

# Unidade II - Tipos, Operadores e Expressões

Disciplina Linguagens de Programação I  
Bacharelado em Ciência da Computação da Uerj  
Professores Guilherme Abelha & Gilson Costa

## ANSI C

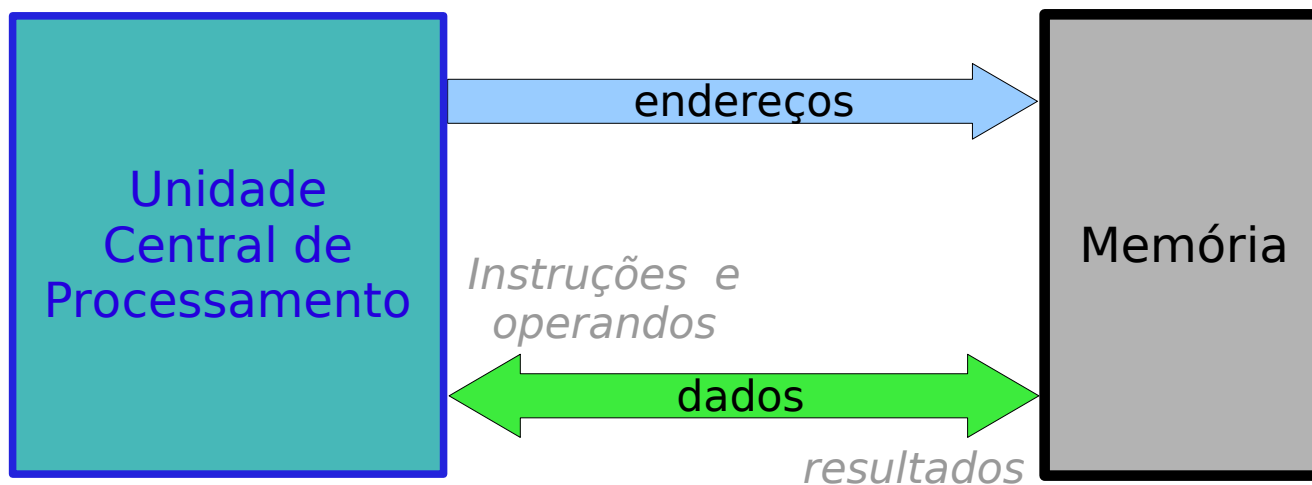
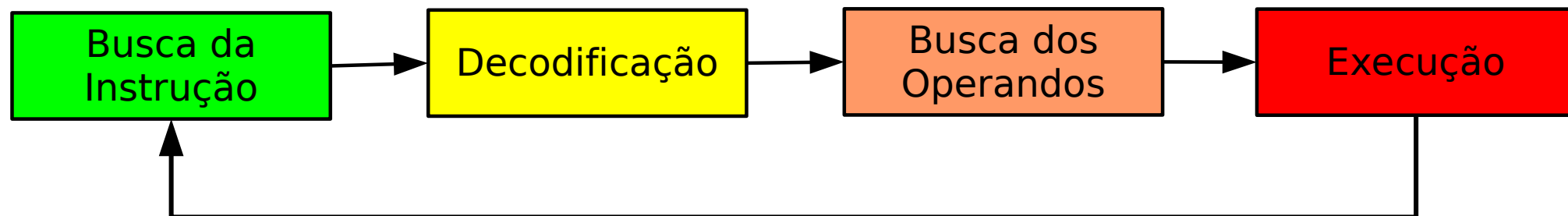
```
#include <stdio.h>
int main ()
{
    printf("Hello World!");
    return 0;
}
```

# Que assuntos serão abordados nesta unidade?

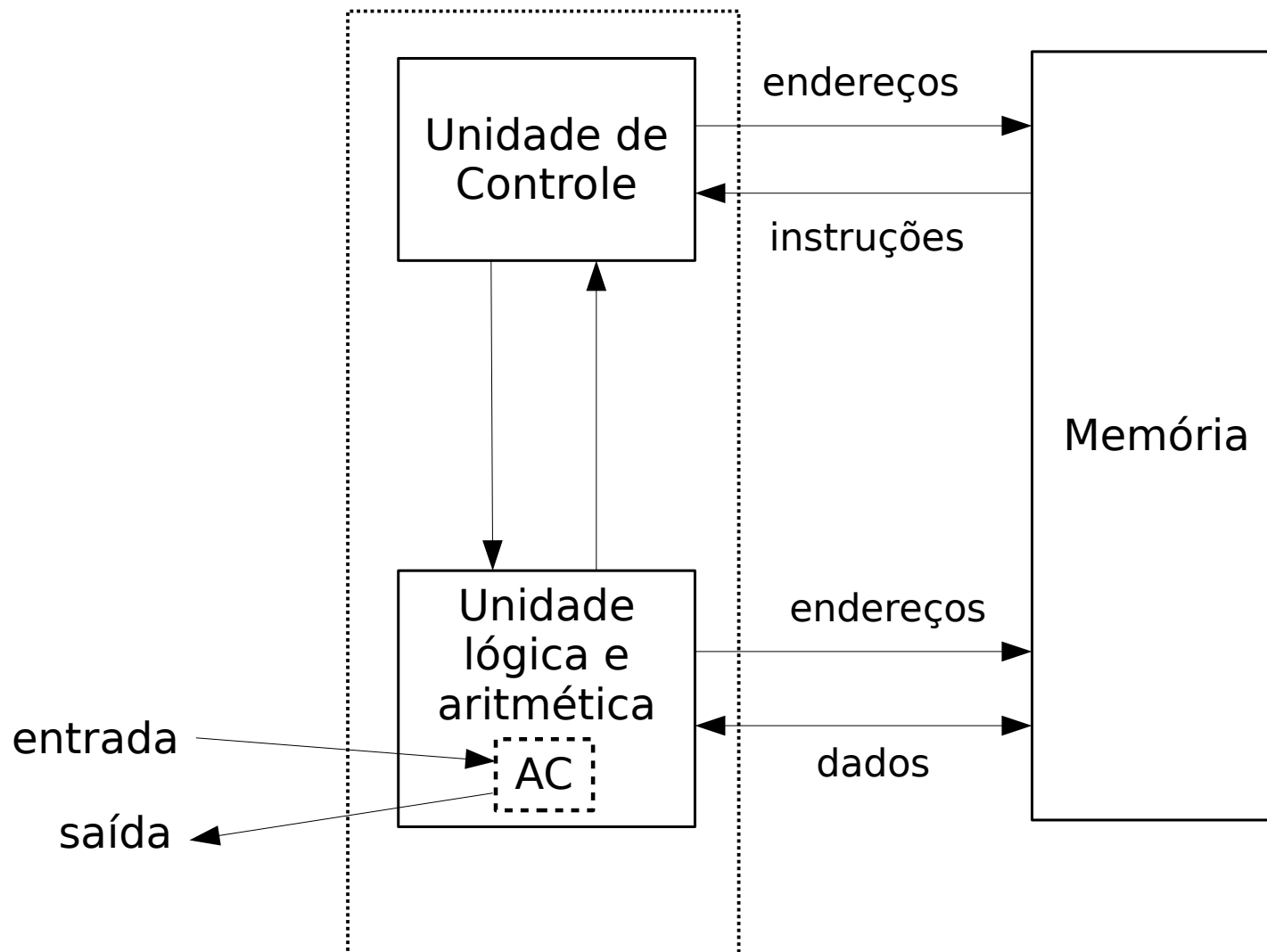
- Organização de Computadores
  - visão geral
  - modelo em níveis
  - subsistema de memória
  - uso dos sistemas de tipos
- Operadores e Expressões no ANSI C
  - operadores
  - precedência de operadores
  - ordem de avaliação
  - conversão de tipos
  - operações com strings
- Sistema de tipos do ANSI C
  - tipos primários
  - declaração de constantes
  - tipos derivados
  - tipos definidos no programa
  - declaração de variáveis
  - alocação de memória
    - ♦ alocação automática
    - ♦ alocação dinâmica
  - variáveis em registradores

