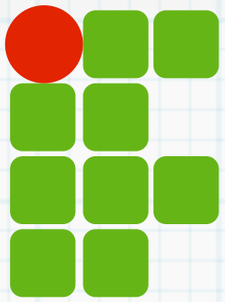


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Algoritmos

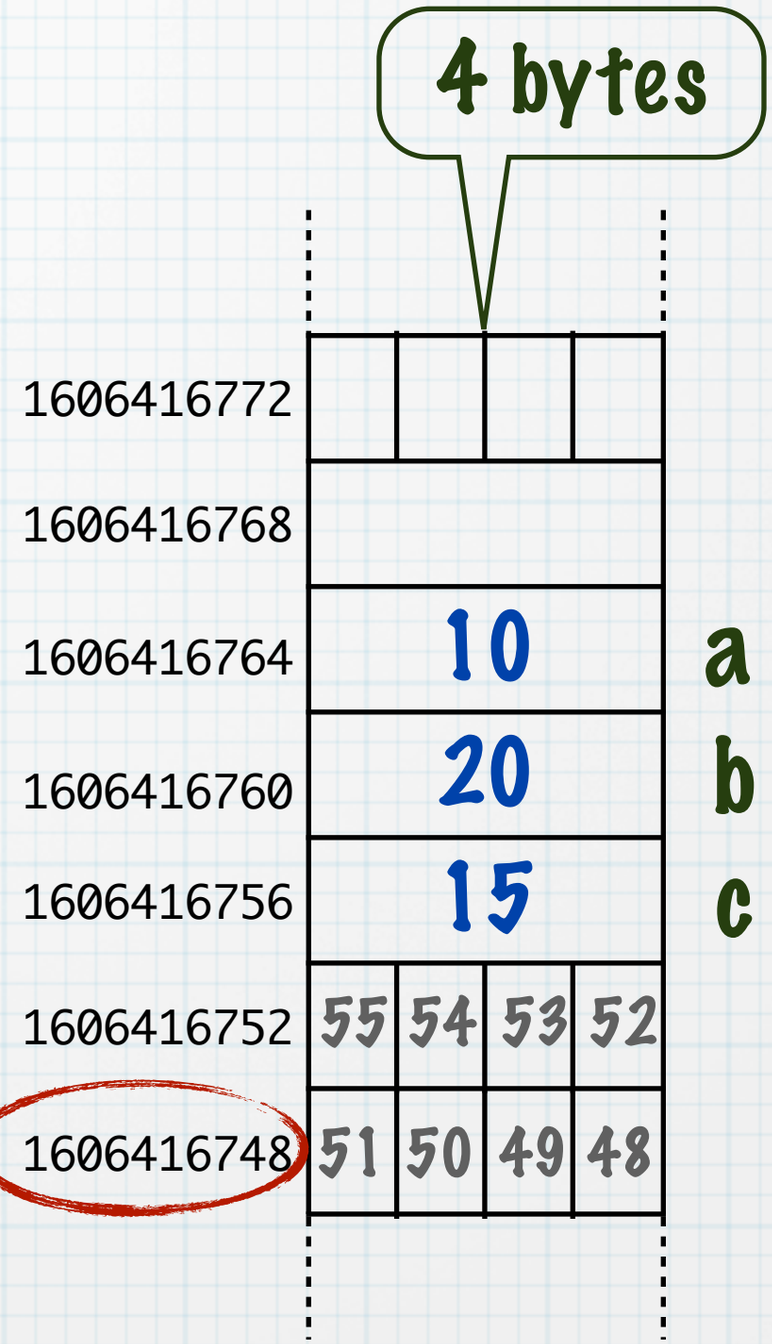
ANSI C - Ponteiros

Agenda

* Memória
* Ponteiros
* Aritmética de ponteiros
* Arrays/strings
* Parâmetros de função
* Exercícios

Antes..

```
int main(int argc, char **argv) {

   int a,b,c;
   a = 10;
   b = 20;
   c = (a+b)/2;
   ...

}
```

