Pedro Kantek

RESUMO DE C

UP

1 C

Esta linguagem foi desenvolvida por D Richie no final dos anos 70 nos laboratórios Bell.

1.1 Estrutura de um programa

```
#include<stdio.h>  // arquivo cabecalho
int i,j,k; // variaveis globais
int funcao(int A, float B) {
   int i,j,k; // variaveis locais à função
   ... comandos da função
   return(0);
}
main(int ac, char *av[]) {
   int i,j,k; // variáveis locais à main
   char lixo;
   ... comandos...
   lixo = getch(); // para ler alguma coisa
   //                antes da janela fechar
   return(0);
}
```

Note que a execução do comando `return` encerra a função atual, ainda que esta tenha mais comandos abaixo do return.

1.2 Variáveis

1.2.1 Tipos

Podem ser
char, int, float, void e double.

1.2.2 Modificadores

Podem ser
signed(int, char)
unsigned(int, char)
long(int, double)
long long int
short(int)
char string[tamanho]

1.2.3 Faixas

Podem ser

tipo	tamanho (bits)	faixa
char	8	-127 a 127
unsigned char	8	0 a 255
signed char	8	-127 a 127
int	16	-32767 a 32767
unsigned int	16	0 a 65.535
signed int		igual a int
short int		igual a int
unsigned short int	16	0 a 65535
signed short int		igual a short int
long int	32	-2147483647 a 2147483647
signed long int	32	o mesmo que long int
unsigned long int	32	0 a 4294967295
float	32	seis dígitos precisão
double	64	dez dígitos precisão
long double	80	dez dígitos precisão

1.3 Constantes

As constantes numéricas são escritas normalmente. As constantes alfanuméricas estão entre aspas duplas. As constantes octais começam com 0= e as constantes hexadecimais começam om 0x=.

Por exemplo:

```
int hex = 0x80; // 128 em decimal
int oct = 012; // 10 em decimal
```

1.4 Barra invertida

Usadas dentro de constantes alfanuméricas

`\b`	back space
`\f`	form feed
`\n`	new line
`\t`	tabulação
`\v`	tabulação vertical
`\"`	aspas duplas
`\'`	aspa simples
`\0`	zero decimal (nulo)
`\\`	uma barra invertida
`\a`	alerta (beep)
`\r`	return
`\0oo`	numero octal ***
`\xhh`	numero hexadecimal

1.5 Palavras reservadas

auto break case char const continue default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while

1.6 Operadores aritméticos

com int e float

`+`	soma
`-`	subtração
`*`	multiplicação
`/`	divisão
`++`	incremento
`-`	decremento
`%`	resto da divisão (só com int)

1.7 Operadores relacionais

Retornam F=0 e V=qualquer coisa diferente de 0.

`>`	maior
`>=`	maior ou igual
`<`	menor
`<=`	menor ou igual
`==`	igual
`!=`	diferente
`&&`	e lógico
`||`	ou lógico
`!`	não lógico

1.8 Operadores bit a bit

`&`	and
`|`	or
`^`	xor
`~`	not

1.9 Deslocamentos de bit

`>>n`	n para a direita
`<<n`	n para a esquerda

2 Comandos

A seguir, descrevem-se os poucos comandos de C.

2.1 if

```
if (condição) {
   ação1;
   ...
   }
   [else {
   ação2;
   ...
   }]
```

Se a condição é verdadeira (isto é diferente de zero) a ação1 é executada. Se a condição é falsa (igual a 0) e a cláusula else existe, a ação2 é executada. Se else não existe, nada ocorre.

Exemplo

```
if ((a>=4)&&(b!=9)){
   a++;
   }
else {
   b++;
   }
```

2.2 for

```
for(inicialização;condição;passo) {
   ação
   ...
   }
```

Antes de tudo, execute o bloco de inicialização. Teste a condição. Se ela for falsa (diferente de 0) caia fora. Se ela for verdadeira, execute a ação. Em seguida execute o bloco de passo. Teste a condição. Se ela for falsa (diferente de 0) caia fora. Se ela for verdadeira, volte a executar a ação, e assim por diante.

Exemplo

```
for(i=1;i<=10;i++){
   printf("%i ",i);
   }
------------------
for(;;) printf("para sempre\n"); // laço infinito
```

O comando acima vai imprimir a sequência 1,2,...,10.

2.3 switch

```
switch(c) {
   case '0' : ação1;
   case '1' : ação2;
   ...
   default : açãon;
   }
```

Caso a variável c seja igual a 0 execute ação1. Se for igual a 1 execute a ação2 e assim por diante. Se não for nenhum dos valores citados, execute a ação n.

Exemplo

```
int alfa;
...
switch (alfa) {
   case 1 : x++;
   case 2 : y++;
   }
----------------------
int flag = -1;
switch(i) {
   case 1:
   case 2:
   case 3:
      flag = 0;
      break;
   case 4:
   ...
}
```

Note que este comando sempre pode ser substituído por ifs.

2.4 while

```
while(condição){
   ação1;
   ...
   }
```


A condição é verificada. Se verdadeira, a ação1 é executada. Ao encontrar a } (chave que encerra o bloco) há um retorno ao comando `while` quando a condição é reavaliada. Em qualquer avaliação, se a condição é falsa (igual a 0) o bloco é saltado e executa-se o próximo comando após o `while`.