# **Estrutura básica de um computador**

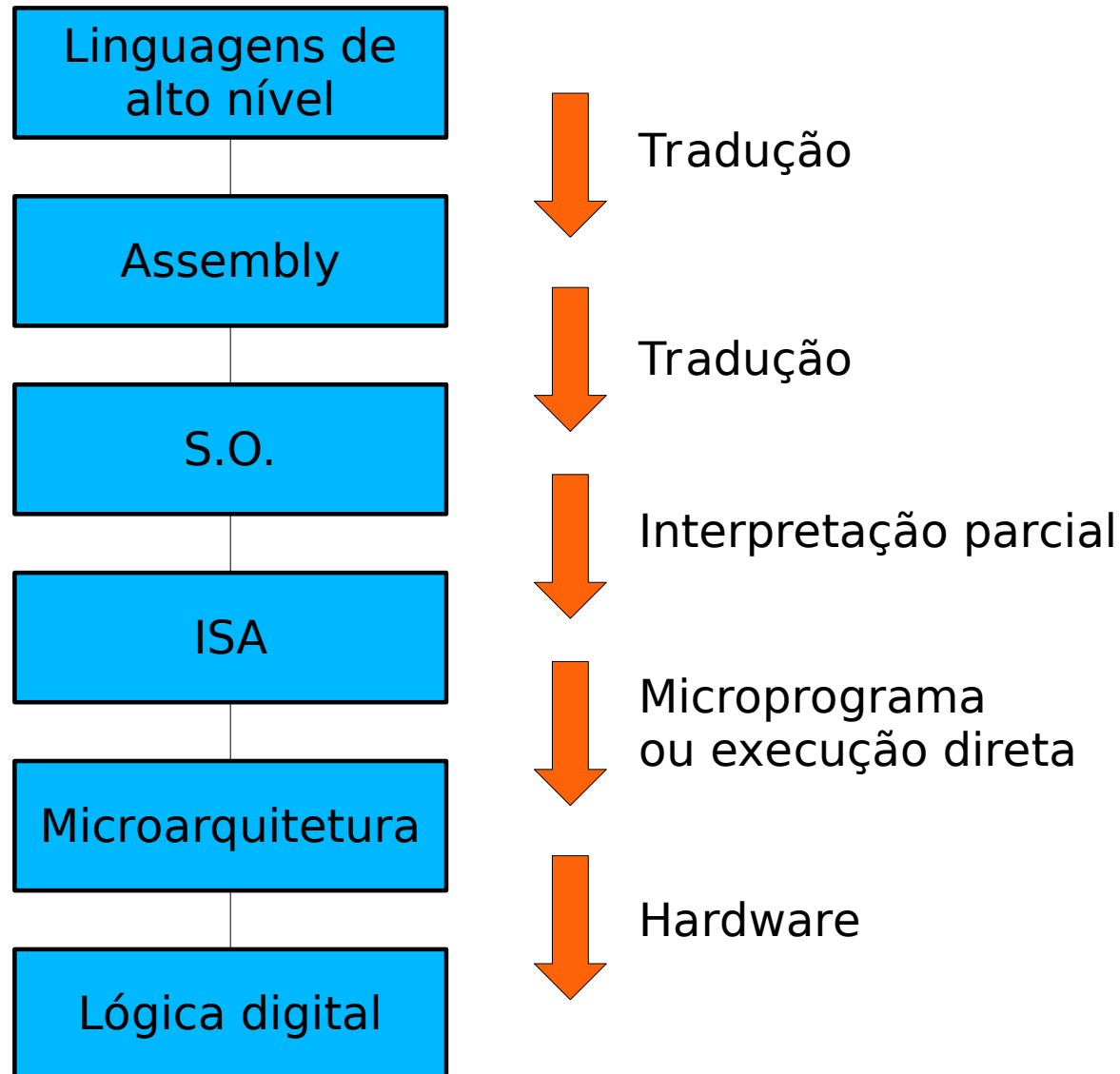
# Ciclo de Instrução



# A Máquina Original de Von Neuman



# Arquitetura de Computadores em Níveis



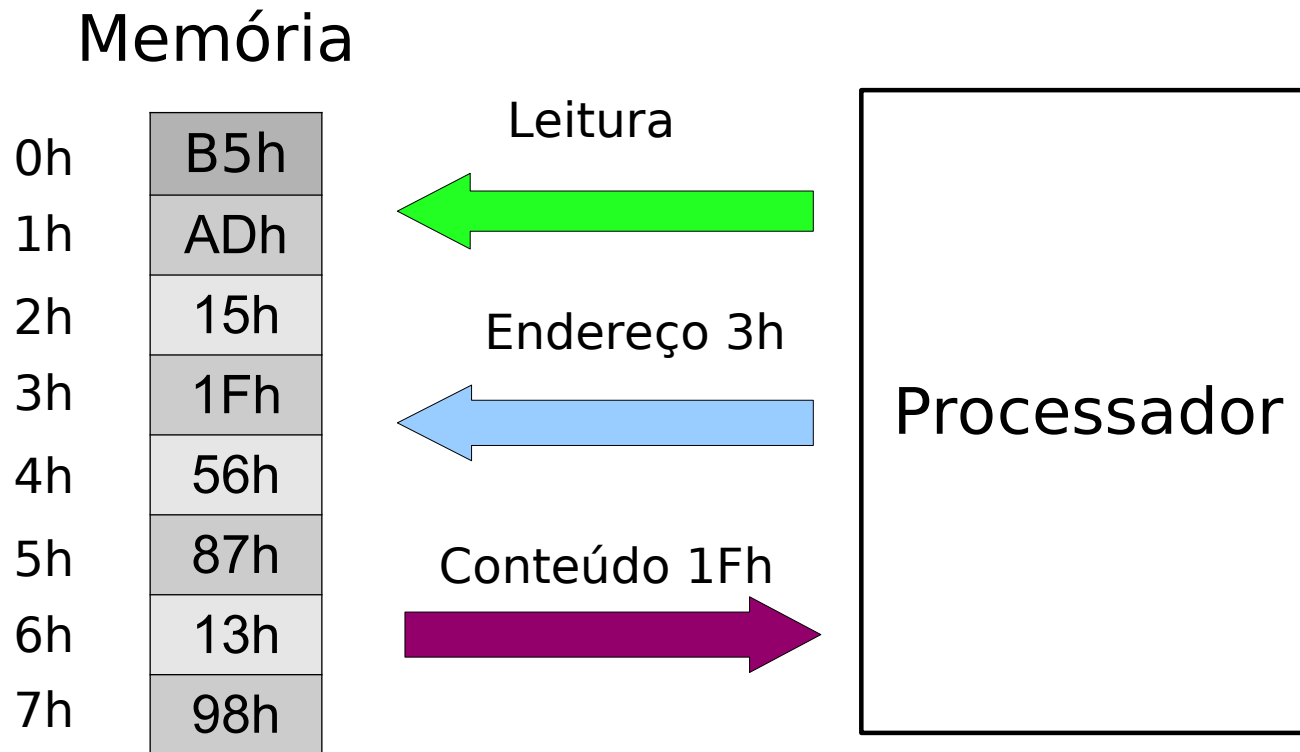
# Memória de computador

# Dispositivos de Memória

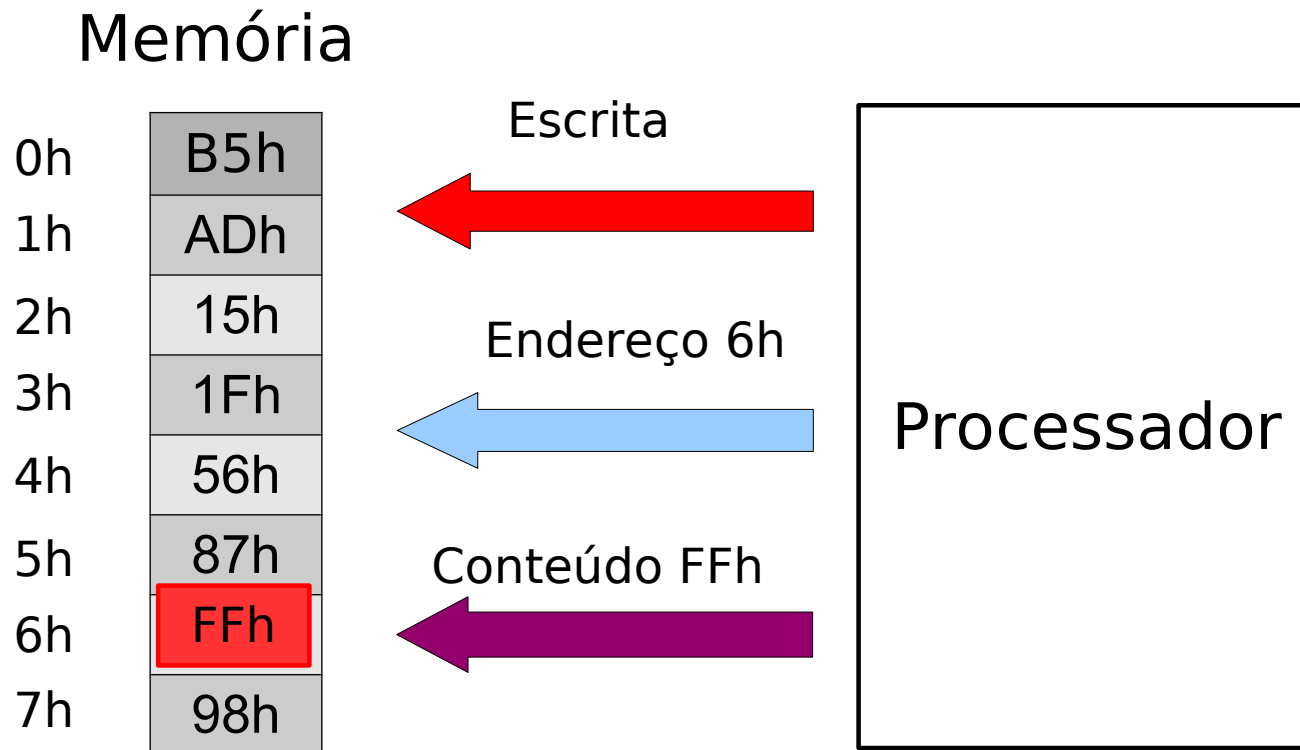




# Leitura da Memória

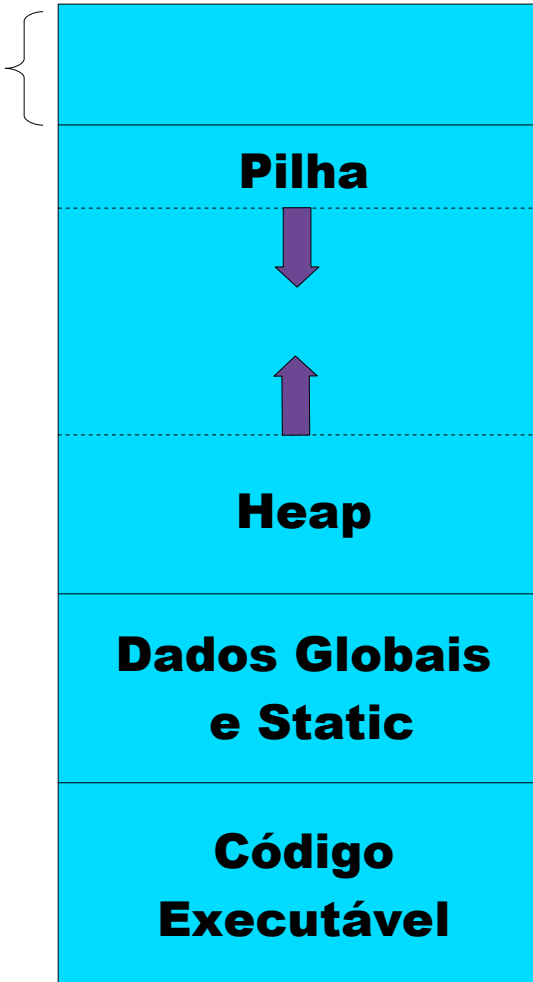


# Escrita na Memória



# Mapa de Memória de um Programa C em Execução

**Argumentos de linha de comando  
e variáveis de ambiente**



# Sistema de tipos

# Funções dos Sistemas de Tipos

- Armazenamento de dados em memória
  - Codificação
  - Decodificação
- Execução de expressões
  - Precedência
  - Ordem de avaliação
- Conversão de valores
- Validação de tipos

# Sistema de Tipos ANSI C

- Tipos primários
  - `char`, `short`, `int`,  
`long`, `long long`,  
(signed / unsigned)
  - `float`, `double`, `long double`
- Tipos definidos pelo usuário
  - `enum`, `typedef`
- Tipos derivados
  - `Tipo []`, `Tipo [][]`,  
`Tipo *`, `Tipo**`,  
`struct {}`, `union`

# **Tipos primários do ANSI C**

# Tipos Primários do Ansi C - Família dos Inteiros

| Tipo      | Tam. (B) | Faixa de valores <i>signed</i>                          | Faixa de valores <i>unsigned</i> |
|-----------|----------|---|----------------------------------|
| char      | 1        | [-128, 127]   | [0, 255]                         |
| short     | 2        | [-32.768, 32.767]                                       | [0, 65.535]                      |
| int       | 4        | [-2.147.483.648, 2.147.483.647]                         | [0, 4.294.967.295]               |
