

Estruturas Básicas

Aula – Memória

Problema 3

Exibir o maior número inteiro que pode ser representado no computador.



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int x;
    short int y;
    char a;
    unsigned char b;

    x = pow(2,31)-1;      // maior int possível
    y = pow(2,15)-1;      // maior short int possível
    printf("x = %d  y = %d\n",x,y);
    x = x + 1;
    y = y + 1;
    printf("x = %d  y = %d\n",x,y);
    /* -----
       Atribuir os maiores valores possíveis
       para as variáveis a e b.
       ----- */
    a = pow(2,7)-1;
    b = pow(2,8)-1;
    printf("a = %d  b = %d\n",a,b);
    a = a + 1;
    b = b + 1;
    printf("a = %d  b = %d\n",a,b);
    system("PAUSE");
    return 0;
}
```

Qual o maior número inteiro?

- Para o compilador **GCC**, números inteiros são representados usando-se 32 bits (4 bytes).
- Como o bit mais significativo representa o sinal, sobram 31 bits para representar o valor do número (complemento-de-2). O **maior inteiro** será:

01111111111111111111111111111111 = $2^{31} - 1 = 2147483647$

- Como assim?
 - Com **n** bits, podemos representar **2^n** números distintos, sendo o maior número **$2^n - 1$** . Exemplo: para $n = 2$, temos 4 números possíveis, sendo 3 o maior número.

Complemento-de-2

- **Atenção!**

- Na representação em complemento-de-2 existe sempre um valor negativo a mais.

0000	0001	0010	0011	0100	0101	0110	0111
0	+1	+2	+3	+4	+5	+6	+7
1000	1001	1010	1011	1100	1101	1110	1111
-8	-7	-6	-5	-4	-3	-2	-1

Menor inteiro

- Assim, o menor valor inteiro representável não será: -2147833647, mas sim -2147833648.
- Como assim?
 - Com n bits, o menor número representável será -2^{n-1} . Exemplo: para $n = 4$, o menor número representável é $-2^3 = -8$.
- Portanto, as variáveis do tipo `int` poderão armazenar valores no intervalo de -2147833648 a 2147833647.

Modificadores de tipo

- A linguagem C define alguns modificadores de tipo. Alguns deles são: `short`, `long`, `unsigned`.
- Um modificador de tipo altera o intervalo de valores que uma variável pode armazenar.
- Ao tipo `float` não se aplica nenhum dos modificadores, ao tipo `double` aplica-se apenas o modificador `long` e ao tipo `char` aplica-se somente o tipo `unsigned`.
- O modificador de tipo `short` instrui o compilador a representar valores inteiros usando 16 bits.
- Logo, uma variável `short int` pode armazenar valores inteiros no intervalo: -2^{15} a $2^{15} - 1$.

Modificadores de tipo

- Para as variáveis do tipo `char`, o compilador reserva 8 bits.
- Assim, variáveis do tipo `char` podem armazenar valores inteiros no intervalo -2^7 a $2^7 - 1$.
- O modificador de tipo `unsigned` instrui o compilador a não considerar o primeiro bit como sinal. Assim, variáveis `unsigned char` podem representar valores positivos maiores. O maior valor será: $2^8 - 1$.

Modificadores de tipo

No programa `p03.c` são atribuídos os maiores valores possíveis às variáveis `x` e `y`.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int x;
    short int y;
    char a;
    unsigned char b;

    x = pow(2,31)-1; // maior int possível
    y = pow(2,15)-1; // maior short int possível
    printf("x = %d y = %d\n",x,y);
    x = x + 1;
    y = y + 1;
    printf("x = %d y = %d\n",x,y);
    /* -----
       Atribuir os maiores valores possíveis
       para as variáveis a e b.
       ----- */
    a = pow(2,7)-1;
    b = pow(2,8)-1;
    printf("a = %d b = %d\n",a,b);
    a = a + 1;
    b = b + 1;
    printf("a = %d b = %d\n",a,b);
    system("PAUSE");
    return 0;
}
```