Exemplo

```
int beta = 0;
...
while (beta < 10) {
   printf("%i ",beta);
   beta++;
   }
```

Serão impressos os valores 0,1,...,8,9.

2.5 do

```
do {
   ação1;
   ...
   }
   while(condição);
```

A ação1 é executada. Ao encontrar a palavra while, a condição é avaliada. Se verdadeira o processamente cai fora seguindo após o while. Se falsa (igual a 0) há um retorno ao início do bloco, e a ação1 é reexecutada.

Exemplo

```
int i = 10;
...
do {
   printf("%i ",i);
   i++;
   } while (i > 15);
```

Vão ser impressos 10, 11, 12, 13, 14 e 15.

2.6 Saídas

Existem diversos comandos para forçar saídas de switch, for, while e do.

`break;`	causa saída imediata
`continue;`	causa uma próxima interação do laço
`exit();`	causa uma saida do programa
`return(0);`	causa saída função atual

2.7 goto

```
if (...) {
   goto lugar;
   }
...
lugar:
...
```

Recomenda-se fortemente que este comando NÃO seja usado. Ele só está aqui por uma certa compatibilidade com linguagens antigas (da época das linguagens pré-estruturadas).

3 Vetor

Um vetor é uma variável que tem um único nome e diversas ocorrências. Cada ocorrência é acessada pelo seu número (começando em 0). Lembrar que todos os dados compartilham o mesmo tipo, isto é são homogêneos.

```
tipo nome[tamanho];
```

A inicialização de um vetor numérico é:
int numa[10]=1,2,3,4,5,6,7,8,9,10;
Quando o vetor é formado por caracteres (char) ele também recebe o nome de string. Um string sempre deve terminar pelo caractere `\0`, e este elemento ocupa espaço.

3.1 Comandos de string

Todas estas funções usam o cabeçalho STRING.H

nome	função
strcpy(a,b)	copia b em a
strcat(a,b)	concatena b ao final de a
strlen(a)	retorna o tamanho de a
strcmp(a,b)	retorna 0 se iguais; negativo se $a < b$ e positivo se $a > b$
strchr(a,ch)	retorna ponteiro para a primeira ocorrência de ch em a
strstr(a,b)	retorna ponteiro para a primeira ocorrência de b em a

A inicialização de um vetor caracter é:
char alfa = ”Isto e uma variavel caracter”;

4 Matriz

O jeitão de declarar uma matriz é

```
tipo nome[linhas][colunas];
```

Um conjunto de strings também forma uma matriz de caracteres. Neste caso, para acessar um determinado string, basta fornecer o primeiro índice. Por exemplo

```
char strings [5] [100];
int i;
for (i=0;i<5;i++) {
   printf("\n digite uma string: ");
   gets(strings[i]);
   }
printf("\n\n As strings digitadas foram: \n");
for (i=0;i<5;i++) {
   printf("%s\n",strings[i]);
   }
```

Uma matriz pode ser multi-dimensional, colocando-se quantas especificações se quiser dentro de colchetes. Como sempre a maior variação está no índice mais à direita.
Uma matriz pode ser inicializada, vejam-se os exemplos

```
float ve[6]={1.3, 4.5, 2.7, 4.1, 0.0, 100.1};
int matr[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
int oba[3][10]={{1,2,3,4,5,6,7,8,9,10},{20,30,40,50,6
                {10,9,8,7,6,5,4,3,2,1}};
char str[10]={'J', 'o', 'a', 'o', '\0'};
char str_ve[3][10]={"Joao", "Maria", "Jose"};
int tabela [2][3] = {{1,2,3}, {4,5,6}};
```


Note que o tamanho do vetor pode ser omitido quando o mesmo é inicializado: por exemplo em:

```
int alfa[] = {1,4,6,7,8,10};
```

Se o vetor não é inicializado, o tamanho explícito é obrigatório.

5 Funções

Funções são chamadas em C, com a passagem dos parâmetros por valor. Para a passagem por referência, deve-se passar não a variável e sim um ponteiro para ela. Agora, modificações que a função faça nessa variável se refletem na variável original. Veja-se um exemplo de passagem por referência

```
void swap (int *x, int *y) {
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
```

6 Estruturas

É a coleção de um conjunto de variáveis de tipos diferentes referenciadas por um único nome. É uma coleção heterogênea, portanto. Eis a definição:

```
struct nome {
   tipo vari1;
   tipo vari2;
   ...
   } ;
```

Note que a estrutura termina por um ;. Depois da definição da estrutura nenhuma variável foi definida ainda. Isto ocorre quando se faz

```
struct nome-estrutura nome-variável;
```

Se bem que ambas coisas podem ser feitas juntas, veja no exemplo

```
struct ende {
   char rua[30];
   int num;
   } ende_compr, ende_vende;
```

Note ainda que se for necessária apenas uma variável com essa estrutura, nem o nome da estrutura é necessário, veja:

```
struct {
   char rua[30];
   int num;
   } endereco;
```

Para referenciar um campo dentro de uma variável heterogênea (que usa uma estrutura) usa-se o ponto. Seu formato

```
nome-estrutura.nome-campo
```

Veja o exemplo:

```
ende_compr.rua = "Mateus Leme";
ende_compr.num = 1561;
```

Finalmente, dados dois conjuntos heterogêneos que tenham a mesma estrutura pode-se atribuir um ao outro, como em

```
ende_compr = ende_vende;
```

7 E/S pela console

Entrada e saída simples

função	operação
getch()	Lê um caractere no teclado, sem eco, espera o enter
getche()	Lê um caractere no teclado, com eco, não espera o enter
getchar()	Lê um caractere no teclado, espera o enter
putchar()	Escreve um caractere na console
gets()	Lê uma string do teclado
puts()	Escreve uma string na console

8 Entrada e saída formatada

Descrevem-se aqui os comandos de entrada e saída na console, usando os arquivos STDIN e STDOUT, ou seus eventuais redirecionamentos.