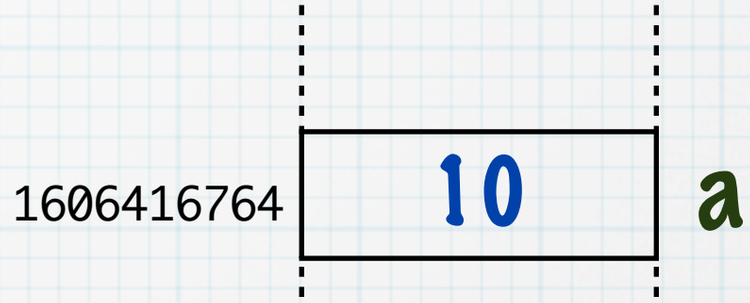


O endereçamento é em bytes

Endereço de memória

* Operador de endereço &
* Lembra do scanf?
  * scanf (“%d”,&a)
* Mostrando o endereço:
  * %p no na função printf

```
&a == 1606416764
```



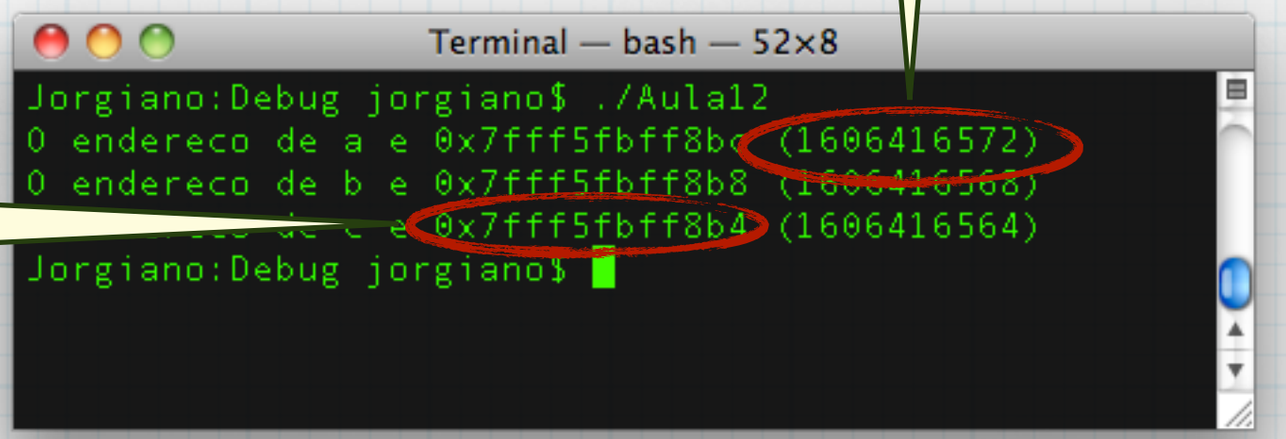

```
printf("O endereço de a e %p \n",&a);
```

Endereço de memória

```c
int main(int argc, char **argv) {
    int a,b,c;
    a = 10;
    b = 20;
    c = (a+b)/2;
    printf("O endereço de a e %p (%d)\n",&a,&a);
    printf("O endereço de b e %p (%d)\n",&b,&b);
    printf("O endereço de c e %p (%d)\n",&c,&c);
    return 0;
}
```

%d mostra o endereço em decimal ("warning" na compilação)

%p mostra o endereço em hexadecimal



Ponteiros

* São variáveis cujo conteúdo é um endereço de memória
  * Tamanho depende da arquitetura do processador/S.O.
  * Usados para manipulação direta da memória
* Declaração, em ANSI C:

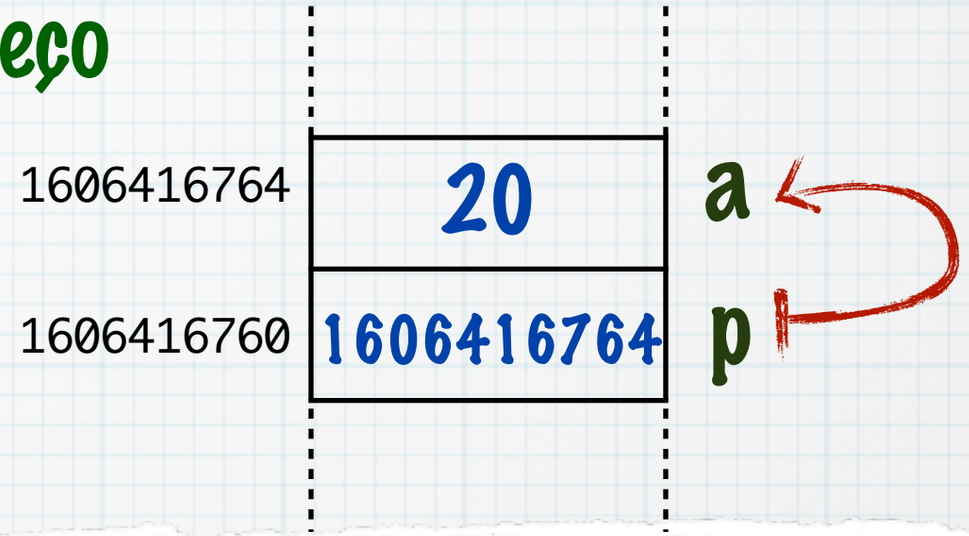
```
int *px;
```

o * indica que a variável é um poneiro

O tipo indica o tipo do conteúdo armazenado no endereços apontato pela variável 'ponteiro'

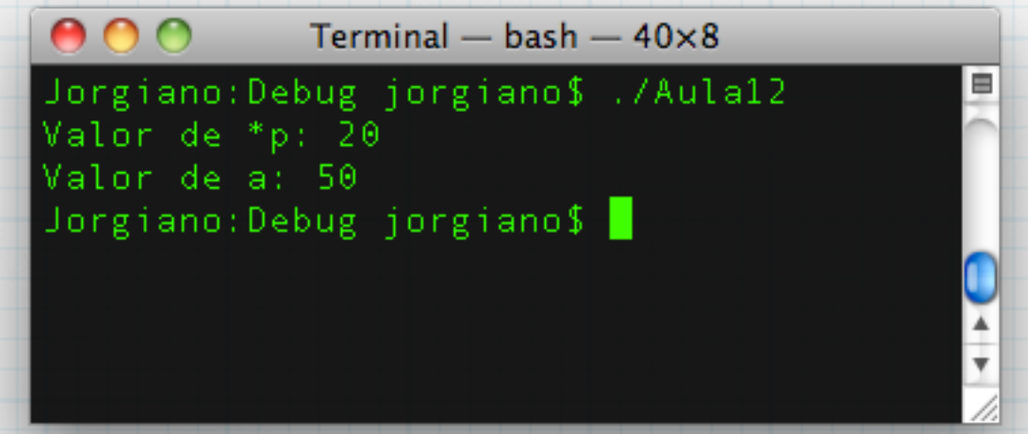
Ponteiros

* Para acessar o conteúdo de um endereço apontado por uma variável ponteiro
  * Operador * (asterisco)



Variável p recebe o endereço da variável a

```
int main(int argc, char **argv) {
   int a = 20;
   int *p;
   p = &a;
   printf("Valor de *p: %d\n",*p);
   *p = 50;
   printf("Valor de a: %d\n",a);
   return 0;
}
```