Modificadores de tipo

C:\Windows\system32\cmd.exe - a.exe

```
C:\Users\Pedro\Desktop\gcc-test>gcc overflow_example.c
```

```
C:\Users\Pedro\Desktop\gcc-test>a.exe
```

```
x = 2147483647 y = 32767
```

```
x = -2147483648 y = -32768
```

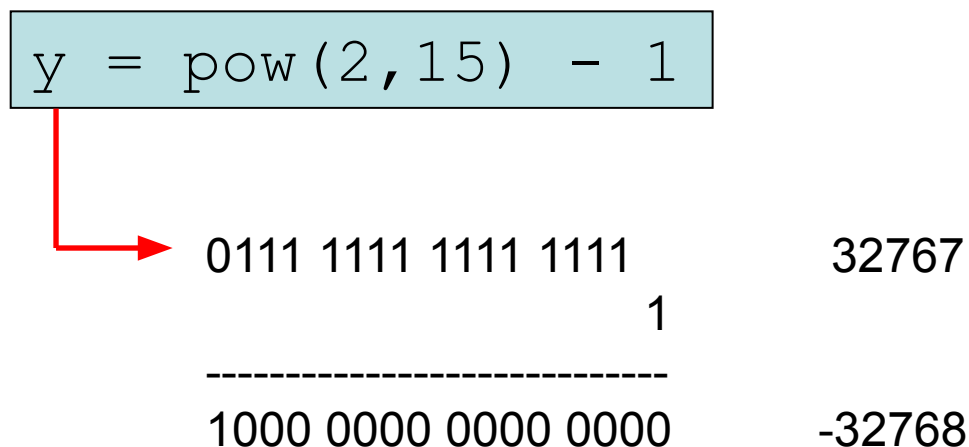
```
a = 127 b = 255
```

```
a = -128 b = 0
```

```
Press any key to continue . . . _
```

Modificadores de tipo

- Em seguida, os valores das variáveis são **incrementados de 1**.
- O que acontece então?
- Ocorre um extravasamento (**overflow**)!
Exemplo: considere a variável **y**.



Modificadores de tipo

- Mais detalhes sobre os modificadores de tipo podem ser vistos aqui:

https://en.wikipedia.org/wiki/C_data_types

Sistema hexadecimal

- O Sistema Hexadecimal (base 16) é o mais usado para representar endereços de memória.
- Grande poder de compactação: consegue representar 1 byte com apenas 2 dígitos!
- Ou seja, cada 4 bits são representados por um único algarismo hexadecimal.
- Neste sistema são utilizados 16 algarismos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Sistema hexadecimal

A tabela abaixo lista a correspondência entre os sistemas **binário**, **decimal** e **hexadecimal**.

Hexa	Decimal	Binário
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Se uma variável acessa o endereço abaixo, como ele é codificado em binário?

0022FF74



0000 0000 0010 0010 1111 1111 0111 0100

Conversão entre sistemas de numeração

- Para converter um valor no sistema hexadecimal para o correspondente valor no sistema binário e vice versa, o que devo fazer?
 - Consulte a tabela exibida na transparência anterior.
- Exemplos:
 - $(1267)_{16} = (0001\ 0010\ 0110\ 0111)_2$
 - $(1010\ 0010)_2 = (A2)_{16}$
 - $(1\ 0100)_2 = (14)_{16}$

Deve-se separar o número binário em blocos de 4 dígitos, da direita para a esquerda:

0001 0100

Conversão entre sistemas de numeração

- Para converter um valor no sistema hexadecimal para o correspondente valor no sistema decimal e vice versa, o que devo fazer?

- Exemplo:

$$(ABAFA)_{16} = (703226)_{10}$$

Casa	Valor da Casa	Lógica	Cálculo	Valor Decimal
5	A	$10 * (16^4)$	$10 * 65536 =$	655 360
4	B	$11 * (16^3)$	$11 * 4096 =$	45 056
3	A	$10 * (16^2)$	$10 * 256 =$	2 560
2	F	$15 * (16^1)$	$15 * 16 =$	240
1	A	$10 * (16^0)$	$10 * 1 =$	10
			Soma	703 226

$$(4711)_{10} = (1267)_{16}$$

Número Decimal	Base	Resultado	Inteiro	Ajuste do Resto	Resto
4711 / 16 =		294,4375	294	$0,4375 * 16 =$	7
294 / 16 =		18,375	18	$0,375 * 16 =$	6
18 / 16 =		1,125	1	$0,125 * 16 =$	2
1 / 16 =		0,0625	0	$0,0625 * 16 =$	1

Endereços de variáveis

- Uma **variável** representa um **nome simbólico** para uma posição de memória.
- Cada posição de memória de um computador possui um **endereço**. Logo, o endereço de uma variável é o endereço da posição de memória representada pela variável.