8.1 Redirecionamento

Uma questão importante é que `scanf` pode ler diretamente de um arquivo assim como `printf` pode imprimir também num arquivo, desde que isso seja informado na chamada do programa. Para que isto funcione, crie um arquivo de nome `ENTRADA` e coloque nele todas as entradas que o programa solicita em STDIN quando é executado. Daí chame o programa redirecionando a entrada para o arquivo entrada (`PROGRAMA < ENTRADA`).
Se usar o redirecionamento de saída, rode o programa como `PROGRAMA > SAIDA` e depois de executar o programa examine o arquivo `SAIDA`, fazendo um TYPE (cat nop linux) ou chamando-o com um editor, como notepad ou notepad++.
Veja nos exemplos:

PROGR1 Enter

Chama a função `main()` do programa `PROGR1`. Nele comandos `scanf` serão lidos do teclado e comandos `printf` serão impressos na console.

PROGR1 < ENTRADA Enter

Chama a função `main()` do programa `PROGR1`. Nele comandos `scanf` serão lidos do arquivo de nome `ENTRADA` e comandos `printf` serão impressos na console.

PROGR1 < ENTRADA > SAIDA Enter

Chama a função `main()` do programa `PROGR1`. Nele comandos `scanf` serão lidos do arquivo de nome `ENTRADA` e comandos `printf` serão impressos (gravados) no arquivo de nome `SAIDA`.
Perceba como este ajuste é altamente favorável para modificar a entrada e a saída padrões.

9 Funções

9.1 abs(int i)

Devolve um inteiro com o valor absoluto de i. Exige `#include<stdlib.h>`. Funções relacionadas: `labs()`.

9.2 acos(double arg)

Devolve o arco coseno de arg em radianos. O argumento deve estar entre -1 e 1, senão ocorre erro de domínio. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas.

```
#include<stdio.h>
#include<math.h>
main() {
   double val = -1.0;
   do {
      printf("cos=%f, arco=%f\n", val, acos(val));
      val = val + 0.1;
    } while (val <= 1.0);
}
```

9.3 asin(double arg)

Devolve o arco seno de arg em radianos. O argumento deve estar entre -1 e 1, senão ocorre erro de domínio. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas.

```
#include<stdio.h>
#include<math.h>
main() {
   double val = -1.0;
   do {
      printf("sen=%f, arco=%f\n", val, asin(val));
      val = val + 0.1;
    } while (val <= 1.0);
}
```

9.4 atan(double arg)

Devolve o arco tangente de arg em radianos. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas.

```
#include<stdio.h>
#include<math.h>
main() {
   double val = -1.0;
   do {
      printf("tan=%f, arco=%f\n", val, atan(val));
      val = val + 0.1;
    } while (val <= 1.0);
}
```

9.5 atan2(double y, double x)

Devolve o arco tangente de y/x. Usa o sinal dos argumentos para calcular o quadrante do valor devolvido. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas.

9.6 atof(*str)

Converte a string apontada por *str* em um valor double. A string deve conter um número em ponto flutuante válido e pode terminar com qualquer caractere que não seja parte de flutuante válido. Se for chamado com "123.5agora vai", devolverá 123.5000. Exige `#include<stdlib.h>`. Funções relacionadas: `atoi()` e `atol()`.

9.7 atoi(*str)

Converte a string apontada por *str* em um valor inteiro. A string deve conter um número inteiro válido e pode terminar com qualquer caractere que não seja parte de inteiro válido. Se o parâmetro não é válido, o resultado é indefinido, mas muitas implementações retornam zero. Se for chamado com "123.5agora vai", devolverá 123. Exige `#include<stdlib.h>`. Funções relacionadas: `atof()` e `atol()`.

9.8 atol(*str)

Converte a string apontada por *str* em um valor inteiro longo. A string deve conter um número inteiro longo válido e pode terminar com qualquer caractere que não seja parte de um inteiro longo válido. Se for chamado com "123.5agora vai", devolverá 123. Exige `#include<stdlib.h>`. Funções relacionadas: `atof()` e `atol()`.

9.9 ceil(double arg)

Devolve o teto, também conhecido como $\lceil$. Retorna um double. O teto de 1.1 é 2.0; o teto de 1.0 é 1.0 e o teto de -1.1 é -1.0. Exige `#include<math.h>`. Funções relacionadas: `floor()` e `fmod()`.

```
printf("%f",ceil(7.33)); // imprime 8.0
```

9.10 cgets(*str)

Lê uma string do teclado para a string apontada. Antes da chamada, o primeiro byte de *str* deve conter o tamanho máximo a ler. Depois da leitura o segundo byte contém a quantidade realmente lida. O processo termina quando se digitar um Enter que é convertido em nulo para terminar a string. O texto digitado começa em str[2] e é este endereço que a fgets devolve. Exige `#include<conio.h>`. Funções relacionadas: `cputs()`, `gets()`, `fgets()` e `puts()`.

```
#include<stdio.h>
#include<conio.h>
main() {
   char str[80];
   str[0]=40;
   cputs("digite algo \n");
   cgets(str);
   cputs(&str[2]);
}
```

9.11 clearerr(FILE *fp)

Zera o indicador de erro para este arquivo. Isto já é normalmente feito pelo `fopen`, mas uma vez setado o erro ele permanece até ser limpo. Exige `#include<stdio.h>`. Funções relacionadas: `feof()`, `ferror()` e `perror()`.

