Linguagem C
Fundamentos

Sidney Batista Filho, M.Sc., SCPJ2P
sidneybf@acm.org

Setembro de 2002
Revisão: agosto de 2004

Agenda

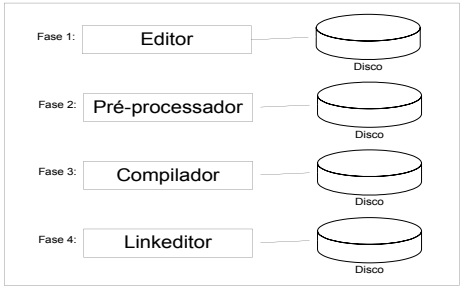
- Introdução
- Tipos de dados / Conversão
- Operadores
- Controle de Fluxo
- Arrays (uni e multidimensionais) / String
- Referências

Introdução
Níveis de linguagens de prog.

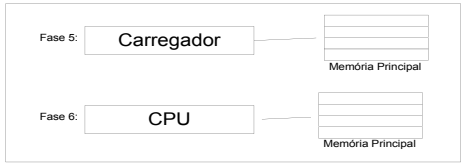
- Linguagens de máquina: 10010...100
  01100...001
  00110...110
- Linguagens assembly: LOAD BASE
  ADD EXTRA
  STORE BRUTO
- Linguagens de alto nível:

  bruto = base + extra

Introdução
Fundamentos do ambiente C



Introdução
Fundamentos do ambiente C



Introdução
Fundamenos do ambiente C

Como criar, compilar e executar um programa

`mkdir programas`	Criar diretório de programas
`cd programas`	Mudar para diretório de programas
`vi prog.c`	Editar programa fonte em formato ASCII
`gcc prog.c`	Compilar o programa fonte
`./a.out`	Executar o programa (em Unix)
`a.exe`	Executar o programa (em Windows)

Alternativa para se compilar

```
gcc prog.c -o prog.exe -lm
```

-o -> nome executável e -lm -> biblioteca math

Tipos de dados e tamanhos
O tipo char

- Uma variável do tipo `char` armazena um número inteiro, que representa o código de um caracter na tabela ASCII.

Principais códigos ASCII

- Caracteres de controle (de 0 a 32):
  0 = null 7 = bell 9 = tab
  10 = nova linha 32 = espaço
- Símbolos (de 33 a 47):
  33 = ! 34 = "
- Números (de 48 = 0 a 57 = 9)
- Letras maiúsculas (de 65 = A a 90 = Z)
- Letras minúsculas (de 97 = a a 122 = z)

Tipos de dados e tamanhos

- Tipos de dados básicos

Tipo	*Tamanho*	*Intervalo*
`char`	1 byte	de - 128 a 127
`int`	2 ou 4 bytes	de - 32768 a 32767, se 2 bytes OU de - 2147483648 a 2147483647, se 4 bytes
`float`	4 bytes	De 1.18E-38 a 3.4E+38 (+/-)
`double`	8 bytes	De 2.23E-308 a 1.8E+308 (+/-)

OBS: os tamanhos dos tipos int, float e double dependem do compilador e da máquina.

Tipos de dados e tamanhos

- Modificadores dos tipos básicos

Modificador	*Descrição*	*Tipo afetado*
`signed`	tipo com sinal (default)	char, int
`unsigned`	tipo sem sinal	char, int
`short`	tipo curto	int
`long`	tipo longo	int, double

```
main() {
      printf("Tipo \t Tam.(bytes)\n");
      printf("int \t %d\n", sizeof(int));
      printf("float \t %d\n", sizeof(float));
}
```


Tipos, Operadores e Expressões
Especificação de formatos

Formato	*Descrição*
`%d`	inteiro
`%nd`	inteiro com pelo menos *n* caracteres
`%f`	ponto flutuante
`%nf`	ponto flutuante com pelo menos *n* caracteres
`%.nf`	ponto flutuante com *n* caracteres após o ponto
`%n.mf`	ponto flutuante com *n* caracteres e *m* após o ponto
`%o`	octal
`%x`	hexadecimal
`%c`	caracter
`%s`	cadeia de caracteres (string)

Tipos, Operadores e Expressões
Seqüência de Escape

Seqüência	*Descrição*
`\n`	Nova linha
`\t`	Tabulação
`\a`	Alarme
`\\`	Imprime o caractere \
`\"`	Imprime o caractere "

Sistema de numeração

- Os números normalmente são representados no sistema de numeração decimal (base 10).
- Pode-se representar números em outras bases
- Os números em uma determinada base podem conter algarismos de 0 até (base - 1).