| long      | 4        | [-2.147.483.648, 2.147.483.647]                         | [0, 4.294.967.295]               |
| long long | 8        | [-9.223.372.036.854.775.808, 9.223.372.036.854.775.807] | [0, 18.446.744.073.709.551.615]  |



# Tipos Primários do Ansi C - Ponto Flutuante

- A codificação dos números reais se assemelha à notação científica. Ex.:  $678 \rightarrow 6,78 \times 10^2$



$$Valor = (-1)^S \cdot M \cdot 2^{E - bias}$$

| Tipo        | Tam. (B) | Faixa de valores*                               |
|-------------|----------|---|
| float       | 4        | $[3,4 \times 10^{-38}, 3,4 \times 10^{38}]$     |
| double      | 8        | $[1,7 \times 10^{-308}, 1,7 \times 10^{308}]$   |
| long double | 10       | $[3,4 \times 10^{-4932}, 3,4 \times 10^{4932}]$ |

\* fonte: "Programming in Ansi C" E. Balagurusamy

# Tipos Primários do Ansi C - Ponto Flutuante

- Exemplo padrão IEEE 754-1985 de 32 bits:

|   |          |                          |
|---|----------|--------------------------|
| S | EXPOENTE | MANTISSA                 |
| 0 | 01111100 | 010000000000000000000000 |

$$Valor = (-1)^S \cdot M \cdot 2^{E - bias}$$

$$Valor = (-1)^0 \cdot 1,0100 \cdot 2^{124 - 127}$$

$$Valor = 1,0100 \cdot 2^{-3}$$

$$Valor = (0,0010100)_2$$

$$Valor = 2^{-3} + 2^{-5} = 1/8 + 1/32$$

$$Valor = 0,125 + 0,0315 = 0,15625$$

# Definição de constantes

# Constantes

```
'A' 'c' /* (char) */  
"Cadeia de char" "" /* (char[]) */  
100 /* (int) */  
100L /* (long) */  
100.0 /* (float) */  
123.7e-2 123.7E-2 /* (double) */  
023 067L /* (int) notação octal */  
0xFD 0XFD 0xdaefeeL /* (int) notação hex */  
'\n' '\t' '\b' '\0' '\'' '\\' '\014' /* (char) */
```