9.12 cos(double arg)

Devolve o coseno de arg. Arg está em radianos. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas.


```
#include<stdio.h>
#include<math.h>
main() {
   double val = -1.0;
   do {
      printf("Arco=%f, cos=%f\n", val, cos(val));
      val = val + 0.1;
    } while (val <= 1.0);
}
```

9.13 cosh(double arg)

Devolve o coseno hiperbólico de arg. Arg está em radianos. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas. Sua fórmula é

$$cosh(x) = \frac{e^x + e^{-x}}{2}$$

.

9.14 cputs(*str)

Escreve na tela a string apontada. Devolve o último caracter escrito ou `EOF` se não obtiver sucesso. Exige `#include<conio.h>`.

9.15 div(int num, int den)

Devolve o quociente e o resto da divisão numa estrutura do tipo `div_t`, definida em `STDLIB.H`. Essa estrutura tem pelo menos dois campos: int quot e int rem. Exige `#include<stdlib.h>`. Função relacionada: `ldiv()`.

```
#include<stdlib.h>
#include<stdio.h>
main() {
   div_t n;
   n = div(10, 3);
   printf("%d %d",n.quot,n.rem);
}
```

9.16 exit(int code)

Provoca o fim normal de um programa. O valor do *code* é passado ao sistema operacional. A convenção é que 0 significa um fim normal. Exige `#include<stdlib.h>`. Função relacionada: `abort()` e `atexit()`.

9.17 exp(double arg)

Devolve *e* elevado a arg. Devolve um double Exige `#include<math.h>`. Funções relacionadas: `log()`.

```
printf("e=%f",exp(1.0)); // imprime e=2.718282
```

9.18 fabs(double num)

Devolve o valor absoluto de *num*. Devolve um double Exige `#include<math.h>`. Funções relacionadas: `abs()`.

9.19 fclose(FILE *fp)

Fecha um arquivo que foi aberto por meio de uma chamada a `fopen()`. Escreve o que ainda porventura exista no buffer e pede para o sistema operacional fechar o arquivo. Esquecer de fechar um arquivo atrai desgraça. Um retorno zero significa operação bem sucedida e qualquer coisa diferente disso indica problemas. Exige `#include<stdio.h>`.

```
#include<stdio.h>
main() {
FILE *fp;
int x;
fp = fopen("c:/tcc/alfa.zzz","w");
putc('a',fp);
x = fclose(fp);
}
```

9.20 feof(FILE *fp)

O uso de `EOF` não é recomendado em arquivos binários, pois ele pode aparecer sem que o final do arquivo tenha sido alcançado. Não esqueça que em arquivo binário pode aparecer qualquer coisa. Esta função devolve verdadeiro se o final do arquivo é alcançado e falso senão. Se quiser, pode usar esta função também para arquivos texto. Exige `#include<stdio.h>`.

```
while (!feof(fp)) ch=getc(fp);
```

9.21 fflush(FILE *fp)

Esta função quando associada a um arquivo aberto de saída, escreve fisicamente no arquivo o conteúdo do buffer. Se for um arquivo de entrada o buffer de entrada é esvaziado. O arquivo permanece aberto e esta é a diferença em relação a fechar o arquivo, Exige `#include<stdio.h>`. Funções relacionadas: `fclose()`, `fopen()`, `fread()`, `fwrite()`, `getc()` e `putc()`.

9.22 fgetc(FILE *fp)

Devolve o próximo caractere do arquivo e incrementa o indicador de posição. O caractere volta como um inteiro. Quando o fim de arquivo é alcançado volta `EOF`. Mas como este valor é um inteiro válido, deve-se usar `feof()` para isso. Se encontra um erro também volta `EOF` e aqui deve-se usar `ferror()`. Exige `#include<stdio.h>`. Funções relacionadas: `fputc()`, `getc()`, `fopen()` e `putc()`.

```
#include<stdio.h>
#include<stdlib.h>
main(int ac, char *av[]) {
   FILE *fp;
   char ch;
   if((fp=fopen(av[1],"r"))==NULL) {
      printf("arquivo deu xabu\n");
  exit(1);
   }
   while((ch=fgetc(fp))!=EOF) {
      printf("%c",ch);
   }
   fclose(fp);
}
```


9.23 fgets(*str, num, FILE *fp)

Le *num* caracteres do arquivo *fp* e coloca-os em *str*. Os caracteres são lidos até que uma nova linha, ou um `EOF` ou o limite especificado seja atingido. Após a leitura um nulo é colocado ao final de *str*. O caractere de nova linha é preservado e faz parte de *str*. Em caso de sucesso volta um endereço para *str* e em caso de falha volta um nulo. Exige `#include<stdio.h>`. Funções relacionadas: `fputs()`, `fgetc()`, `gets()` e `puts()`.

```
#include<stdio.h>
#include<stdlib.h>
main(int ac, char *av[]) {
   FILE *fp;
   char ch[120];
   if((fp=fopen(av[1],"r"))==NULL) {
      printf("arquivo deu xabu\n");
  exit(1);
   }
   while (!feof(fp)) {
      if (fgets(ch, 118, fp)) {
     printf("%s", ch);
  }
   }
   fclose(fp);
}
```