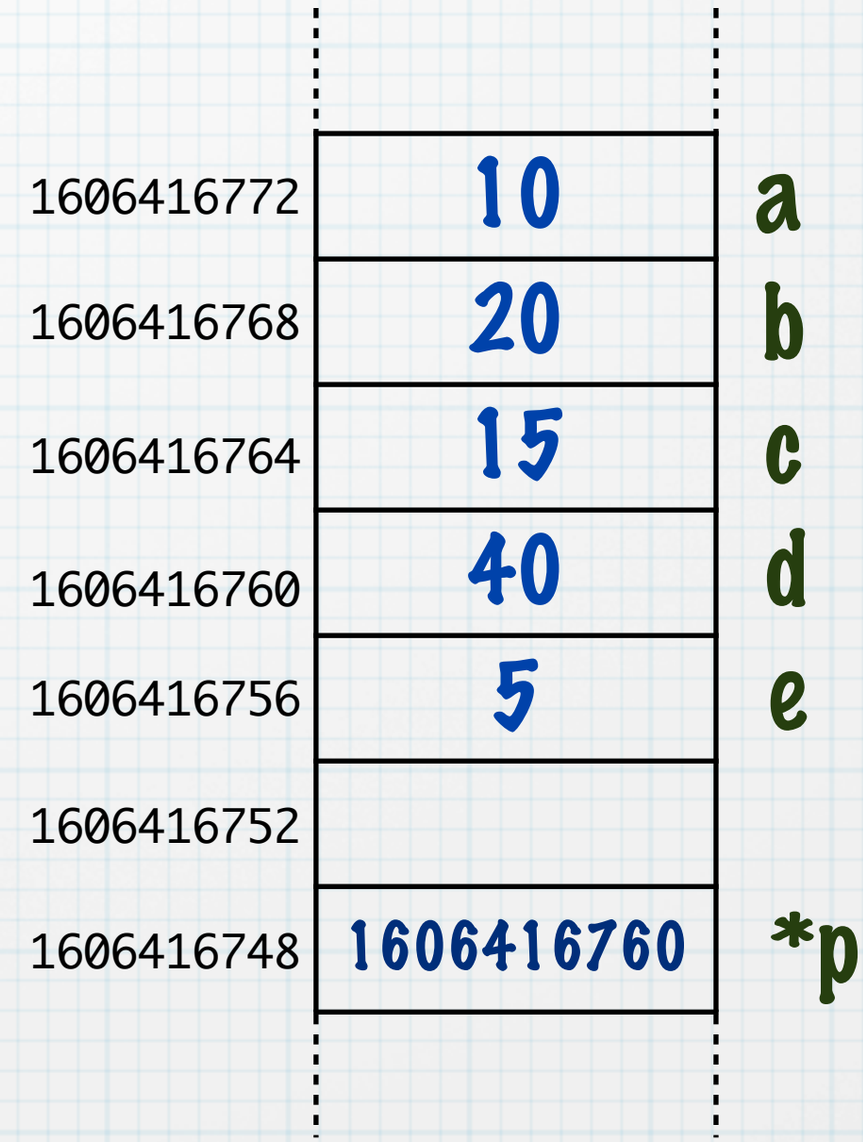


Aritmética de ponteiros

* Endereços de memória
* Podemos adicionar ou subtrair um ponteiro
* Se p é um ponteiro para inteiro,
  * p=p+1 fará que p aponte para o endereço do próximo inteiro
  * Observe que um inteiro possui **4 bytes**
* Também pode ser usado para acessar o espaço de memória a partir de uma base
  * *(p+10) acessa dez espaços a partir de p
    * **Pense em como arrays funcionam!!!!**