- Exemplo:
 - Operador para obtenção do endereço da variável
 - Endereço no sistema hexadecimal

```
int x = 3;
printf("%d %p", x, &x);
```

Exibe: 3 0022FF74

Endereços de variáveis

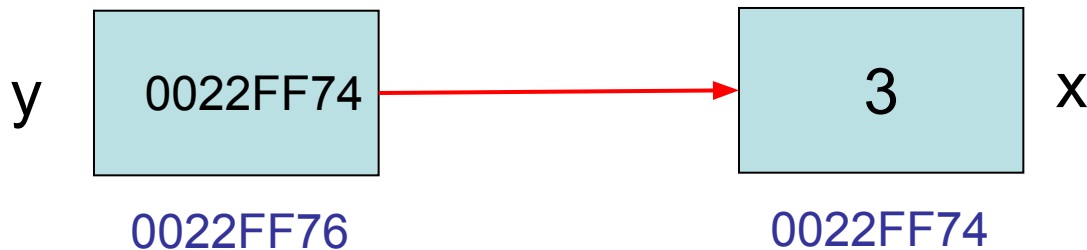
endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347

Endereços de variáveis

- Note que o endereço de uma variável é um valor. Logo, uma variável pode armazenar um endereço.
- Uma variável que armazena um endereço de memória é conhecida como **ponteiro** (*pointer*).
- Daí o porquê do tag usado para exibir endereços de memória ser **%p**.

Endereços de variáveis

- Exemplo: suponha que y armazene o endereço $0022FF74$ de uma posição de memória representada pela variável x e que x contenha o valor inteiro 3.
- Esquematicamente, podemos representar:



- Diz-se que y é um **ponteiro** para x , ou que y **aponta** para x .

Endereços de variáveis


endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74

Endereços de variáveis



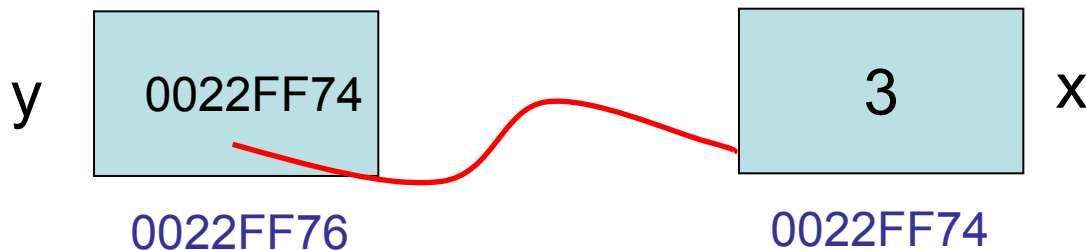
Endereços de variáveis

endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74



Endereços de variáveis

- Qual é o tipo da variável *y*?
 - Para **declarar um ponteiro** é preciso saber para qual tipo de valor este ponteiro irá apontar.
 - Exemplo do caso anterior:



- Neste caso, o ponteiro aponta para um valor inteiro. Assim, diz-se que o tipo de *y* é **int ***.
- A declaração da variável *y* será:

```
int *y;
```

The diagram shows the C code declaration 'int *y;'. A red circle is drawn around the asterisk (*). A red arrow points from this circle down to the text 'Indica que y é um ponteiro (para int, no caso)'.