9.24 floor(double arg)

Devolve o chão, também conhecido como $\lfloor$. Retorna um double. O chão de 1.1 é 1.0; o chão de 1.0 é 1.0 e o chão de -1.1 é -2.0. Exige `#include<math.h>`. Funções relacionadas: `ceil()`.

```
printf("%f",floor(7.33)); // imprime 7.0
```

9.25 fopen(*nome, *modo)

Abre um arquivo e retorna um ponteiro associado a ele. Se ocorrer um erro, o ponteiro volta nulo. O arquivo pode ser texto ou binário. O arquivo tipo texto tem linhas terminadas por retorno de carro mais alimentação de linha. Não há nenhum tipo de terminador em arquivos binários. Os modos de abertura de um arquivo podem ser

modo	significado
r	arquivo texto; só para leitura
w	arquivo texto; para escrita
a	anexa um arquivo texto
rb	arquivo binário; só leitura
wb	arquivo binário; para escrita
ab	anexa um arquivo binário
r+	arquivo texto; leitura e escrita
w+	cria um arquivo texto para leitura e escrita
a+	anexa ou cria um arquivo texto para leitura e escrita
r+b	abre arquivo binário para leitura e escrita. Pode ser rb+
w+b	cria um binário para leitura e escrita
a+b	anexa a um arquivo binário para leitura e escrita.

Se for aberto um arquivo para escrita e já houver arquivo com este nome ele será apagado e um novo com o mesmo nome será criado. Se o arquivo não existe ele será criado. O modo "a"adiciona dados ao final de um arquivo já existente (ou que acabou de ser criado, se ele não existia). Exige `#include<stdio.h>`.

```
FILE *fp;
fp = fopen("c:\lixo\alfa","w");
------------ ou ------------
if ((fp = fopen("teste","w")) == NULL) {
   printf("arquivo nao pode ser aberto\n");
   exit(1);
}
```

9.26 fmod(double arg)

Devolve o resto da divisão de x por y. Retorna um double. Exige `#include<math.h>`. Funções relacionadas: `floor()` e `fabs()` e `ceil()`.

9.27 fprintf(FILE *fp, controle, campos...)

Escreve no arquivo de maneira similar a como `printf` escrevia na console. Veja o comando `printf` para mais detalhes. Exige `#include<stdio.h>`. Funções relacionadas: `printf()`, `fscanf()`.

```
fprintf(fp, "vamos se %i e maior do que %i\n",a,b);
```

9.28 fputc(ch FILE *fp)

Escreve o caracter *ch* no arquivo e avança o indicador de posição do arquivo. Devolve o caracter escrito. Ocorrendo erro volta `EOF`. Exige `#include<stdio.h>`. Funções relacionadas: `fgetc()`, `fopen()`, `fprintf()`, `fread()` e `fwrite()`.

```
while (*st) if (!ferror(fp)) fputc(*st++,fp);
```

9.29 fputchar(ch)

Escreve o caracter *ch* em `STDOUT` e avança o indicador de posição do arquivo. Devolve o caracter escrito. Ocorrendo erro volta `EOF`. Exige `#include<stdio.h>`. Funções relacionadas: `fgetc()`, `fopen()`, `fprintf()`, `fread()` e `fwrite()`.

```
while (*st) if (!ferror(fp)) fputchar(*st++);
```

9.30 fputs(*str, FILE *fp)

Escreve uma string num arquivo. Se ocorrer um erro `EOF` será devolvido. Exige `#include<stdio.h>`.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main() {
   char str[80];
   FILE *fp;
   if ((fp = fopen("ALPHA","w"))==NULL) {
      printf("arq ALPHA nao pode ser aberto\n");
  exit(1);
   }
   do {
      printf("Digite string (CR p/ terminar):\n");
      gets(str);
```



```
      strcat(str,"\n");
      fputs(str, fp);
    } while(*str!='\n');
}
```

9.31 fread(*str, size, count, FILE *fp)

Lê *count* número de objetos, cada um com um tamanho de *size* da stream apontada e coloca-os na matriz *str*. O indicador de posição é adiantado o número de bytes lidos. Devolve o número real de itens lidos. Caso sejam menos do que o solicitado, então ocorreu um erro ou chegou-se ao fim de arquivo. Deve-se utilizar `feof()` ou `ferror()` para saber o que aconteceu. Exige `#include<stdio.h>`. Funções relacionadas: `fwrite()`, `fopen()`, `fscanf()`, `fgetc()` e `getc()`.

9.32 fscanf(FILE *fp, controle, campos)

Lê como se fosse o scanf, mas ao invés de ler em `STDIN` busca os dados no arquivo mencionado. Veja mais detalhes em `scanf()`. Exige `#include<stdio.h>`. Funções relacionadas: `scanf()` e `fprintf()`.