# Código ASCII

| Dec | Hx | Oct | Char                               | Dec | Hx | Oct | Html  | Chr          | Dec | Hx | Oct | Html  | Chr      | Dec | Hx | Oct | Html   | Chr        |
|-----|----|-----|------------------------------------|-----|----|-----|-------|--------------|-----|----|-----|-------|----------|-----|----|-----|--------|------------|
| 0   | 0  | 000 | <b>NUL</b> (null)                  | 32  | 20 | 040 | &#32; | <b>Space</b> | 64  | 40 | 100 | &#64; | <b>@</b> | 96  | 60 | 140 | &#96;  | <b>`</b>   |
| 1   | 1  | 001 | <b>SOH</b> (start of heading)      | 33  | 21 | 041 | &#33; | <b>!</b>     | 65  | 41 | 101 | &#65; | <b>A</b> | 97  | 61 | 141 | &#97;  | <b>a</b>   |
| 2   | 2  | 002 | <b>STX</b> (start of text)         | 34  | 22 | 042 | &#34; | <b>"</b>     | 66  | 42 | 102 | &#66; | <b>B</b> | 98  | 62 | 142 | &#98;  | <b>b</b>   |
| 3   | 3  | 003 | <b>ETX</b> (end of text)           | 35  | 23 | 043 | &#35; | <b>#</b>     | 67  | 43 | 103 | &#67; | <b>C</b> | 99  | 63 | 143 | &#99;  | <b>c</b>   |
| 4   | 4  | 004 | <b>EOT</b> (end of transmission)   | 36  | 24 | 044 | &#36; | <b>\$</b>    | 68  | 44 | 104 | &#68; | <b>D</b> | 100 | 64 | 144 | &#100; | <b>d</b>   |
| 5   | 5  | 005 | <b>ENQ</b> (enquiry)               | 37  | 25 | 045 | &#37; | <b>%</b>     | 69  | 45 | 105 | &#69; | <b>E</b> | 101 | 65 | 145 | &#101; | <b>e</b>   |
| 6   | 6  | 006 | <b>ACK</b> (acknowledge)           | 38  | 26 | 046 | &#38; | <b>&amp;</b> | 70  | 46 | 106 | &#70; | <b>F</b> | 102 | 66 | 146 | &#102; | <b>f</b>   |
| 7   | 7  | 007 | <b>BEL</b> (bell)                  | 39  | 27 | 047 | &#39; | <b>'</b>     | 71  | 47 | 107 | &#71; | <b>G</b> | 103 | 67 | 147 | &#103; | <b>g</b>   |
| 8   | 8  | 010 | <b>BS</b> (backspace)              | 40  | 28 | 050 | &#40; | <b>(</b>     | 72  | 48 | 110 | &#72; | <b>H</b> | 104 | 68 | 150 | &#104; | <b>h</b>   |
| 9   | 9  | 011 | <b>TAB</b> (horizontal tab)        | 41  | 29 | 051 | &#41; | <b>)</b>     | 73  | 49 | 111 | &#73; | <b>I</b> | 105 | 69 | 151 | &#105; | <b>i</b>   |
| 10  | A  | 012 | <b>LF</b> (NL line feed, new line) | 42  | 2A | 052 | &#42; | <b>*</b>     | 74  | 4A | 112 | &#74; | <b>J</b> | 106 | 6A | 152 | &#106; | <b>j</b>   |
| 11  | B  | 013 | <b>VT</b> (vertical tab)           | 43  | 2B | 053 | &#43; | <b>+</b>     | 75  | 4B | 113 | &#75; | <b>K</b> | 107 | 6B | 153 | &#107; | <b>k</b>   |
| 12  | C  | 014 | <b>FF</b> (NP form feed, new page) | 44  | 2C | 054 | &#44; | <b>,</b>     | 76  | 4C | 114 | &#76; | <b>L</b> | 108 | 6C | 154 | &#108; | <b>l</b>   |
| 13  | D  | 015 | <b>CR</b> (carriage return)        | 45  | 2D | 055 | &#45; | <b>-</b>     | 77  | 4D | 115 | &#77; | <b>M</b> | 109 | 6D | 155 | &#109; | <b>m</b>   |
| 14  | E  | 016 | <b>SO</b> (shift out)              | 46  | 2E | 056 | &#46; | <b>.</b>     | 78  | 4E | 116 | &#78; | <b>N</b> | 110 | 6E | 156 | &#110; | <b>n</b>   |
| 15  | F  | 017 | <b>SI</b> (shift in)               | 47  | 2F | 057 | &#47; | <b>/</b>     | 79  | 4F | 117 | &#79; | <b>O</b> | 111 | 6F | 157 | &#111; | <b>o</b>   |
| 16  | 10 | 020 | <b>DLE</b> (data link escape)      | 48  | 30 | 060 | &#48; | <b>0</b>     | 80  | 50 | 120 | &#80; | <b>P</b> | 112 | 70 | 160 | &#112; | <b>p</b>   |
| 17  | 11 | 021 | <b>DC1</b> (device control 1)      | 49  | 31 | 061 | &#49; | <b>1</b>     | 81  | 51 | 121 | &#81; | <b>Q</b> | 113 | 71 | 161 | &#113; | <b>q</b>   |
| 18  | 12 | 022 | <b>DC2</b> (device control 2)      | 50  | 32 | 062 | &#50; | <b>2</b>     | 82  | 52 | 122 | &#82; | <b>R</b> | 114 | 72 | 162 | &#114; | <b>r</b>   |
| 19  | 13 | 023 | <b>DC3</b> (device control 3)      | 51  | 33 | 063 | &#51; | <b>3</b>     | 83  | 53 | 123 | &#83; | <b>S</b> | 115 | 73 | 163 | &#115; | <b>s</b>   |
| 20  | 14 | 024 | <b>DC4</b> (device control 4)      | 52  | 34 | 064 | &#52; | <b>4</b>     | 84  | 54 | 124 | &#84; | <b>T</b> | 116 | 74 | 164 | &#116; | <b>t</b>   |
| 21  | 15 | 025 | <b>NAK</b> (negative acknowledge)  | 53  | 35 | 065 | &#53; | <b>5</b>     | 85  | 55 | 125 | &#85; | <b>U</b> | 117 | 75 | 165 | &#117; | <b>u</b>   |
| 22  | 16 | 026 | <b>SYN</b> (synchronous idle)      | 54  | 36 | 066 | &#54; | <b>6</b>     | 86  | 56 | 126 | &#86; | <b>V</b> | 118 | 76 | 166 | &#118; | <b>v</b>   |
| 23  | 17 | 027 | <b>ETB</b> (end of trans. block)   | 55  | 37 | 067 | &#55; | <b>7</b>     | 87  | 57 | 127 | &#87; | <b>W</b> | 119 | 77 | 167 | &#119; | <b>w</b>   |
| 24  | 18 | 030 | <b>CAN</b> (cancel)                | 56  | 38 | 070 | &#56; | <b>8</b>     | 88  | 58 | 130 | &#88; | <b>X</b> | 120 | 78 | 170 | &#120; | <b>x</b>   |
| 25  | 19 | 031 | <b>EM</b> (end of medium)          | 57  | 39 | 071 | &#57; | <b>9</b>     | 89  | 59 | 131 | &#89; | <b>Y</b> | 121 | 79 | 171 | &#121; | <b>y</b>   |
| 26  | 1A | 032 | <b>SUB</b> (substitute)            | 58  | 3A | 072 | &#58; | <b>:</b>     | 90  | 5A | 132 | &#90; | <b>Z</b> | 122 | 7A | 172 | &#122; | <b>z</b>   |
| 27  | 1B | 033 | <b>ESC</b> (escape)                | 59  | 3B | 073 | &#59; | <b>;</b>     | 91  | 5B | 133 | &#91; | <b>[</b> | 123 | 7B | 173 | &#123; | <b>{</b>   |
| 28  | 1C | 034 | <b>FS</b> (file separator)         | 60  | 3C | 074 | &#60; | <b>&lt;</b>  | 92  | 5C | 134 | &#92; | <b>\</b> | 124 | 7C | 174 | &#124; | <b> </b>   |
| 29  | 1D | 035 | <b>GS</b> (group separator)        | 61  | 3D | 075 | &#61; | <b>=</b>     | 93  | 5D | 135 | &#93; | <b>]</b> | 125 | 7D | 175 | &#125; | <b>}</b>   |
| 30  | 1E | 036 | <b>RS</b> (record separator)       | 62  | 3E | 076 | &#62; | <b>&gt;</b>  | 94  | 5E | 136 | &#94; | <b>^</b> | 126 | 7E | 176 | &#126; | <b>~</b>   |
| 31  | 1F | 037 | <b>US</b> (unit separator)         | 63  | 3F | 077 | &#63; | <b>?</b>     | 95  | 5F | 137 | &#95; | <b>_</b> | 127 | 7F | 177 | &#127; | <b>DEL</b> |

Source: [www.LookupTables.com](http://www.LookupTables.com)

# Código ASCII

|     |   |     |   |     |   |     |   |     |   |     |   |     |   |     |   |
|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|
| 128 | Ç | 144 | É | 160 | á | 176 | ░ | 192 | Ł | 208 | ⌌ | 224 | α | 240 | ≡ |
| 129 | ü | 145 | æ | 161 | í | 177 | ▒ | 193 | ⌐ | 209 | ⌑ | 225 | β | 241 | ± |
| 130 | é | 146 | Æ | 162 | ó | 178 | ▓ | 194 | ⌒ | 210 | ⌒ | 226 | Γ | 242 | ≥ |
| 131 | â | 147 | ô | 163 | ú | 179 |   | 195 | ⌓ | 211 | ⌌ | 227 | π | 243 | ≤ |
| 132 | ä | 148 | ö | 164 | ñ | 180 | ⌔ | 196 | — | 212 | ⌍ | 228 | Σ | 244 | ∫ |
| 133 | à | 149 | ò | 165 | Ñ | 181 | ⌕ | 197 | ⌕ | 213 | ⌎ | 229 | σ | 245 | ∫ |
| 134 | å | 150 | û | 166 | ª | 182 | ⌖ | 198 | ⌖ | 214 | ⌏ | 230 | μ | 246 | ÷ |
| 135 | ç | 151 | ù | 167 | º | 183 | ⌗ | 199 | ⌗ | 215 | ⌐ | 231 | τ | 247 | ≈ |
| 136 | ê | 152 | ÿ | 168 | ¿ | 184 | ⌘ | 200 | ⌘ | 216 | ⌑ | 232 | Φ | 248 | ° |
| 137 | ë | 153 | Ö | 169 | ⌑ | 185 | ⌙ | 201 | ⌙ | 217 | ⌒ | 233 | ⊗ | 249 | · |
| 138 | è | 154 | Ü | 170 | ⌒ | 186 | ⌚ | 202 | ⌚ | 218 | ⌓ | 234 | Ω | 250 | · |
| 139 | ì | 155 | • | 171 | ½ | 187 | ⌛ | 203 | ⌛ | 219 | ■ | 235 | δ | 251 | √ |
| 140 | î | 156 | £ | 172 | ¼ | 188 | ⌜ | 204 | ⌜ | 220 | ■ | 236 | ∞ | 252 | ∞ |
| 141 | ï | 157 | ¥ | 173 | ¡ | 189 | ⌝ | 205 | = | 221 | ■ | 237 | φ | 253 | ² |
| 142 | Ä | 158 | £ | 174 | « | 190 | ⌞ | 206 | ⌞ | 222 | ■ | 238 | ε | 254 | ■ |
| 143 | Å | 159 | ƒ | 175 | » | 191 | ⌟ | 207 | ⌟ | 223 | ■ | 239 | ∩ | 255 |   |

Source: [www.LookupTables.com](http://www.LookupTables.com)

## Exercício U2.1 - Tipos primários no Linux 64

Crie um programa na linguagem C que apresente na tela os tamanhos dos tipos primários da linguagem (char, short, int, long, long long, float, double e long double).