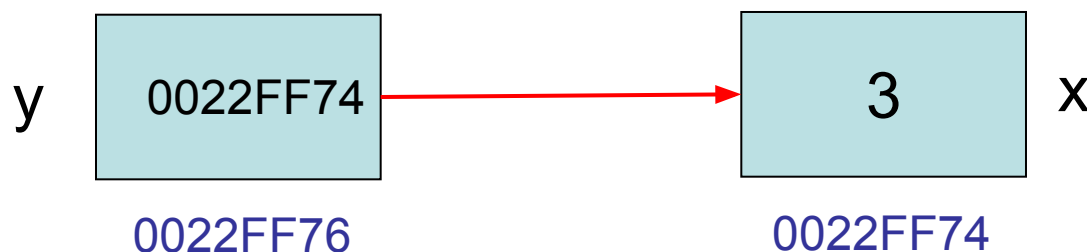
Indica que *y* é um ponteiro (para int, no caso)

Endereços de variáveis

- Como acessar o conteúdo do endereço apontado por *y*?

```
int *y = &x;
```

Indica que *y* é um ponteiro (para int, no caso)



- Usa-se o operador `*` para isso:
 - `printf("O conteúdo do endereço apontado por y é: %d", *y);` //vai imprimir 3

Endereços de variáveis

```
int x = 3;  
int *y;
```

endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	

Endereços de variáveis

```
int x = 3;  
int *y;
```

endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	



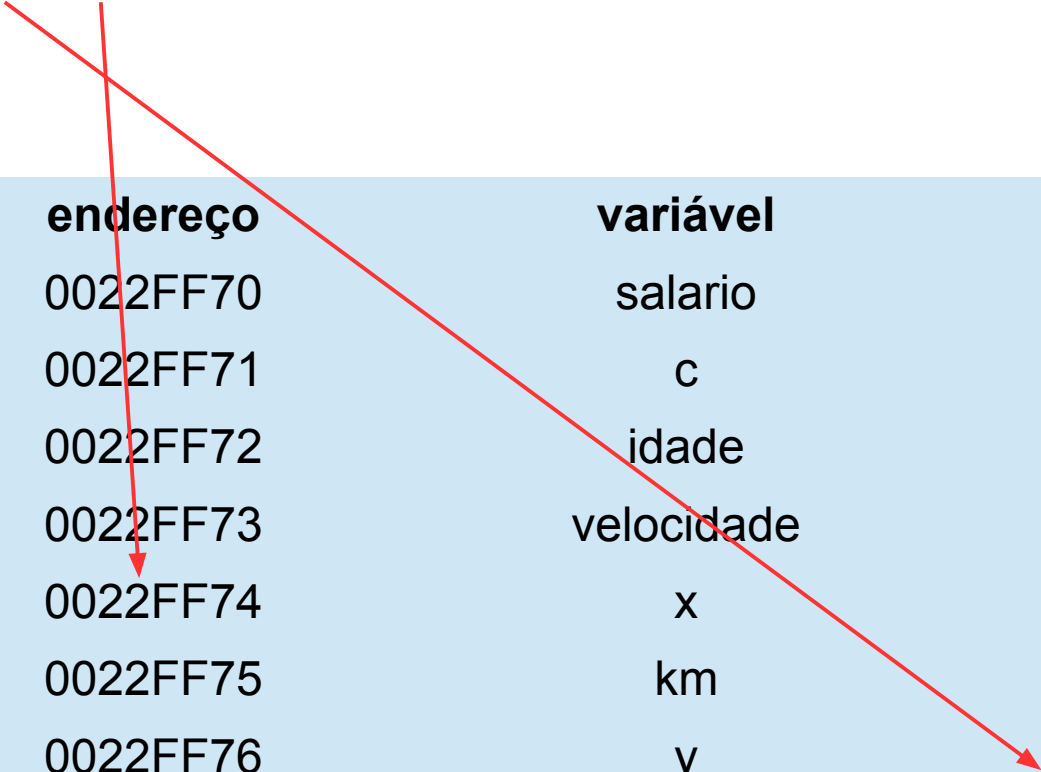
Endereços de variáveis

```
int x = 3;  
int *y;  
y = &x; //y recebe o endereço de x
```

endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74

Endereços de variáveis

```
int x = 3;  
int *y;  
y = &x; //y recebe o endereço de x
```



endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74

Endereços de variáveis

```
int x = 3;  
int *y;  
y = &x; //y recebe o endereço de x  
printf("conteudo de y: %d", *y);  
//*y = conteudo do endereco armazenado em y
```

endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74

Endereços de variáveis

```
int x = 3;
int *y;
y = &x; //y recebe o endereço de x
printf("conteudo de y: %d", *y);
//*y = conteudo do endereco armazenado em y
```

endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74

y

*y

ou

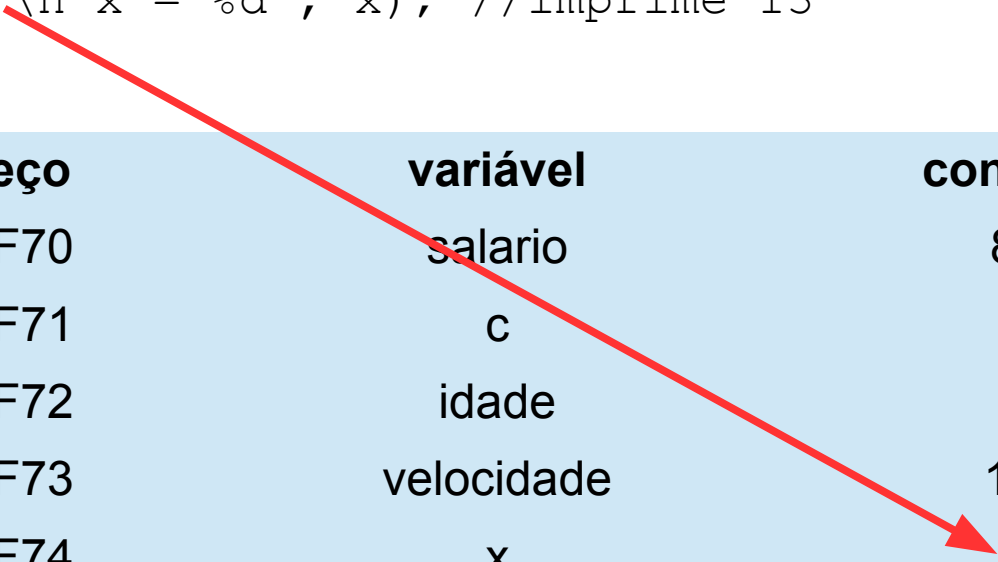
*(0022FF74)

Endereços de variáveis

```
int x = 3;
int *y;
y = &x; //y recebe o endereço de x
printf("conteudo de y: %d", *y);
//*y = conteudo do endereco armazenado em y
//usa-se tambem para alterar a variavel apontada
*y = *y + 10;
printf("\n x = %d", x); //imprime 13
```

Endereços de variáveis

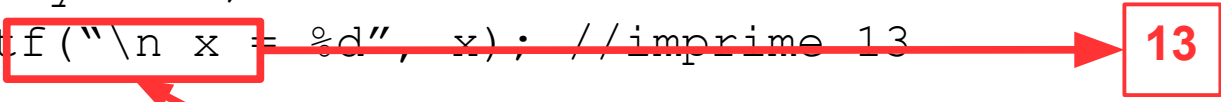
```
int x = 3;
int *y;
y = &x; //y recebe o endereço de x
printf("conteudo de y: %d", *y);
//*y = conteudo do endereco armazenado em y
//usa-se tambem para alterar a variavel apontada
*y = *y + 10;
printf("\n x = %d", x); //imprime 13
```