```
char st[80];
float f;
fscanf(fp, "%s%f", str, &f);
```

9.33 fseek(FILE *fp, long offset, int origin)

Coloca o indicador de posição do arquivo de acordo com $offset$ e *origin*. O primeiro ($offset$) indica o número de bytes a partir de *origin* que pode ser (definidos em `STDIO.H`). `SEEK_SET` move a partir do início do arquivo; `SEEK_CUR` move a partir da posição atual; `SEEK_END` move a partir do fim do arquivo; A devolução de um zero significa que deu certo, algo diferente de zero sinaliza um erro. Pode-se avançar além do final do arquivo, mas é um erro avançar para antes do início. Exige `#include<stdio.h>`. Funções relacionadas: `ftell()`, `rewind()`, `fopen()`, `fgetpos()` e `fsetpos()`.

```
struct endere {
   char nome[30];
   char rua[40];
   int nume;
   } cadastro

void find(long client_num){
   FILE *fp;
   ... open do arquivo ...
   fseek(fp,
    client_num*sizeof(struct endere), SEEK_SET);
   fread(&cadastro, sizeof(struct endere), 1, fp);
   fclose(fp);
}
```

9.34 fsetpos(FILE *fp, *posicao)

Move o indicador de posição para o ponto especificado em *posicao* que deve ter sido salvada antes via `fgetpos()`. O tipo de *posicao* é definido em `STDIO.H` e seu nome é `fpos_t`. Se der certo, volta zero. Exige `#include<stdio.h>`. Funções relacionadas: `fgetpos()`, `fseek()`, `ftell()`.

```
fsetpos(fp, &file_loc);
```

9.35 ftell(FILE *fp)

Devolve o indicador de posição em um longo. Para streams binários é o exato número de bytes desde o início do arquivo. Para streams de texto pode não ser por conta das traduções. Devolve -1L quando ocorre um erro. Exige `#include<stdio.h>`. Funções relacionadas: `fgetpos()`, `fseek()`.

```
long i;
if((i=ftell(fp))==-1L)
   printf("Erro no arquivozn");
```

9.36 fwrite(*str, size, count, FILE *fp)

Escreve *count* número de objetos, cada um com um tamanho de *size* da stream apontada a partir da matriz *str*. O indicador de posição é adiantado o número de bytes lidos. Devolve o número real de itens escritos. Caso sejam menos do que o solicitado, então ocorreu um erro ou chegou-se ao fim de arquivo. Deve-se utilizar `feof()` ou `ferror()` para saber o que aconteceu. Exige `#include<stdio.h>`. Funções relacionadas: `fread()`, `fscanf()`, `fgetc()` e `getc()`.

9.37 getc(FILE *fp)

Lê um caracter no arquivo fp. Se bem sucedida ela devolve o caractere lido de um arquivo aberto como leitura. Quando o arquivo acabar, devolve `EOF`. Também devolve `EOF` quando ocorrer um erro. Exige `#include<stdio.h>`.

```
do {
   ch = getc(fp);
} while (ch != EOF);
```

9.38 getch()

Lê um caracter de `STDIN`, mas não o mostra na tela. Exige `#include<conio.h>`.

```
do {
   ch = getch(fp);
} while (ch != EOF);
```

9.39 getchar()

Lê um caracter de `STDIN`. Exige `#include<stdio.h>`.

9.40 gets(*str)

Lê caracteres de `STDIN` e coloca-os em *str*. Devolve um endereço para *str* ou apontador nulo em caso de erro. Exige `#include<stdio.h>`.

9.41 isalnum(ch)

Devolve verdadeiro se ch é uma letra ou um dígito. Em inglês, A..Z, a..z ou 0..9. Exige `#include<ctype.h>`.

```
#include<ctype.h>
void main(void) {
   char ch;
   for (;;) {
      ch = getc(stdin);
  if (ch==' ') break;
  if (isalnum(ch)) printf("%c e alfanum\n",ch);
```



```
}
}
```

9.42 isalpha(ch)

Devolve verdadeiro se ch é uma letra. Em inglês, A..Z ou a..z. Exige `#include<ctype.h>`.

9.43 iscntrl(ch)

Devolve verdadeiro se ch está entre 0 e 0x1F ou se é igual a 0x7F (del). Exige `#include<ctype.h>`.

9.44 isdigit(ch)

Devolve verdadeiro se ch é um dígito, ou seja 0..9. Exige `#include<ctype.h>`.

9.45 isgraph(ch)

Devolve verdadeiro se ch é caracter que pode ser impresso, com exceção do espaço. Geralmente na faixa de 0x21 a 0x7E. Exige `#include<ctype.h>`.

9.46 islower(ch)

Devolve verdadeiro se ch é uma letra minúscula. Em inglês, a..z. Exige `#include<ctype.h>`.

9.47 isprint(ch)

Devolve verdadeiro se ch é caracter que pode ser impresso, incluindo um espaço. Geralmente na faixa de 0x20 a 0x7E. Exige `#include<ctype.h>`.

9.48 ispunct(ch)

Devolve verdadeiro se ch é caracter de pontuação (isto é pode ser impresso, mas não é letra nem número). Exige `#include<ctype.h>`.

9.49 isspace(ch)

Devolve verdadeiro se ch é espaço, tabulação horizontal ou vertical, form feed, lin feed ou carriage return. Exige `#include<ctype.h>`.

9.50 isupper(ch)

Devolve verdadeiro se ch é uma letra maiúscula. Em inglês, A..Z. Exige `#include<ctype.h>`.

9.51 isxdigit(ch)

Devolve verdadeiro se ch é um dígito hexadecimal. Ou seja, A..F, a..f, 0..9 Exige `#include<ctype.h>`.

9.52 itoa(int num, *str, int base)

Converte a variável inteira *num* para uma string apontada por *str*, na base *base*. Exige `#include<stdlib.h>`. Funções relacionadas: `atoi()` e `sscanf()`.