Tipos, Operadores e Expressões
Constantes

- Inteiras: 50, +14, -80
- Inteiras longas: 50L, 45l, 200L
- Octais (iniciam com 0): 023, 037, 020
- Hexadecimais (iniciam c/ 0x): 0x1f
- Ponto-flutuante: 10.4, 5e2, 4.1E2
- Caracter (é um inteiro): 'a', 97, '0', 48
- Cadeia de caracteres: "Alo Mundo!"

Tipos, Operadores e Expressões
Nomes de identificadores

- Podem conter apenas letras e dígitos.
- O primeiro caracter deve ser uma letra.
- O underscore "_" é considerdo letra.
- Letras maiúsculas e minúsculas são considerados caracteres diferentes.
- Não pode ser uma palavra reservada.

Tipos, Operadores e Expressões
Precedência e associatividade

Preced. e associat. de operadores

Livro Deitel, pág. 466, apêndice B

Exemplos:

c = a + b * c; a = b = 0;

Tipos, Operadores e Expressões
Operadores Aritméticos

Operador	*Operação*
++ --	incremento, decremento
* / %	multiplicação, divisão, módulo (resto da divisão inteira)
+ -	adição, subtração

OBS: O operador módulo (%) não pode ser aplicado para operandos float nem double.

Tipos, Operadores e Expressões
Operadores Relacionais e Lógicos

Operador	*Operação*
!	Operador de negação
> >= < <=	Operadores relacionais
== !=	igualdade diferença
&	E lógico
|	OU lógico
&&	E lógico
||	OU lógico

OBS: O resultado pode ser: 0 (falso) ou diferente de 0 (verdadeiro).

Controle de Fluxo
Sentenças e Blocos de sentenças

- Toda sentença, ou instrução, termina com um **;**

```
x = 0;
printf(...);
```

- Um bloco de sentenças é delimitado por { e }

```
{
   x = 5; x++;
}
```

Controle de Fluxo
Construção if-else

```
if (expressão)
    sentença1
else
    sentença2
```

- Se a *expressão* for diferente de 0 (verdade), a *sentença1* é executada. Caso contrário, a *sentença2* é executada.

Controle de Fluxo
Construção if-else (ambigüidade)

```
z = 10;
if (n > 0)
    if(a > b)
        z = a;
else
    z = b;
```

Cuidado! Equivale a:

```
z = 10;
if (n > 0){
    if(a > b)
        z = a;
    else
        z = b;
}
```

Controle de Fluxo
Estilo else-if

```
if (expressão1)
    sentença1
else if (expressão2)
    sentença2
else if (expressão3)
    sentença3
else
    sentença4
```


Tipos, Operadores e Expressões
Conversão de Tipos

- Quando um operador possui operandos de diferentes tipos, eles são convertidos em um tipo comum.
- A conversão será automática se for de um operando mais restrito (*narrower*) p/ outro mais abrangente (*wider*), sem perdas de informação.

```
char → int → float → double → long double
short ↗
```

Tipos, Operadores e Expressões
Conversão de Tipos

- Para forçar uma conversão usa-se o *cast*. Por exemplo:

```
int a = 5, b = 2, i;
float f = 4.5;

i = (int) f;
f = a / b; //divisão inteira. f = 2.0
f = (float) a / b; //divisão
                   //ponto flutuante. f = 2.5
f = 5.0 / b; //divisao ponto flut.f = 2.5
```


Operadores
Operadores de atribuição

var op= expr;

equivale à:

var = (var) op (expr);

Onde *op* pode ser: + - * / % << >> & ^ |

Controle de Fluxo
Construção switch

```
switch (expressão) {
   case exprConst1: sentenças [break;]
   case exprConst2: sentenças [break;]
   default: sentenças         [break;]
}
```

- *exprConst1* e *exprConst2* devem ser uma **constante inteira**.
- **default** é opcional e é executado se nenhum case for satisfeito.
- **break** é opcional e causa a saída imediata do switch.