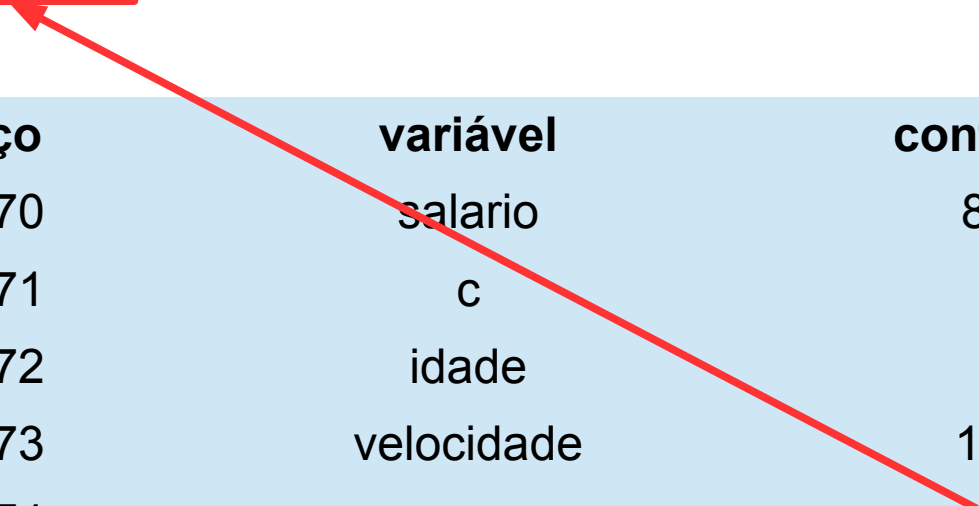
endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74

Endereços de variáveis

```
int x = 3;
int *y;
y = &x; //y recebe o endereço de x
printf("conteudo de y: %d", *y);
//*y = conteudo do endereco armazenado em y
//usa-se tambem para alterar a variavel apontada
*y = *y + 10;
printf("\n x = %d", x); //imprime 13
```



endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74



Endereços de variáveis

```
int x = 3;
int *y;
y = &x; //y recebe o endereço de x
printf("conteudo de y: %d", *y);
//*y = conteudo do endereco armazenado em y
//usa-se tambem para alterar a variavel apontada
*y = *y + 10;
printf("\n x = %d", x); //imprime 13
```



endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	3
0022FF75	km	298347
0022FF76	y	0022FF74

Endereços de variáveis


```
int x = 3;
int *y;
y = &x; //y recebe o endereço de x
printf("conteudo de y: %d", *y);
//*y = conteudo do endereco armazenado em y
//usa-se tambem para alterar a variavel apontada
*y = *y + 10;
printf("\n x = %d", x); //imprime 13
```



endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	13
0022FF75	km	298347
0022FF76	y	0022FF74

Endereços de variáveis

```
int x = 3;
int *y;
y = &x; //y recebe o endereço de x
printf("conteudo de y: %d", *y);
//*y = conteudo do endereco armazenado em y
//usa-se tambem para alterar a variavel apontada
*y = *y + 10;
printf("\n x = %d", x); //imprime 13
```



endereço	variável	conteúdo
0022FF70	salario	891
0022FF71	c	'a'
0022FF72	idade	8
0022FF73	velocidade	16.1
0022FF74	x	13
0022FF75	km	298347
0022FF76	y	0022FF74

Endereços de variáveis

```
#include <stdio.h>

void main() {
    int x = 20;
    int *y = &x;
    printf("Conteudo de x: %d\n", x);
    printf("Endereco de x: %p\n", &x);
    printf("Conteudo de y: %p\n", y);
    printf("Endereco de y: %p\n", &y);
    printf("Conteudo do endereco apontado por y: %d\n", *y);
    system("PAUSE");
}
```

Endereços de variáveis

```
#include <stdio.h>

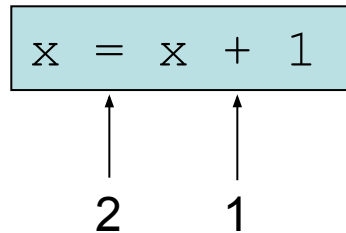
void main() {
    int x = 20;
    int *y = &x;
    printf("Conteudo de x: %d\n", x);
    printf("Endereco de x: %p\n", &x);
    printf("Conteudo de y: %p\n", y);
    printf("Endereco de y: %p\n", &y);
    printf("Conteudo do endereco apontado por y: %d\n", *y);
    system("PAUSE");
}
```

C:\Windows\system32\cmd.exe - a.exe

```
C:\Users\Pedro\Desktop\gcc-test>a.exe
Conteudo de x: 20
Endereco de x: 0028FF1C
Conteudo de y: 0028FF1C
Endereco de y: 0028FF18
Conteudo do endereco apontado por y: 20
Press any key to continue . . .
```

Operadores de incremento e decremento

- Uma operação muito comum em programas de computador é **incrementar de 1** o valor da variável.
- Para fazer isso devemos:
 1. Somar 1 ao valor atual da variável;
 2. Armazenar o resultado na própria variável.