```
#include<stdio.h>
#include<stdlib.h>
main() {
   char p[20];
   itoa(8826, p, 16);
   printf(p);  // escreve o hexadecimal 227A
}
```

9.53 labs(long num)

Devolve o valor absoluto de *num*. Devolve um long Exige `#include<stdlib.h>`. Funções relacionadas: `abs()`.

9.54 log(double arg)

Devolve o logaritmo natural (neperiano, de base e) de arg. Devolve um double. Se arg é negativo ocorre erro de domínio e se arg é zero ocorre erro de escala. Exige `#include<math.h>`. Funções relacionadas: `ceil()`. A proposito: $2.71^{2.30} \approx 10$

```
#include<stdio.h>
#include<math.h>
main() {
   double val = 1.0;
   do {
      printf("arg= %f, log = %f\n", val, log(val));
      val = val + 1;
    } while (val <= 11.0);
}
```

9.55 log10(double arg)

Devolve o logaritmo de base 10 de arg. Devolve um double. Se arg é negativo ocorre erro de domínio e se arg é zero ocorre erro de escala. Exige `#include<math.h>`. Funções relacionadas: `log()`.

```
#include<stdio.h>
#include<math.h>
main() {
   double val = 1.0;
   do {
      printf("arg= %f, log = %f\n", val, log10(val));
      val = val + 1;
    } while (val <= 11.0);
}
```

9.56 ltoa(long num, *str, int base)

Converte a variável longa *num* para uma string apontada por *str*, na base *base*. Exige `#include<stdlib.h>`. Funções relacionadas: `itoa()`.

```
#include<stdlib.h>
#include<stdio.h>
main() {
   char p[20];
   ltoa(8826L, p, 16);
   printf(p);  // escreve o hexadecimal 227A
}
```


9.57 memchr(*buff, int ch, int count)

Procura em *buff a primeira ocorrência de ch, nos primeiros count caracteres. Devolve um ponteiro ou um ponteiro nulo. Exige `#include<string.h>`.

```
#include<stdio.h>
#include<string.h>
void main(void) {
   char *p;
   p = memchr("Vamos ver se esta coisa",'v',14);
   printf(p); // imprime ver se esta...
}
```

9.58 memcmp(*arg1, *arg2, int count)

Compara os primeiros `count` caracteres de `arg1` e de `arg2`. Retorna um valor negativo se $arg1 < arg2$, zero se eles são iguais e um valor positivo se $arg1 > arg2$. Exige `#include<string.h>`.

```
#include<stdio.h>
#include<string.h>
void main(int ac, char *av[]) {
   int outc, tam, l1, l2;
   if (ac != 3) {
      print("numero errado de parametros\n");
  exit(1);
   }
   l1 = strlen(av[1]);
   l2 = strlen(av[2]);
   tam = l1 < l2 ? l1 : l2;
   outc = memcmp (av[1], av[2], tam);
   if (!outc) printf("iguais\n");
   else if(outc<0) printf("primeiro menor\n");
        else printf("segundo menor\n");
}
```

9.59 memcpy(*to, *from, int tam)

Copia tam caracteres de *from para *to. Devolve um ponteiro para *to. Se os parametros se sobrepõem seu comportamente é indefinido. Exige `#include<string.h>`.

```
#include<stdio.h>
#include<string.h>
#define TAM 80
void main(void) {
   char buf1[TAM], buf2[TAM];
   strcpy(buf1, "vamos ver se esta coisa...");
   memcpy(buf2, buf1, TAM);
   printf(buf2);
}
```

9.60 memmove(*to, *from, int tam)

Copia tam caracteres de *from para *to. Devolve um ponteiro para *to. Se os parametros se sobrepõem a cópia ocorrerá normalmente, *to ficará correto e *from será modificado. Exige `#include<string.h>`.

9.61 memset(*buf, int ch, int ctd)

Copia o byte menos significativo de ch nos primeiros ctd caracteres de *buf. Devolve um apontador para *buf. Exige `#include<string.h>`.

```
char buf[1000];
memset(buf,'\0',100);
memset(buf,'X',10);
printf(buf);
```

9.62 modf(double num, double *i)

Decompõe o número *num* em suas partes inteira e fracionária. Devolve a parte fracionário e a parte inteira vai para a variável apontada por *i*. Exige `#include<math.h>`. Funções relacionadas: `frexp()` e `ldexp()`.