# **Tipos derivados do ANSI C**



# Arrays

`float[]`

|      |      |      |      |       |
|------|------|------|------|-------|
| 0.27 | 2.45 | 6.12 | 5.07 | 0.145 |
|------|------|------|------|-------|

`int[][]`

|    |    |    |    |
|----|----|----|----|
| 10 | 20 | 15 | 18 |
| 44 | 65 | 2  | 5  |
| 16 | 25 | 6  | 36 |
| 23 | 15 | 56 | 36 |

# struct

- Cria tipos compostos

```
struct <identificadorDoTipo>
{
    <tipo1> <identificadorMembro1>;
    ...
    <tipon> <identificadorMembron>;
};
```

```
struct Point2D
{
    float x;
    float y;
    int label;
};
```

# struct

```
struct Point2D
{
    float x;
    float y;
    int label;
} Var1;

struct Point2D P1;

P1.x = 10.0;
P1.y = 20.0;
P1.label = 20;

Var1 = P1;
```

# union

- Cria tipos que pode assumir valores e se comportar como tipos diferentes
- Não é feito teste de consistência, o programa deve saber o tipo em uso

```
union u_tag{  
    int ival;  
    float fval;  
    char *sval;  
} u;  
  
printf("%d\n", u.ival);  
printf("%f\n", u.fval);  
printf("%s\n", u.sval);
```

# Apontadores

```
float* pfloat;
```

```
float** ppFloat;
```

# **Tipos definidos no programa**

# typedef

- Cria novos tipos
- Sinônimo para um tipo primitivo ou derivado

```
typedef <tipo> <identificadorDoTipo>;
```

```
typedef int Idade;  
typedef char* String;
```

# enum

- Cria um tipos que enumera constantes simbólicas do tipo inteiro
- Não é feito teste de consistência entre o valor inteiro armazenado e as constantes listadas

```
enum booleano { NO, YES }; /* NO = 0, YES = 1 */
```

```
enum escapes { BELL = '\a',  
    BACKSPACE = '\b', TAB = '\t',  
    NEWLINE = '\n', VTAB = '\v', RETURN='\r' };
```

```
enum months { JAN = 1, FEB, MAR, APR, MAY,  
    JUN, JUL, AUG, SEP, OCT, NOV, DEC }; /* FEB = 2,  
    MAR = 3, etc. */
```



## Exercício U2.2 - Tipos derivados no Linux 64

1. Baixe o arquivo Ud3Exercicio2.c
2. Analise o código
3. Compile e execute este programa
4. Compare o tamanho dos tipos derivados com o dos respectivos tipos primitivos
5. Avalie a coerência dos tamanhos dos tipos em função da teoria vista na aula

# Declaração de Variáveis

# Declaração de Variáveis

<tipo> <identificadorDaVariavel>;

```
int fahr;  
char c;  
unsigned int indice;  
long double saldoContaEikeBatista;  
short cont;  
float salarioDeProfessor;  
float vector[3];
```

# Declaração e Inicialização de Variáveis

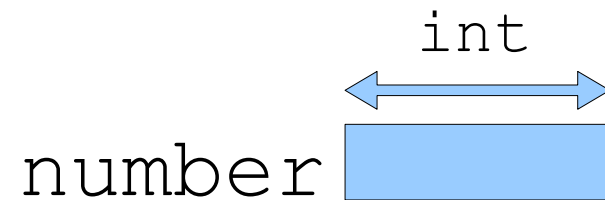
<tipo> <Variavel> = <constante>;

```
int fahr = 50;
char c = 50;
char c = '2';
char c = 0x32;
char c = 062;
unsigned int indice = 50;
long double saldoContaEikeBatista = 50.7e127;
short cont = 50;
float salarioDeProfessor = 50;
float vector[3] = {50, 12, 8};
char str1[] = "Sou uma cadeia de caracteres\n";
```

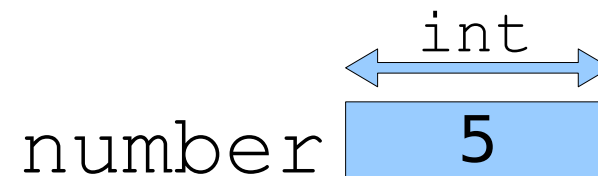
# **Alocação automática e dinâmica de memória**

# Alocação automática de memória

```
int number;
```



```
int number = 5;
```



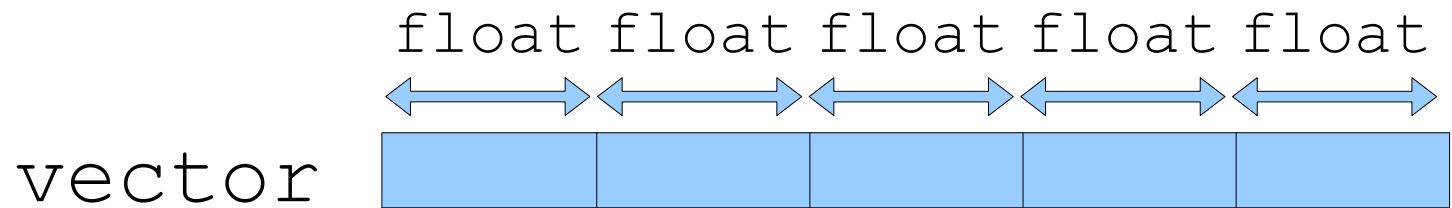
# Alocação automática de memória

```
int main()
{
    short Sh1;
    char Ch1;
    float F11;
    struct Point2d
    {
        float x,y;
    } P1;
}
```

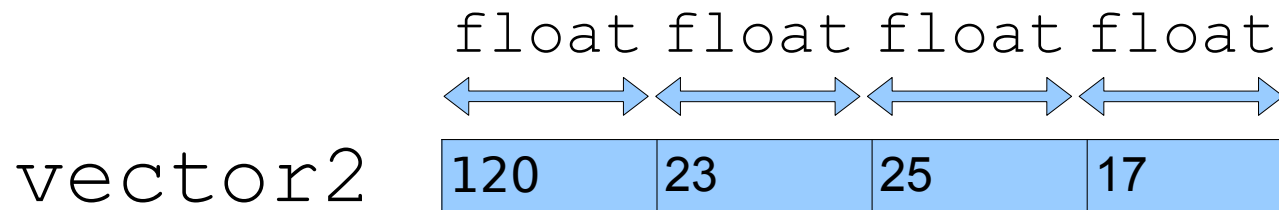
|                    |      |     |
|--------------------|------|-----|
| 0x0000000003CE10CB | 0xA5 | Sh1 |
| 0x0000000003CE10CC | 0xFD |     |
| 0x0000000003CE10CD | 0x01 | Ch1 |
| 0x0000000003CE10CE | 0xFF | F11 |
| 0x0000000003CE10CF | 0x96 |     |
| 0x0000000003CE10D0 | 0xB7 |     |
| 0x0000000003CE10D1 | 0x32 | P1  |
| 0x0000000003CE10D2 | 0x08 |     |
| 0x0000000003CE10D3 | 0xA5 |     |
| 0x0000000003CE10D4 | 0xFD |     |
| 0x0000000003CE10D5 | 0x01 |     |
| 0x0000000003CE10D6 | 0xFF |     |
| 0x0000000003CE10D7 | 0x96 |     |
| 0x0000000003CE10D8 | 0xB7 |     |
| 0x0000000003CE10D9 | 0x32 |     |
| 0x0000000003CE10DA | 0x08 |     |

# Alocação automática de memória

```
float vect[5];
```



```
float vect2[]={120.0, 23.0, 25.0, 17.0};
```





# Alocação automática de memória

```
int main()  
{  
    short v[2];  
    v;  
    v[0]=2;  
    v[1]=5;  
    * (v+1)=5;  
}
```

|                    |
|--------------------|
| 0x0000000003CE10CB |
| 0x0000000003CE10CC |
| 0x0000000003CE10CD |
| 0x0000000003CE10CE |
| 0x0000000003CE10CF |
| 0x0000000003CE10D0 |
| 0x0000000003CE10D1 |
| 0x0000000003CE10D2 |
| 0x0000000003CE10D3 |
| 0x0000000003CE10D4 |
| 0x0000000003CE10D5 |
| 0x0000000003CE10D6 |
| 0x0000000003CE10D7 |
| 0x0000000003CE10D8 |
| 0x0000000003CE10D9 |
| 0x0000000003CE10DA |

|      |
|------|
| 0xA5 |
| 0xFD |
| 0x01 |
| 0xFF |
| 0x96 |
| 0xB7 |
| 0x32 |
| 0x08 |
| 0xA5 |
| 0xFD |
| 0x01 |
| 0xFF |
| 0x96 |
| 0xB7 |
| 0x32 |
| 0x08 |

v[0]

v[1]