- Como a operação **incremento de 1** é muito comum, em C tem-se um operador especial: **`++`**.

Operadores de incremento e decremento

- Ao invés de escrevermos $x = x + 1$, podemos escrever: $x++$
- Da mesma forma, para a operação decremento de 1: Em vez de $x = x - 1$, podemos escrever: $x--$
- Os operadores $++$ e $--$ podem ser usados como **prefixo** ou como **sufixo** do nome da variável

<pre>int a = 5, b = 3; int c;</pre>			
<pre>c = a++ + b;</pre>	→	a = 6 b = 3 c = 8	
<pre>c = ++a + b;</pre>	→	a = 7 b = 3 c = 10	


Operações combinadas com a atribuição

- As operações de **incremento** (++) e **decremento** (--) são exemplos de operações combinadas com a atribuição.
- Na linguagem C, sempre que for necessário escrever uma operação de atribuição da forma:

```
variavel = variavel operador expressao;
```

podemos combinar as operações:

Exemplos:

<code>x = x + 5;</code>		<code>x += 5;</code>
<code>x = x - (a + b);</code>		<code>x -= (a + b);</code>
<code>x = x * (a - b);</code>		<code>x *= (a - b);</code>
<code>x = x / (x + 1);</code>		<code>x /= (x + 1);</code>

Operações bit-a-bit

- Tabela-verdade para cada operador.

and (&)

	y	0	1
x			
0	0	0	0
1	0	0	1

or (|)

	y	0	1
x			
0	0	0	1
1	1	1	1

xor (^)

	y	0	1
x			
0	0	0	1
1	1	1	0

Operações bit-a-bit

- Vou a praia...
 - x: se for fim de semana
 - y: se fizer sol

Operações bit-a-bit

- Vou a praia...
 - x: se for fim de semana
 - y: se fizer sol
- Codificação
 - 1: sim
 - 0: não

or (|)

	y	0	1
x			
0		0	1
1		1	1

and (&)

	y	0	1
x			
0		0	0
1		0	1

xor (^)

	y	0	1
x			
0		0	1
1		1	0

Operações bit-a-bit

- Por uma questão de eficiência, a linguagem C dispõe de operações que podem ser feitas sobre a representação binária dos números inteiros.

Operador	Operação
< <	deslocamento para a esquerda
> >	deslocamento para a direita
&	conjunção bit-a-bit (<i>and</i>)
	disjunção bit-a-bit (<i>or</i>)
^	disjunção exclusiva bit-a-bit (<i>xor</i>)
~	negação bit-a-bit (<i>inverso</i>)

Operações bit-a-bit

Hexadecimal	Binário
0FF0	0000 1111 1111 0000
FF00	1111 1111 0000 0000
0FF0 << 4	1111 1111 0000 0000 = FF00
0FF0 >> 4	0000 0000 1111 1111 = 00FF
0FF0 & FF00	0000 1111 1111 0000 1111 1111 0000 0000 ----- 0000 1111 0000 0000 = 0F00
0FF0 FF00	0000 1111 1111 0000 1111 1111 0000 0000 ----- 1111 1111 1111 0000 = FFF0
0FF0 ^ FF00	0000 1111 1111 0000 1111 1111 0000 0000 ----- 1111 0000 1111 0000 = F0F0
~ 0FF0	0000 1111 1111 0000 ----- 1111 0000 0000 1111 = F00F

Operações bit-a-bit

- Exemplos:

```
int a = 0xFF0;  
int b = 0xFF00;  
int c;  
  
c = a << 4; printf("%04X << 4 = %04X\n", a, c);  
c = a >> 4; printf("%04X >> 4 = %04X\n", a, c);  
c = a & b;  printf("%04X & %04X = %04X\n", a, b, c);
```

Serão exibidos:

```
0FF0 << 4 = FF00  
0FF0 >> 4 = 00FF  
0FF0 & FF00 = 0F00
```