Aritmética de ponteiros

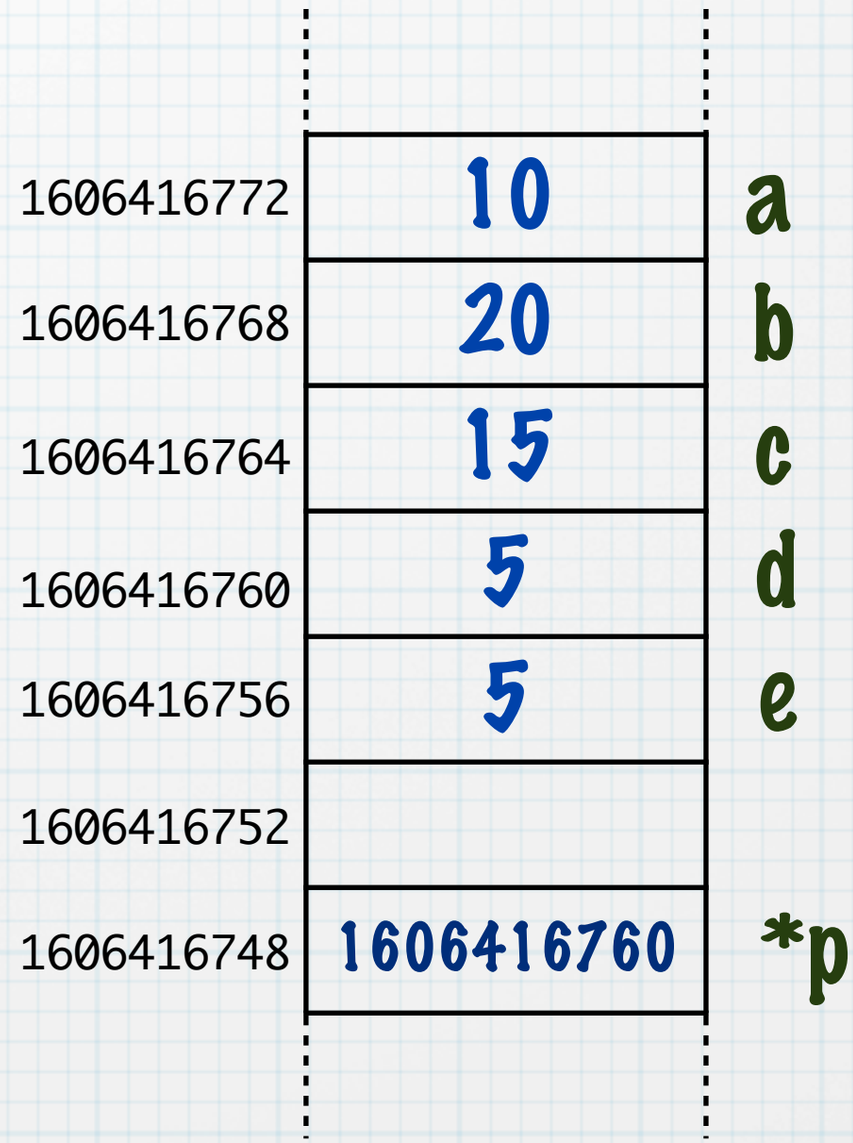
- O que muda com a seguinte expressão?
  - `*p = *(p+1)-*(p+3)`