Controle de Fluxo
Construção switch - Exemplo 1

```
char grau = 'A'; int contA = 0, contB = 0;
switch (grau) {
   case 'A':
   case 'a':
      ++contA;
      break;
   case 'B':
   case 'b':
      contB++;
      break;
   default:
      printf("Conceito invalido!\n");
      break;
}
printf("contA = %d\tcontB = %d\n", contA, contB);
```

Controle de Fluxo
Construção switch - Exemplo 2

```
int opcao = 3;
switch (opcao) {
   case 1:
      printf("Opcao 1\n");
      break;
   case 2:
   case 3:
      printf("Opcao 2 ou 3\n");
      break;
   case 4:
      printf("Opcao 4\n");
   case 5:
      printf("Opcao 4 ou 5\n");
      break;
}
```


Tipos, operadores e expressões
Expressão condicional

expr1 **?** ***expr2*** **:** ***expr3***

- Se *expr1* for diferente de 0 (verdade), então o resultado da expressão condicional será o resultado da *expr2*. Caso contrário, será o resultado da *expr3*.

```
int a = 3, b = 6, z;
z = (a > b) ? a : b; //z = 6
//equivale a:
if (a > b) z = a;
else
   z = b;
```

Controle de Fluxo
Construção while

while (*expressão*)
** *sentença***

- Enquanto a ***expressão*** for diferente de 0 (verdade), a sentença será executada.

```
#define MAX 5  //constante
main() {
   int cont = 0;
   while (cont < MAX) {
      printf("cont=%d\n", cont);
      cont++; } //fim do while
}
```


Entrada e saída de caracteres
getchar() e putchar()

- Funções usadas para ler/ecrever um caractere de cada vez da/na entrada/saída padrão.

```
int main() {
   char c;

   while ( (c = getchar()) != EOF ) {
      putchar(c);
      putchar('\n');
   }
   return 0;
}
```

Controle de Fluxo
Construção for

`for (<inic> ; <cond> ; <incr>)`
`  sentença`

- Passos da execução do comando for:
  - 1. A expressão <*inic*> é avaliada;
  - 2. A expressão <*cond*> é avaliada;
  - 3. Se o resultado de <cond> for VERDADE, a sentença é executada, caso contrário, o laço for é terminado;
  - 4. A expressão <incr> é avaliada;
  - 5. Volta ao passo2.

Controle de Fluxo
Construção for - Exemplo 1

Exemplo 1:

```
for (cont = 0; cont < 5; cont++) {
  printf("cont=%d\n", cont);
}
```

O trecho acima é equivalente à:

```
cont = 0;
while (cont < 5) {
  printf("cont=%d\n", cont);
  cont++;
}
```

Controle de Fluxo
Construção for - Exemplo 2

Exemplo 2:

```
for (;;) { //loop infinito
   ...
}
```


Controle de Fluxo
Construção do-while

```
do
   sentença
while (expressão);
```

- Passos da execução do comando do-while:
  - 1. Executa a sentença;
  - 2. Avalia a expressão;
  - 3. Se a expressão for VERDADE, volta ao passo 1, caso contrário, termina o do-while.
- A sentença é avaliada pelo menos uma vez.

Controle de Fluxo
Construção do-while - Exemplo

```
#define MAX 5

main() {
  int cont = 0;
  do {
    printf("cont=%d\n", cont);
    cont++;
  } while (cont < MAX);
}
```


Controle de Fluxo
Sentença break

- **break** faz com que a construção que a engloba seja terminada imediatamente.

```
#define MAX 20
main() {
   int cont = 1;
   while (cont < MAX) {
      //se cont for divisivel p/4 termina
      if ((cont % 4) == 0) break;
      printf("cont=%d\n", cont);
      cont++;
   }
}
```

Controle de Fluxo
Sentença continue

- **continue** interrompe a iteração e causa a próxima iteração do laço que a engloba.

```
#define MAX 5   //constante
main() {
   int cont = 0;
   while (cont < MAX) {
      //se for divisivel p/ 2
      if ((cont % 2) == 0) {
         cont++;
         continue;
      }
      printf("cont=%d\n", cont++);
   }
}
```


Vetor (array)
Introdução

- É uma estrutura de dados homogênea, ou seja, todos os seus elementos são do mesmo tipo.
- É uma estrutura de dados estática.
- Declaração:

  tipo identificador **[*tamanho*];**

- Espaço ocupado na memória (em bytes):

  espaço = tamanho * (espaço do tipo)

Vetor (array) Exemplo

```
#define TAM 4
main() {
   float   vetor[TAM];
   int     i = 0;
   vetor[i]=1.7; vetor[1]=2.0; vetor[2] = 1.2;
   vetor[3] = 9.3; //vetor[4] = 3.0; ERRADO!
   for(i = 0; i < TAM; i++) {
      printf("vetor[%d]=%.1f\n", i, vetor[i]);
   }
   printf("Espaco= %d\n", TAM * sizeof(float));
   printf("Espaco= %d\n", sizeof(vetor));
   printf("Espaco= %d\n",sizeof(vetor[0]));
}
```


Vetor (array) Declaração e inicialização

```
int n[10] = {0};
```

inicializa todos os elementos c/ zero.

```
int n[10] = {5};
```

inicializa o 1o. elemento com 5 e os demais c/ zero.

```
int n[3] = {4, 10, 20}; ou
int n[] = {4, 10, 20};
```

inicializa vetor de três elementos.

