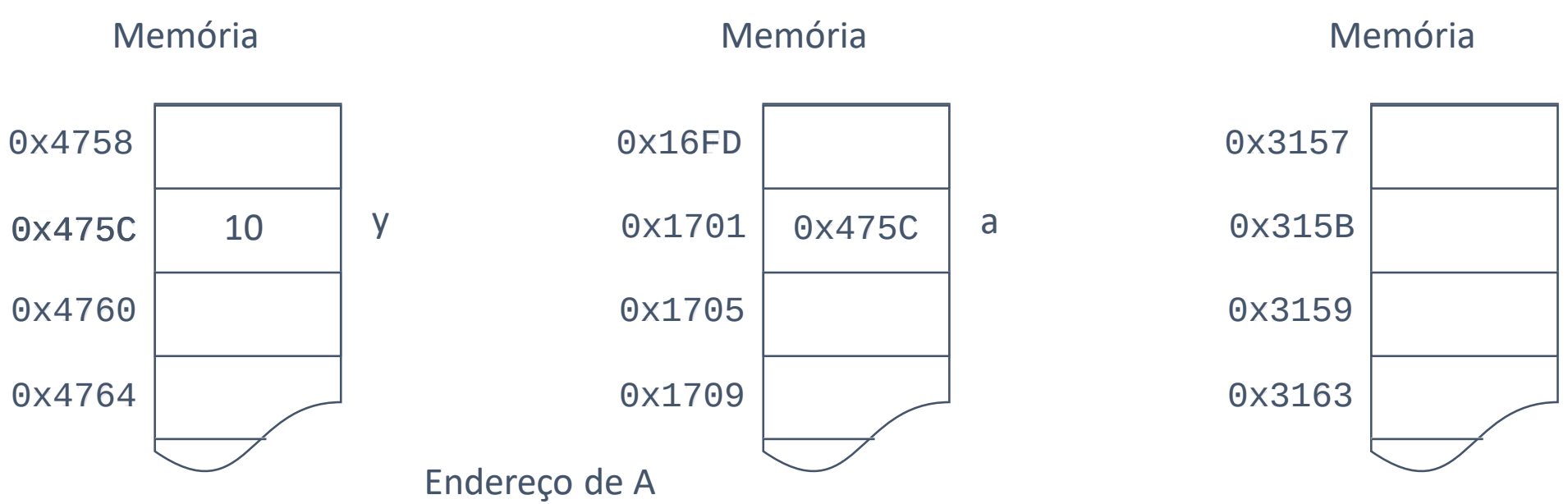
Passagem de Parâmetros por Referência

```
int main(int argc, char *argv[])
    int y = 5;
    printf("Valor y: %i\n", y);
    algo1(&y);
    printf("Valor y: %i\n", y);
```



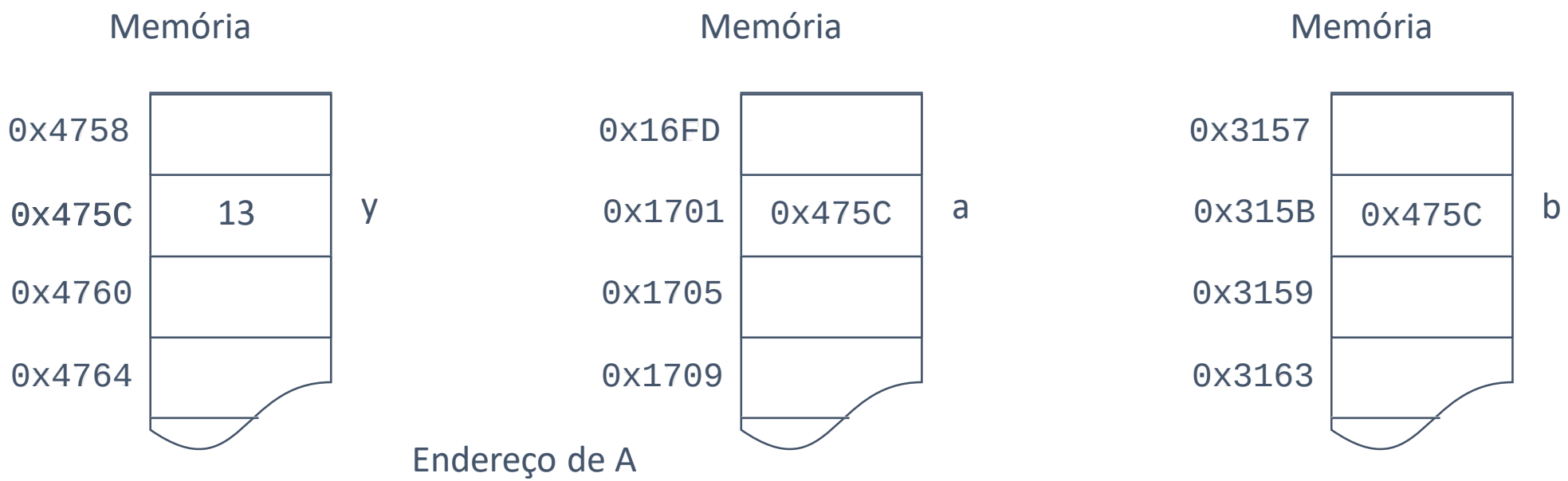
Passagem de Parâmetros por Referência

```c
void algo1(int *a){
    *a = *a + 5;
    algo2(a);
}

void algo2(int *b){
    *b = *b * 3;
}
```