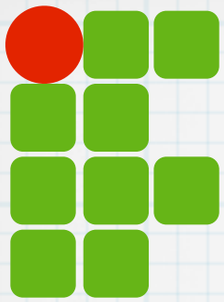


Aritmética de ponteiros

* O que muda com a seguinte expressão?
  * `*p = *(p+1)-*(p+3)`

* O valor de **d** passa a ser **5**



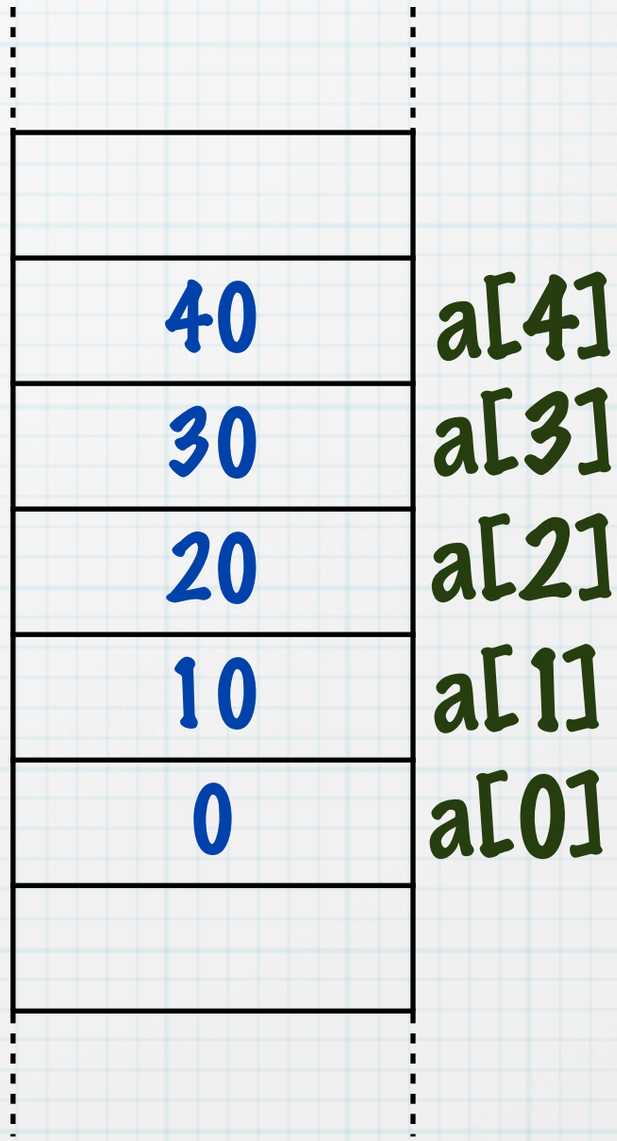


Ponteiros e Arrays

* Arrays e strings
  * 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)

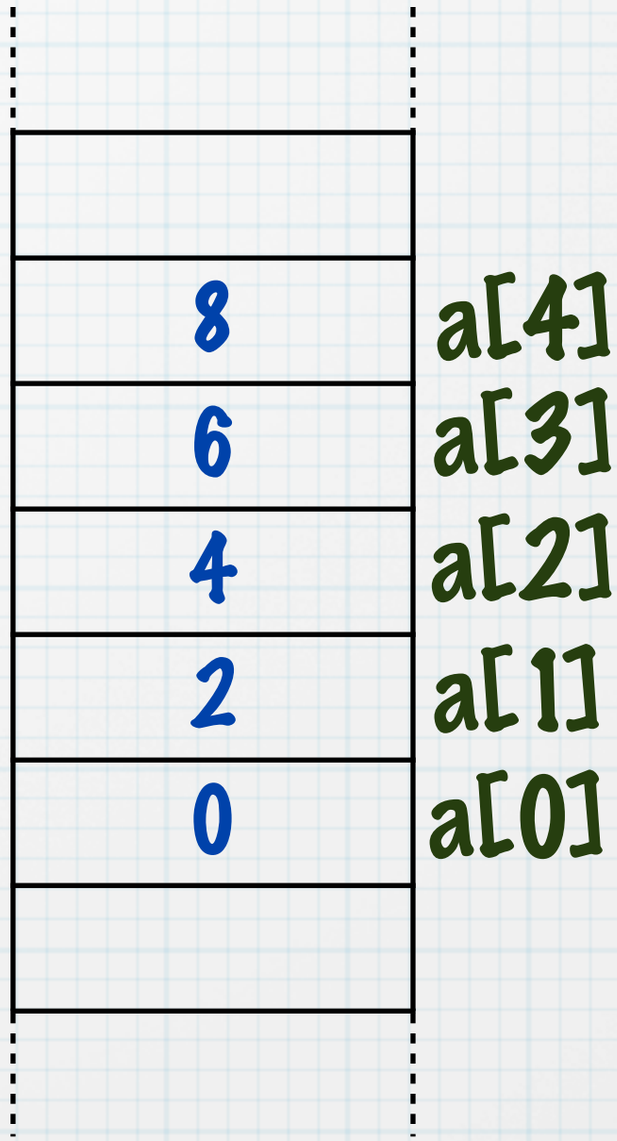
Ponteiros e Arrays

* Considere um array
  * `int a[5]`
* a variável a é um ponteiro para inteiros
  * 5 espaços de memórias consecutivos são reservados
* São equivalentes:
  * a e &a
  * *(a) e a[0]
  * *(a+2) e a[2]