Vetor (array)
Exercício 1

Fazer um programa para criar um vetor de 10 elementos inteiros, inicializar este vetor com inteiros pares de 2 a 20 e imprimir a soma de todos os elementos do vetor.

Vetor (array)
Exercício 1 - solução

```
#define TAM 10
main() {
   int vetor[TAM];
   int i, soma = 0;

   for(i = 0; i < TAM; i++) {
      vetor[i] = 2 * i + 2;
      soma += vetor[i];
   }
   printf("soma = %d\n", soma);
}
```

Vetor (array)
Exercício 2

Entrar com 5 números inteiros, armazená-los em um vetor e imprimir a média dos elementos do vetor.

Vetor (array)
Exercício 2 - solução

```
#define TAM 5
main() {
   int  vetor[TAM];
   int  i, soma = 0;

   for(i = 0; i < TAM; i++) {
      printf("Digite o %do. numero: ", i+ 1);
      scanf("%d", &vetor[i]);
      soma += vetor[i];
   }
   printf("media=%.2f\n", soma/(float)TAM);
}
```

Cadeia de caracteres:String Introdução

- String é uma cadeia (array) de caracteres especial.

```
char cor[]={'a','z','u','l','\0'};
```

```
cor[0]    'a'
cor[1]    'z'
cor[2]    'u'
cor[3]    'l'
cor[4]    '\0'   → caractere nulo
```


String Exemplo 1

```
main() {
  char cor[]={'a', 'z', 'u', 'l', '\0'};
  char tamanho[] = "grande";
  char estado[TAM];
  int i;
  printf("Digite a sigla do estado: ");
  scanf("%s", estado); //SEM &
  printf("cor=%s - tamanho=%s - "
         "estado=", cor, tamanho);
  for(i = 0; estado[i] != '\0'; i++)
    printf("%c", estado[i]);
}
```

String
Exemplo 2

```
#include <string.h>
#define TAM  20
main() {
   char item1[TAM], item2[TAM];
   float capac1 = 5.4, capac2 = 1.5;
   char unidade1[] = "grama";
   char unidade2[] = "litro";
   sprintf(item2, "%6.2f ", capac2);
   strcat(item2, unidade2);
   sprintf(item1, "%6.2f %s",
           capac1, unidade1);
   printf("item1=%s\nitem2=%s",
           item1, item2);
}
```


String
string.h

Algumas funções da biblioteca string.h

Funções de cadeia	*Descrição*
`strcat(dest, ori)`	Concatena cadeia origem ao final de destino
`strlen(str)`	Calcula o número de caracteres da cadeia sem o caractere nulo.
`strlwr(str)`	Converte cadeia para minúsculas
`strupr(str)`	Converte cadeia para maiúsculas
`strcpy(dest, ori)`	Copia cadeia origem para destino
`strcmp(str1, str2)`	Compara duas cadeias. Retorna zero se iguais, menor que zero se str1 < str2 e maior que zero se str1 > str2.

Array multidimensional Introdução

- Em C, arrays (arranjos) podem ter mais de uma dimensão.
- Declaração:

```
tipo ident [dim1][dim2]...[dimN];
```

- Espaço ocupado na memória (em bytes):

  espaço = sizeof(tipo) * dim1 * ... * dimN.

Array multidimensional Exemplo 1

```
#define DIML 3
#define DIMC 2
main(){
   int i, j;
   int matriz[DIML][DIMC];

   for (i = 0; i < DIML; i++)
      for (j = 0; j < DIMC; j++)
         scanf("%d", &matriz[i][j]);
   printf("Espaco = %d\n", sizeof(matriz));
   for (i = 0; i < DIML; i++){
      for (j = 0; j < DIMC; j++)
         printf("%4d", matriz[i][j]);
      printf("\n");
   } /* for */  } /* main */
```

Array multidimensional
Exemplo 2

```
#define DIML 3
#define DIMC 2
main(){
   int i, j;
   int matriz[DIML][DIMC] = {{1, 2},
                             {3, 4},
                             {5, 6}};
   for (i = 0; i < DIML; i++){
      for (j = 0; j < DIMC; j++)
         printf("%4d", matriz[i][j]);
      printf("\n");
   }
}
```