# Alocação automática de memória

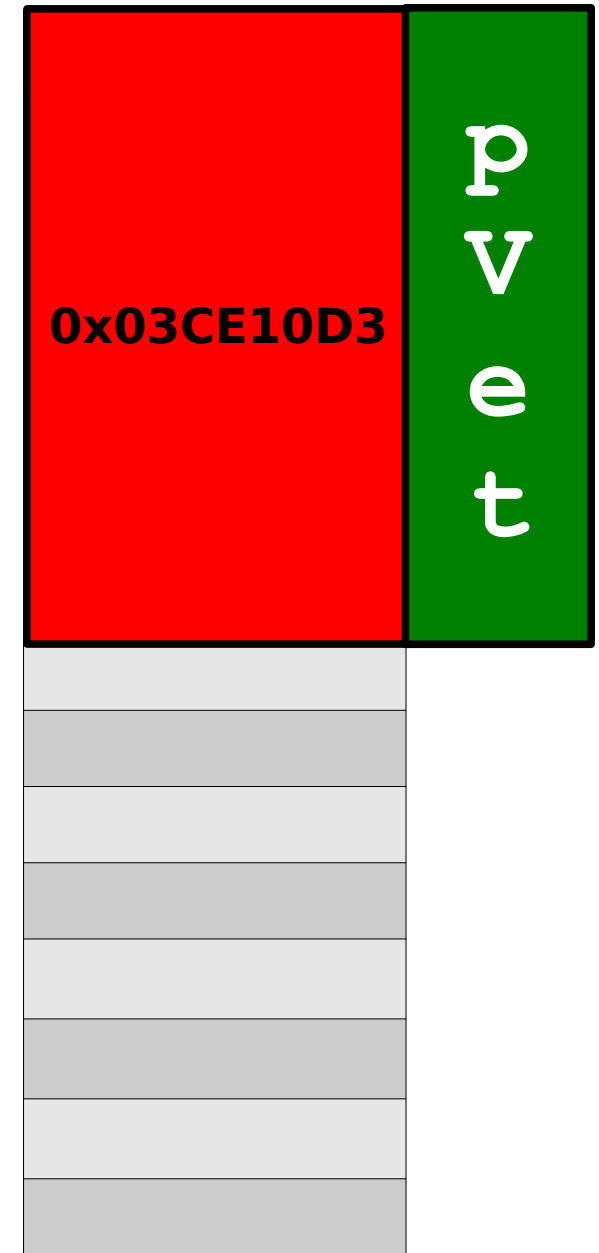
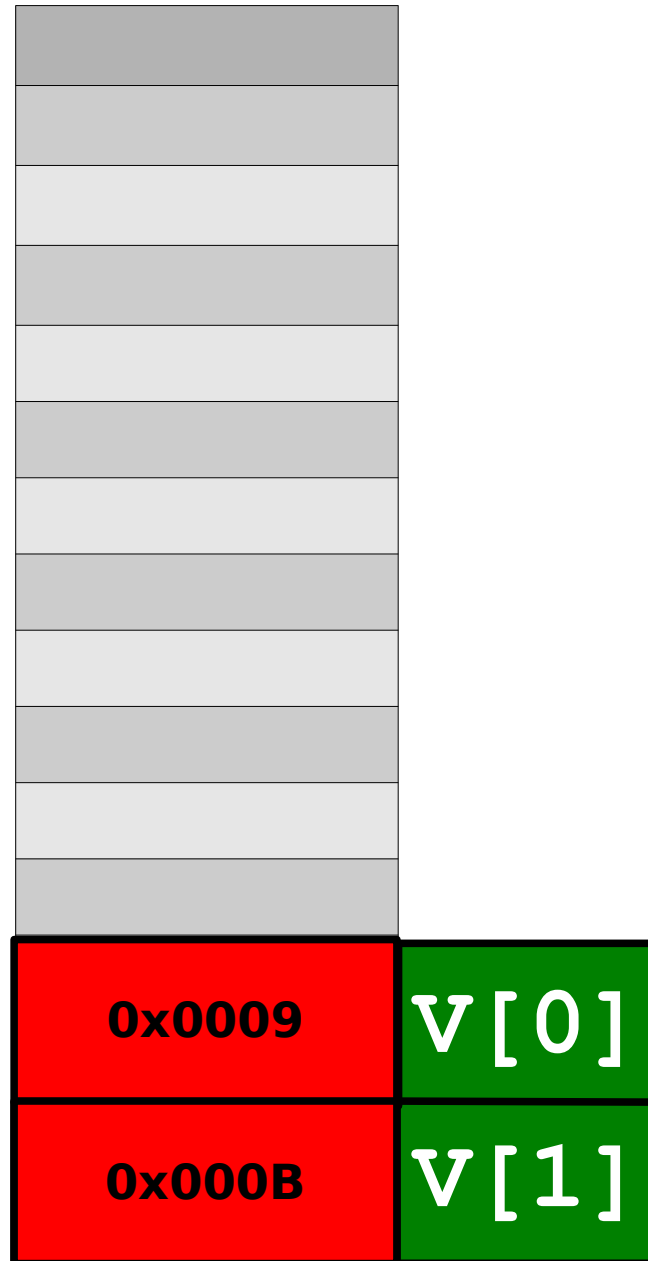
```
int main()  
{  
    char S[5];  
}
```

|                     |      |      |
|---------------------|------|------|
| 0x000000000001C10AB | 0xA5 |      |
| 0x000000000001C10AC | 0xFD |      |
| 0x000000000001C10AD | 0x01 |      |
| 0x000000000001C10AE | 0xFF |      |
| 0x000000000001C10AF | 0x96 |      |
| 0x000000000001C10B0 | 0xB7 |      |
| 0x000000000001C10B1 | 0x32 |      |
| 0x000000000001C10B2 | 0x08 |      |
| 0x000000000001C10B3 | 0xA5 | S[0] |
| 0x000000000001C10B4 | 0xFD | S[1] |
| 0x000000000001C10B5 | 0x01 | S[2] |
| 0x000000000001C10B6 | 0xFF | S[3] |
| 0x000000000001C10B7 | 0x96 | S[4] |
| 0x000000000001C10B8 | 0xB7 |      |
| 0x000000000001C10B9 | 0x32 |      |
| 0x000000000001C10BA | 0x08 |      |

# Apontadores e Arrays

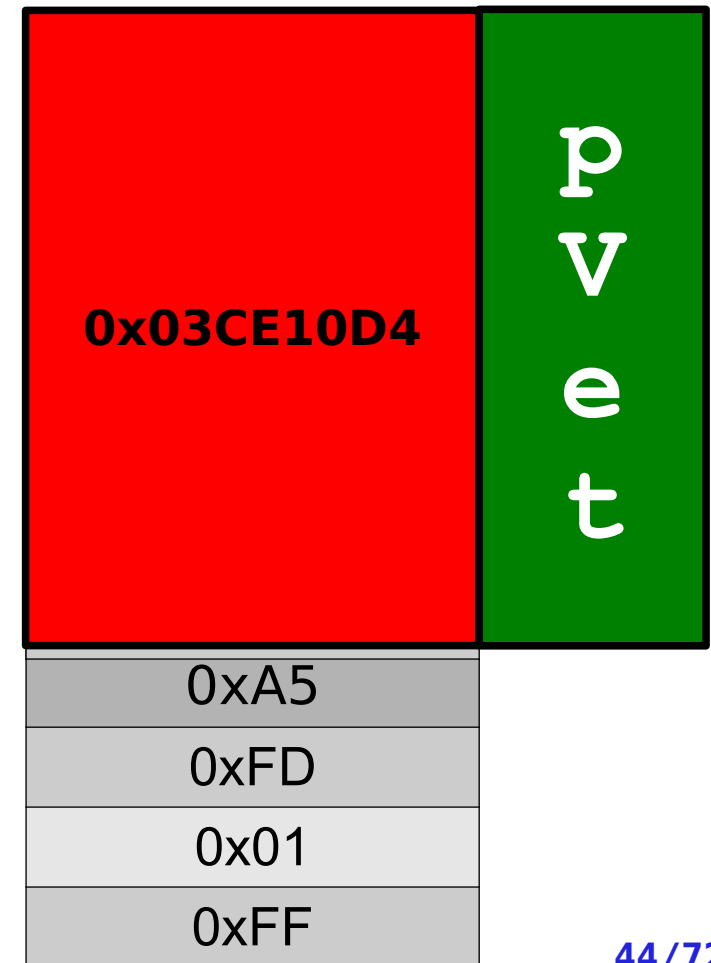
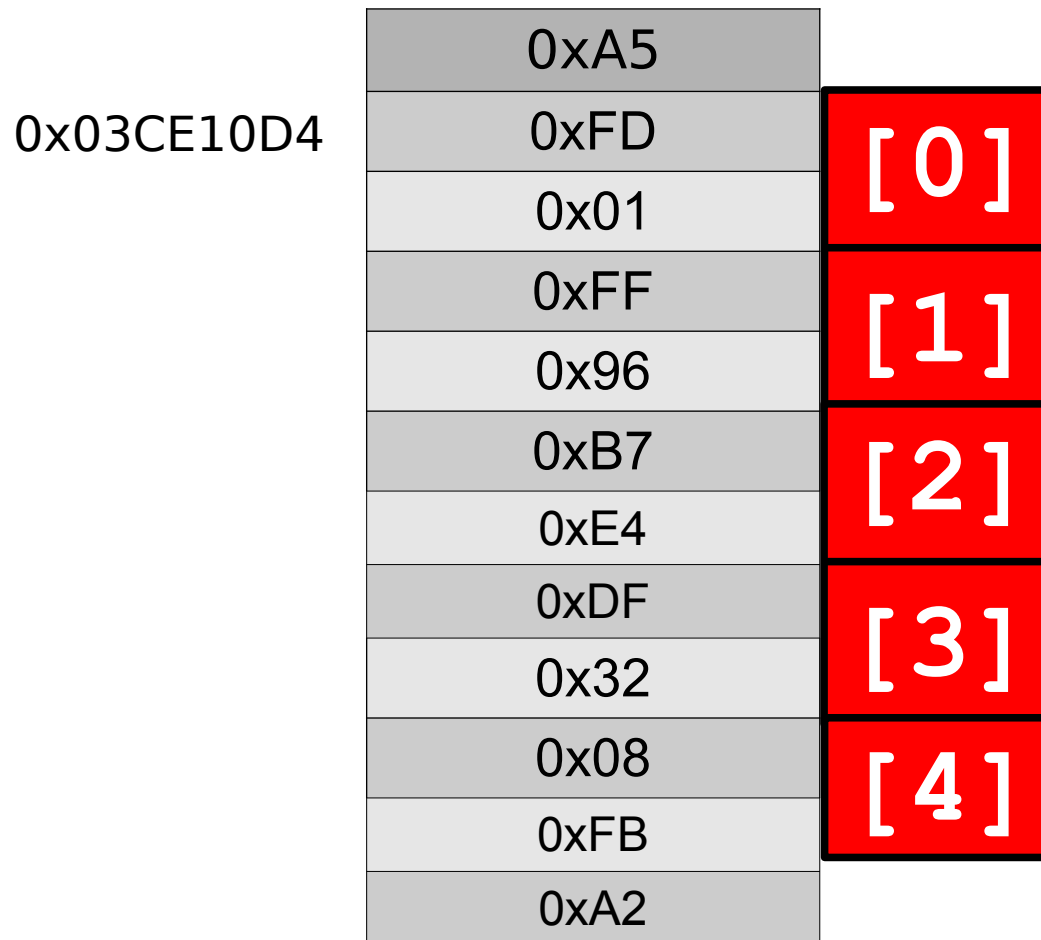
```
int main()  
{  
    short V[2];  
    short* pVet;  
    pVet = V;  
    *pVet=5;  
    pVet[0]=9;  
    pVet[1]=10;  
    *(pVet+1)=11;  
}
```