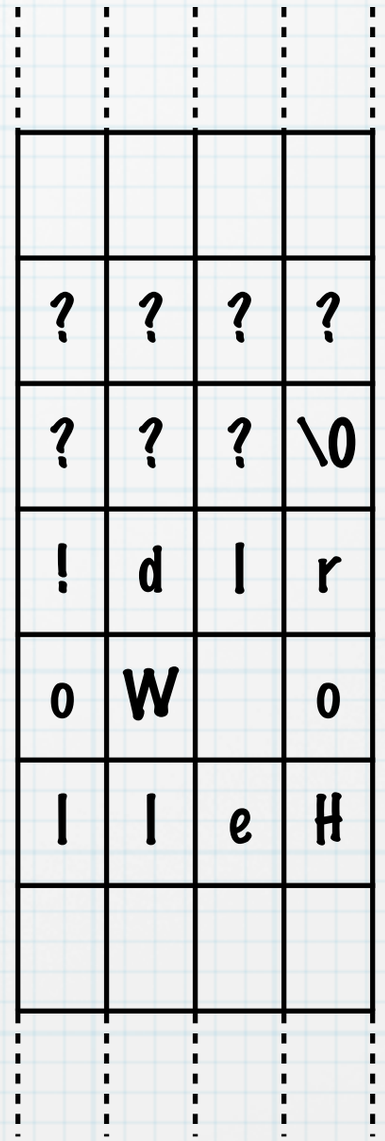
Ponteiros e Arrays

```c
int main(int argc,  char * argv[]){
    int i,a[5];
    int *p;
    for (i=0 ; i<5 ; i++){
        a[i]=i*2;
    }
    p = a;
    for (i=0 ; i<5 ; i++){
        printf("%d ",*p);
        p++;
    }
    return 0;
}
```

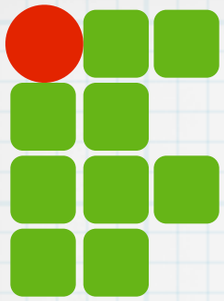


Ponteiros e Strings

```c
int main(int argc, char **argv){
    char nome[20] = "Hello World!";
    char *p;
    p = nome;
    while (*p!='\0'){
        printf("%c",*p);
        p++;
    }
    printf("\n");
    return 0;
}
```

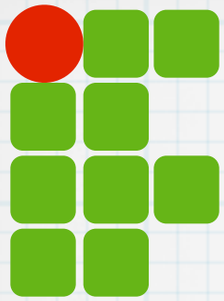


Nesse caso, a operacão "p++" muda em apenas 1 byte (tamanho de um caractere)



Ponteiros

* Arrays são ponteiros para área fixa de memória
  * Constante
  * `int x[10], y[10], *p;`
  * p = x; é válido
  * y = x; NÃO é válido!
  * y = p; NÃO é válido!
  * x = p; NÃO é válido!
* Pode mudar o conteúdo do endereço



Parâmetros de função

- Um ponteiro pode ser passado como parâmetro de função
  - Permite acessar e modificar o conteúdo da variável original
  - Parâmetros Arrays e Strings são apenas ponteiros para a área original da memória