Array multidimensional
Revisão

```
char vetor1[6]; //vetor de caracteres

char vetor2[3] = {'a', 'b', 'c'};//vetor de caracteres

char vetor3[4] = {'a', 'b', 'c', '\0'};//string

char vetor4[3][4] = {
                      {'p', 'r', 'i', 'm'},
                      {'s', 'e', 'g', 'u'},
                      {'t', 'e', 'r', 'c'}
                    }; //matriz de caracteres
```

Array multidimensional
Vetor de cadeia de caracteres

```
char vetor5[3][5] = {
                       {'p', 'r', 'i', 'm', '\0'},
                       {'s', 'e', 'g', 'u', '\0'},
                       {'t', 'e', 'r', 'c', '\0'}
                    }; //vetor de 3 strings
char vetor6[3][5] = {
                       {"prim"},
                       {"segu"},
                       {"terc"}
                    }; //vetor de 3 strings

//vetor5 e vetor6 são "equivalentes"
```


Array multidimensional
Vetor de cadeia de caracteres

```
#define DIML 3
#define DIMC 40
main(){
     int i;
     char nomes[DIML][DIMC];

     for (i=0; i<DIML; i++){
         printf("Entre com a linha %d: ", i);
         gets(nomes[i]);
     }
     for (i=0; i<DIML; i++){
         printf("O nome %d e\n", i);
         puts(nomes[i]);
     }
}
```

Array multidimensional
Exercício

Fazer um programa para ler o nome e as 2 notas (PR1 e PR2) de 20 alunos, calcular a média e a situação dos alunos. Ao final, apresentar o nome, as notas, a média e a situação dos alunos.

Array multidimensional
Exercício - Solução (versão 1)

```
#define TAM 20
main() {
   float pr1[TAM], pr2[TAM], media[TAM];
   char nome[TAM][30], sit[TAM][12];
   int i;

   for (i = 0; i < TAM; i++) {
      printf("Entre com o nome: ");
      gets(nome[i]);
      printf("1a nota: "); scanf("%f", &pr1[i]);
      printf("2a nota: "); scanf("%f", &pr2[i]);
      printf("Codigo do caractere no "
         "buffer de entrada = %d\n", getchar());
   }
   ...
```

Array multidimensional
Exercício - Solução (versão 1) (cont.)

```
   ...
   for (i = 0; i < TAM; i++) {
      media[i] = (pr1[i] + pr2[i]) / 2;
      if (media[i] >= 7) {
         //sit[i] = "APROVADO"; E R R O !
         strcpy(sit[i], "APROVADO");
      } else if (media[i] >= 4) {
         strcpy(sit[i], "PROVA FINAL");
      } else
         strcpy(sit[i], "REPROVADO");
   }
   for (i = 0; i < TAM; i++)
      printf("%-30s %5.2f %5.2f %5.2f %s\n",
         nome[i], pr1[i], pr2[i], media[i], sit[i]);
}
```


Array multidimensional
Exercício - Solução (versão 2)

```
main() {
   float notas[TAM][3];
   char nome[TAM][30], sit[TAM][12];
   int i;

   for (i = 0; i < TAM; i++) {
      printf("Entre com o nome: ");
      gets(nome[i]);
      printf("1a nota: "); scanf("%f",&notas[i][1]);
      printf("2a nota: "); scanf("%f",&notas[i][2]);
      printf("Codigo do caractere no "
         "buffer de entrada = %d\n", getchar());
   }
   ...
```

Array multidimensional
Exercício - Solução (versão 2) (cont.)

```
   ...
   for (i = 0; i < TAM; i++) {
      notas[i][0] = (notas[i][1] + notas[i][2]) / 2;
      if (notas[i][0] >= 7) {
         strcpy(sit[i], "APROVADO");
      } else if (notas[i][0] >= 4) {
         strcpy(sit[i], "PROVA FINAL");
      } else
         strcpy(sit[i], "REPROVADO");
   }
   for (i = 0; i < TAM; i++)
      printf("%-30s %5.2f %5.2f %5.2f %s\n",
         nome[i], notas[i][1], notas[i][2],
         notas[i][0], sit[i]);
}
```


Referências

- Kernighan, B.W.; Ritchie, D.M. The C Programming Language. New Jersey: Prentice-Hall, Inc. Second Edition, 1988.
- Deitel, H.M.; Deitel, P.J. Como Programar em C. Rio de Janeiro: Editora LTC. Segunda Edição, 1999.
- Sun Microsystems, Inc. Numerical Computation Guide. Palo Alto: Sun Microsystems, Inc. 2001. Disponível em: http://docs.sun.com/
- http://equipe.nce.ufrj.br/adriano/c/apostila/indice.htm