0x0000000003CE10D3



# Alocação Dinâmica de memória

```
int main()
{
    short* pVet;
    pVet = malloc(5*sizeof(*pVet));
}
```



# Tratamento de caracteres em C

# Funções da Biblioteca `ctype.h`

- Avaliação de caracteres:
  - `isalnum()` se é alfanumérico
  - `isalpha()` se é alfabético
  - `isblank()` se é branco
  - `iscntrl()` se is um caracter de controle
  - `isdigit()` se é um dígito decimal
  - `isgraph()` se possui representação gráfica
  - `islower()` se é letra minúscula
  - `isprint()` se é imprimível
  - `ispunct()` se caracter é símbolo de pontuação

# Funções da Biblioteca `ctype.h`

- Avaliação de caracteres:
  - `isspace()` se é maiúsculo espaço em branco
  - `isupper()` se é maiúsculo
  - `isxdigit()` se é um dígito hexadecimal válido
- Conversão de caracteres:
  - `tolower()` para minúsculo
  - `toupper()` para maiúsculo

# String em C



# String

```
char Str[5]="Uerj";
```

```
char Str[ ]="Uerj";
```

0x000000000001C10B3

|             |
|-------------|
| 0xA5        |
| 0xFD        |
| 0x01        |
| 0xFF        |
| 0x96        |
| 0xB7        |
| 0x32        |
| 0x08        |
| <b>0x55</b> |
| <b>0x65</b> |
| <b>0x72</b> |
| <b>0x6A</b> |
| <b>0x00</b> |

|               |
|---------------|
| <b>Str[0]</b> |
| <b>Str[1]</b> |
| <b>Str[2]</b> |
| <b>Str[3]</b> |
| <b>Str[4]</b> |

# Funções da Biblioteca `string.h`

- Cópia:
  - `strcpy()` copia string
  - `strncpy()` copia caracteres de uma string
- Concatenation:
  - `strcat()` concatena strings
  - `strncat()` concatena parte de uma string a outra
- Comparison:
  - `strcmp()` compara duas strings caracter a caracter
  - `strcoll()` compara duas strings usando `LC_COLLATE`
  - `strncmp()` compara até  $n$  caracteres de duas strings
  - `strxfrm()` transforma string usando `LC_COLLATE`

# Funções da Biblioteca `string.h`

- Busca:
  - `strchr()` a primeira ocorrência de um caracter
  - `strcspn()` fornece o comprimento da substring inicial de `str1` contendo somente caracteres que não pertençam a `str2`
  - `strpbrk()` idem `strcspn()`, retornando ponteiro
  - `strrchr()` a última ocorrência de um caracter
  - `strspn()` fornece o comprimento da substring inicial de `str1` contendo exclusivamente caracteres em `str2`
  - `strstr()` localiza substring
  - `strtok()` quebra string em tokens
- Outras:
  - `strerror()` obtém apontador para mensagem de erro
  - `strlen()` obtém o comprimento da string

## Exercício U3.3 - Funções de Manipulação de Strings

Crie um programa que receba através do canal de entrada *default* uma string e teste se esta é um palíndromo.

`fgets (char * str, int num, FILE* stream)` lê uma string a partir do canal especificado

`strcmp (char * str1, char str2)` compara strings

# **Recursos complementares**

# Register

- Indica ao compilador que uma determinada variável será usada exaustivamente.
- Variáveis register não são armazenadas na memória principal, mas nos registradores localizados dentro do processador.
- Melhora o desempenho dos programas

```
register int x;
```

```
register char c;
```

# const

- Especifica que o valor de uma variável ou parâmetro de uma função não poderá ser modificado.
- Num array `const` os valores dos elementos permanecem inalterados.

```
const double pi = 3.141592654;
```

```
const char c[] = "Guilherme é legal";
```

# Operadores



# Operadores Aritméticos

- Operadores binários

+ soma

– subtração

\* multiplicação

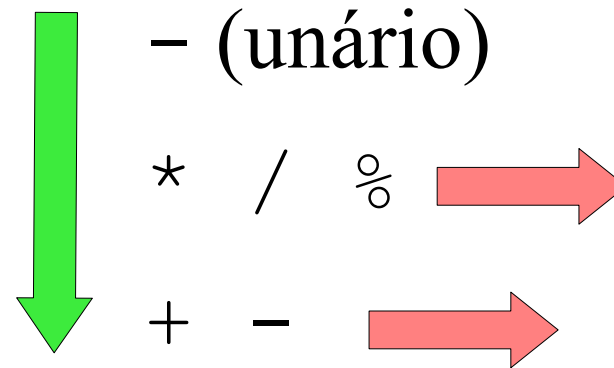
/ divisão

% resto da divisão

- Operador unário

– negativo

- Precedência e ordem de avaliação



- Exemplo

$$2 * 25.2 / 32 * -3$$
$$= -4.725$$

# Operadores Relacionais e Lógicos

- Operadores binários

> maior que

>= maior ou igual

< menor que

<= menor ou igual

== igual a

!= diferente

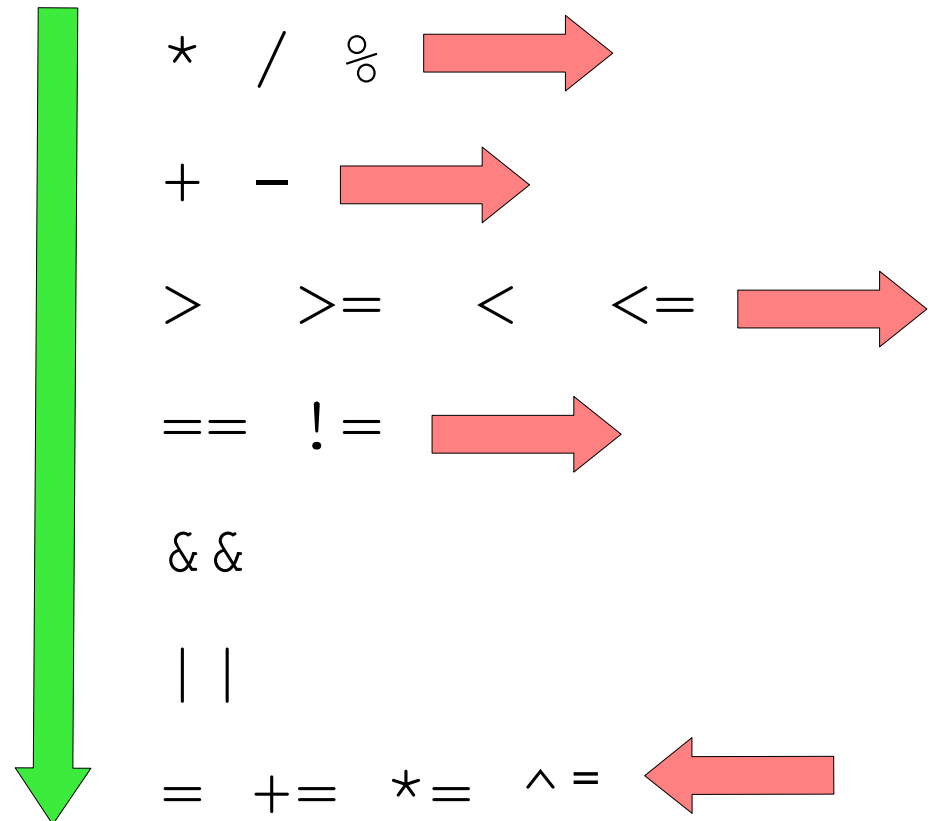
&& E lógico

|| OU lógico

- Operador unário

! negação

- Precedência e ordem



# Operadores Lógicos bit-a-bit

- Aplicáveis a operandos da família de inteiros:

`char`

`short`

`int`

`long`

`long long`

- Operadores binários

`&` E-Lógico

`|` OU-Lógico

`^` Ou-Exclusivo

- Operadores unários

`<<` *left shift*

`>>` *right shift*

`~` complemento

# Operadores Lógicos bit-a-bit

- Exemplos

```
int n, x, y, z;  
  
n = 0xFF & 0177; /* n = 0x7F */  
  
x = 0xF00 | 0xFF; /* x = 0xFFF */  
  
y = 0xFB << 2; /* y = 0x3EC */  
  
z = ~ 0xFF; /* z = 0xFF00 */
```