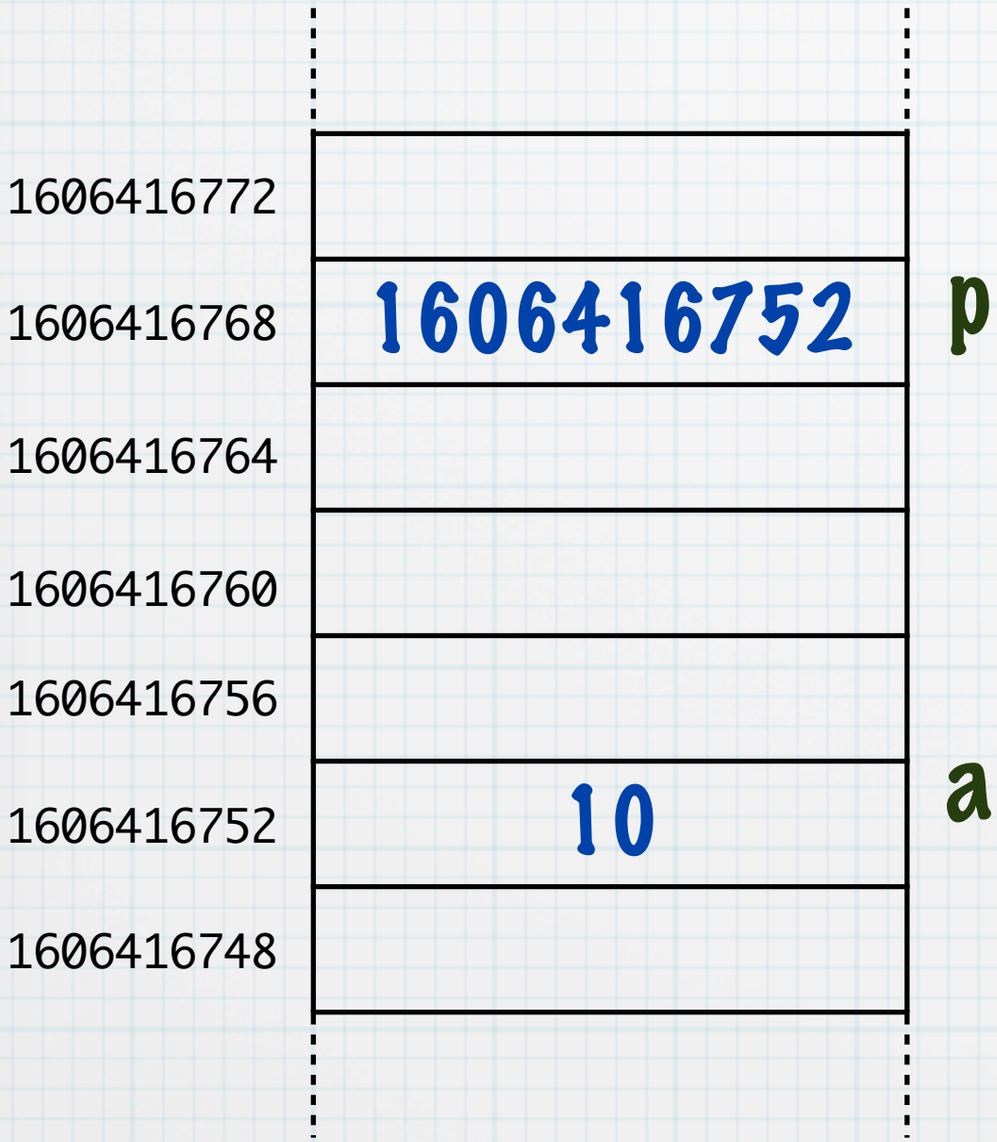
Recebe um ponteiro para inteiro

Modifica o conteúdo de a

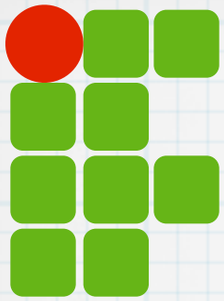
Passa o endereço de a

```
void incrementa(int *p){
   *p = *p + 1;
}

int main(int argc, char **argv){
   int a = 10;
   printf("Valor de a = %d\n",a);
   incrementa(&a);
   printf("Valor de a = %d\n",a);
   return 0;
}
```

Parâmetros de função




```c
void incrementa(int *p){
   *p = *p + 1;
}

int main(int argc, char **argv){
   int a = 10;
   printf("Valor de a = %d\n",a);
   incrementa(&a);
   printf("Valor de a = %d\n",a);
   return 0;
}
```



Parâmetros de função

```c
void converteParaMaiuscula(char *texto) {
    char *ptr;
    ptr = texto;
    char dif = 'a' - 'A';
    while (*ptr != '\0') {
        if (*ptr >= 'a' && *ptr <= 'z') {
            *ptr = *ptr - dif;
        }
        ptr++;
    }
}

int main(int argc, char **argv){
    char nome[20];
     /*...*/
    converteParaMaiuscula(nome);
    /*...*/
    return 0;
}
```

Erros

* 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!

Dúvidas?