Passagem de Parâmetros por Referência

```
void algo1(int *a){
    *a = *a + 5;
    algo2(a);
}

void algo2(int *b){
    *b = *b * 3;
}
```



Ponteiros e Arrays

- ***Ponteiros*** e ***Arrays*** possuem uma ligação muito forte:
  - Arrays são agrupamentos de dados do mesmo tipo na memória;
  - Quando declaramos um array, pedimos ao computador para reservar uma certa quantidade de memória a fim de se armazenar os elementos do array de forma sequencial;
  - Como resultado dessa solicitação, o computador nos devolve um endereço de memória do começo dessa sequencia de bytes na memória.

Ponteiros e Arrays

- O nome do array, sem índice, é apenas o endereço de memória do primeiro elemento do array.

```c
int main (int argc, char *argv[]){
  int vet[] = {1,2,3,4,5};
  int *p;

  p = vet;

  printf("Terceiro elemento: %i ou %i\n", vet[2], *(p+2));

  return 0;
}
```

```
Terceiro elemento: 3 ou 3
```

Ponteiros e Arrays

- Um outro exemplo:

```
int main (int argc, char *argv[]){
  int vet[7];
  int *p, i;

  p = vet;

  for(i = 0; i < 7; i++)
    *(p+i) = i;

  for(i = 0; i < 7; i++)
    printf("vet[%i] = %i\n",i, vet[i]);

  return 0;
}
```

```
vet[0] = 0
vet[1] = 1
vet[2] = 2
vet[3] = 3
vet[4] = 4
vet[5] = 5
vet[6] = 6
```

Ponteiros e Structs

- Para se utilizar um ponteiro com struct temos que armazenar o endereço do struct no ponteiro e depois o acesso pode ser feito de duas maneiras:

```c
int main (int argc, char *argv[]){
  ponto c;
  ponto *p;

  p = &c;

  (*p).x = 10;
  (*p).y = 20;

  printf("c.x = %.2f e c.y = %.2f\n", c.x,c.y);

  return 0;
}
```

```
c.x = 10.00 e c.y = 20.00
```

Ponteiros e Structs

- Para se utilizar um ponteiro com struct temos que armazenar o endereço do struct no ponteiro e depois o acesso pode ser feito de duas maneiras:

```
int main (int argc, char *argv[]){
  ponto c;
  ponto *p;

  p = &c;

  p->x = 10;
  p->y = 20;

  printf("c.x = %.2f e c.y = %.2f\n", c.x,c.y);

  return 0;
}
```

```
c.x = 10.00 e c.y = 20.00
```

Ponteiros e Array de Structs

- Para se utilizar um ponteiro com um array de struct temos que tomar cuidado com os índices:

```
int main (int argc, char *argv[]){
  ponto lista[3], *p;

  p = lista;

  for(int i=0;i<3;i++){
    p->x = i; p->y = i;
    p++;
  }

  for(int i=0;i<3;i++){
    printf("lista[%i] = { %.2f , %.2f }\n", i,lista[i].x,lista[i].y);
  }

  return 0;
}
```

```
lista[0] = { 0.00 , 0.00 }
lista[1] = { 1.00 , 1.00 }
lista[2] = { 2.00 , 2.00 }
```

Atividade

- Abra a pasta Downloads no VS Code
  - Utilize o comando git clone https://github.com/ECM404/lista7.git --recursive
  - Entre na pasta utilizando o comando cd lista7
  - Inicialize o diretório com o comando make install