# Operadores de Atribuição

- Atribuição

```
n = 0xFF ;
```

- Incremento e decremento

```
n++; ++n; n--; --n;
```

- Atribuição + operador binário

```
n+=2; /* n=n+2  
-= *= /= %= <<= >>= &= ^= |= */
```

# Operadores: Precedência e Ordem de Avaliação

| Precedência dos operadores        | Ordem de avaliação |
|-----------------------------------|--------------------|
| () [] -> .                        | Esquerda → Direita |
| ! ~ ++ - + - * (type) sizeof      | Esquerda ← Direita |
| * / %                             | Esquerda → Direita |
| + -                               | Esquerda → Direita |
| << >>                             | Esquerda → Direita |
| < <= > >=                         | Esquerda → Direita |
| == !=                             | Esquerda → Direita |
| &                                 | Esquerda → Direita |
| ^                                 | Esquerda → Direita |
|                                   | Esquerda → Direita |
| &&                                | Esquerda → Direita |
|                                   | Esquerda → Direita |
| ?:                                | Esquerda ← Direita |
| = += -= *= /= %= &= ^=  = <<= >>= | Esquerda ← Direita |
| ,                                 | Esquerda → Direita |

# Conversões de Tipos

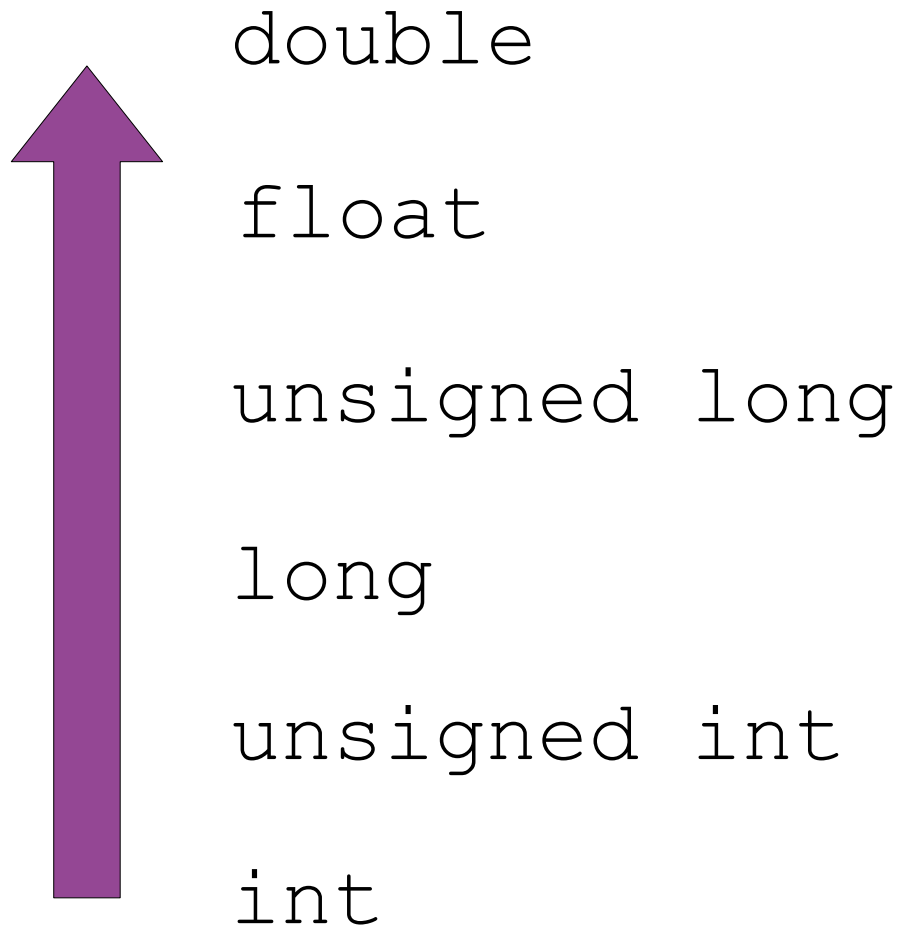
# Conversões Automáticas

- Expressões aritméticas fazem conversões automáticas de tipos
  - `var1 <operador> var2`
  - se `var1` e `var2` são do `tipo1` o resultado é `tipo1`
  - se `var1` e `var2` são de tipos diferentes, durante o cálculo um dos operandos é convertido
    - o resultado terá o mesmo para o qual o operando foi convertido



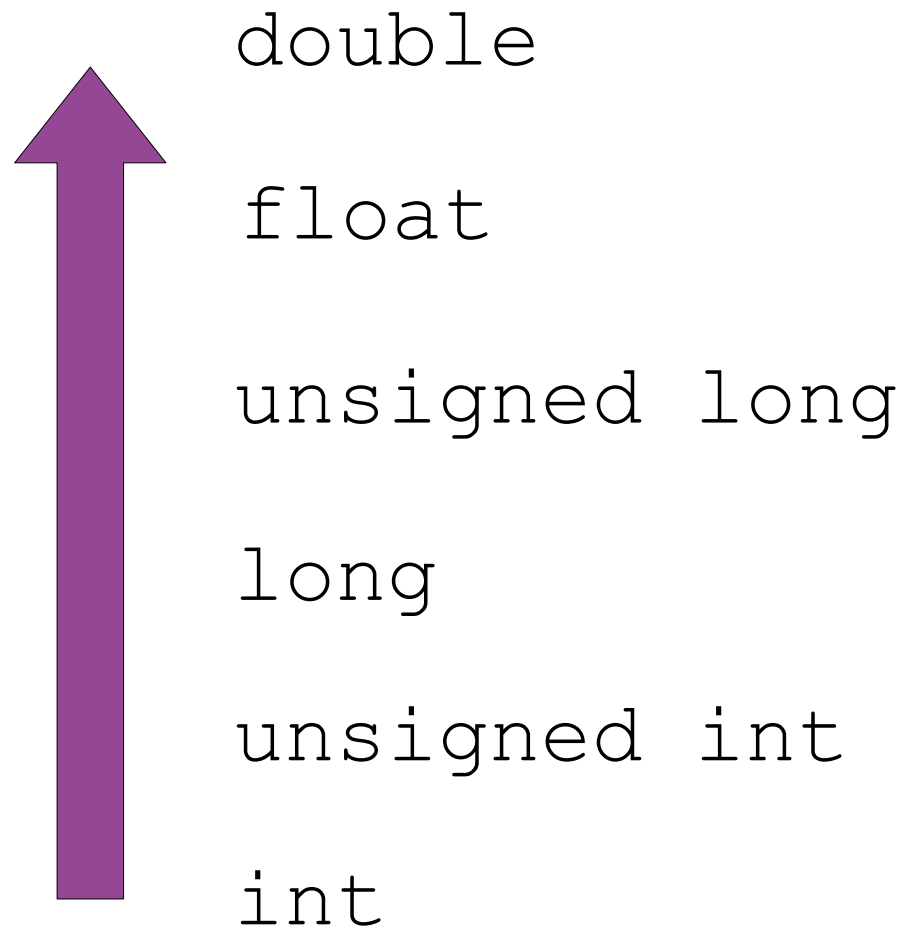
# Operandos de Tipos Distintos

- Hierarquia de Conversão



# Conversão Automática na Atribuição

- Do tipo menor para o tipo maior → OK



- Do tipo maior para o tipo menor → Truncamento

# Conversão Automática do tipo char

- Todo `char` é convertido para `short`

```
/* atoi: convert s to integer */
int atoi(char s[])
{
    int i, n;
    n = 0;
    for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
        n = 10 * n + (s[i] - '0');
    return n;
}
```

# Conversão Explícita de Tipo

- O operador de conversão de tipo (`<tipo>`) muda o tipo do resultado da expressão a sua direita

`(<tipo>) <expressão>`

- O tipo da expressão se mantém
- O resultado é convertido para `<tipo>`

# **Compilação e Tabelas de Símbolos**

# Tabela de Símbolos

- Estrutura de dados, normalmente uma hash, que armazena informações sobre os identificadores
- Na compilação:
  - Monitoramento dos identificadores em uso
  - Análise léxica, sintática e semântica
- Arquivo objeto:
  - Mapeamento das variáveis em endereços relativos de memória
- *Linking*:
  - definição dos endereços de memória absoluta
- *Debugging* de Executáveis
- Conjunto de entradas em bibliotecas compartilhadas

# Tabela de Símbolos

```
void myProc ( int A, short B)
{
    int D, E;
    D = 0;
    E = A / round(B);
    if ( E > 5)
        printf("%d", D);
}
```

| Símbolo | Token | Dtype    | Inicializado? | Endereço |
|---------|-------|----------|---------------|----------|
| myProc  | id    | procName | –             | 0x0E     |
| A       | id    | int      | SIM           | 0x00     |
| B       | id    | short    | SIM           | 0x04     |
| D       | id    | int      | SIM           | 0x06     |
| E       | id    | int      | SIM           | 0x0A     |

## Trabalho U2.4 - Conversão de Base

Crie um programa que dada uma string contendo um número na base 8 (número octal) a converta para uma string relativa ao número na base 2 (binário) correspondente. Em seguida, seu programa deve converter a string com a representação binária em uma string com a representação hexadecimal.



**Fim**