```
double i;
double f;
f =  modf(54.129, &i);
printf("%f %f",i,f);   // imprime 54.000 e 0.129
```

9.63 pow(double base, double expoente)

Calcula e devolve a base elevada ao expoente. Se base é zero e expoente é negativo ou zero ocorre erro de domínio. Idem se a base é negativa e expoente não é inteiro. Se der estouro ocorre erro de escala. Exige `#include<math.h>`. Funções relacionadas: `exp()` e `log()` e `sqrt()`.

9.64 printf

A função printf gera saída formatada na console. Seu formato

```
int printf ("string de controle", variáveis,...);
```

Os códigos que fazem parte da string de controle são:

%c	um único caractere
%d	inteiro com sinal
%i	inteiro com sinal
%e	notação exponencial (e minúsculo)
%E	notação exponencial (E maiúsculo)
%f	ponto flutuante
%g	usa %e ou %f, o que for mais curto
%G	usa %E ou %f, o que for mais curto
%o	octal sem sinal
%s	string terminado por \0
%u	inteiro sem sinal
%x	hexadecimal sem sinal (a, b, ...)
%X	hexadecimal sem sinal (A, B, ...)
%p	ponteiro
%%	o caractere %

Um número colocado entre o % e a letra indicadora diz ao C para mostrar no mínimo essa quantidade de caracteres. Assim, se necessário o C inclui brancos no campo. Se o preenchimento for com zeros, colocar um zero antes desse número.
Se houver um especificador de precisão, deve-se colocá-lo após o preenchimento, separando-o por um ponto. Se este número for negativo haverá uma justificação à esquerda (já que o padrão é alinhamento à direita).

```
int i = 23;
printf("%5d",i); // imprime ///23
printf("%05d",i); // imprime 00023
float p = 3.4;
printf("%5.2f",p); // imprime /3.40
printf("%-5.2f",p); // imprime 3.40/
```

Nos exemplos acima, por razões óbvias usou-se uma \ para indicar um espaço em branco.

9.65 putc(char c, FILE *fp)

Escreve o caracter c no arquivo fp. Se bem sucedida ela devolve o caractere gravado. Se mal sucedida, devolve `EOF`. Exige `#include<stdio.h>`.

```
#include<stdio.h>
main() {
FILE *fp;
fp = fopen("c:/tcc/alfa.zzz","w");
putc('a',fp);
fclose(fp);
}
```

9.66 rand()

Gera uma sequência de números pseudo-randômicos. Cada vez que ela é chamada volta um número entre zero e `RAND_MAX`. Exige `#include<stdlib.h>`. Função relacionada: `srand()`.

```
#include<stdlib.h>
#include<stdio.h>
void main(void) {
   int i;
   for (i=0; i<10; i++) {
      printf("%d ",rand());
}  // sao impressos 41 18467 6334 26500
   // 19169 15724 11478 29358 26962 24464
}
```

9.67 remove(*nome)

Apaga o arquivo especificado por *nome*. Devolve zero se for bem sucedido e diferente de zero senão. Exige `#include<stdio.h>`. Função relacionada: `rename()`.

```
#include<stdio.h>
main(int ac, char *av[]) {
   if(remove(av[1])) printf("Erro de exclusao\n");
}
```

9.68 rename(*velho, *novo)

Altera o nome do arquivo de *velho* para *novo*. Não deve haver nenhum nome igual a *novo* no diretório atual. Devolve zero se for bem sucedido e diferente de zero senão. Exige `#include<stdio.h>`. Função relacionada: `remove()`.

```
#include<stdio.h>
main(int ac, char *av[]) {
   if(rename(av[1], av[2])!=0)
      printf("Erro de troca de nome\n");
}
```

9.69 rewind(FILE *fp)

Move o indicador de posição para o início do arquivo (ou stream). Limpa os indicadores de erro e de fim de arquivo. Não há valor de retorno. Exige `#include<stdio.h>`. Função relacionada: `fseek()`.

```
void le_duas_vezes(FILE *fp) {
   while(!feof(fp)) putchar(getc(fp));
   rewind(fp);
   while(!feof(fp)) putchar(getc(fp));
}
```

9.70 scanf

A função scanf, é usada para entrar com dados formatados a partir do teclado. seu formato é

```
int scanf("string de controle", &var1, &var2, ...);
```

Note que ao contrário de `printf`, em `scanf`, as variáveis são passadas através de seu endereço (por isso o símbolo & antes do nome das variáveis). Quanto à string de controle, os códigos são praticamente os mesmos de `printf`.

9.71 sin(double arg)

Devolve o seno de arg. Arg está em radianos. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas.

```
#include<stdio.h>
#include<math.h>
main() {
   double val = -1.0;
   do {
      printf("Arco=%f, sin=%f\n", val, sin(val));
      val = val + 0.1;
    } while (val <= 1.0);
}
```

9.72 sinh(double arg)

Devolve o seno hiperbólico de arg. Arg está em radianos. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas. Sua fórmula

$$senh(t) = \frac{e^t - e^{-t}}{2}$$

.

9.73 sprintf(*buf, controle, campos)

Idêntica a `printf` exceto que a saída é colocada na matriz *buf*. Veja a função `printf` para detalhes. Exige `#include<stdio.h>`. Função relacionada: `printf()` e `fsprintf()`.

```
char st[80];
sprintf(str, "%s %d %c","dez", 20, 30);
```

9.74 sqrt(double num)

Devolve a raiz quadrada de *num*. Retorna um double. Se chamar com negativo, dá um erro de domínio. Exige `#include<math.h>`. Funções relacionadas: `exp()`, `log()` e `pow()`.

```
printf("%f",sqrt(9));  // imprime 3.00000
```

9.75 srand(unsigned seed)

Estabelece um ponto de partida para a sequência de rand(). Exige `#include<stdlib.h>`. Função relacionada: `rand()`.

9.76 sscanf(*buf, controle, campos)

Idêntica a `scanf` exceto que os dados são lidos da matriz *buf*. Veja a função `scanf` para detalhes. Exige `#include<stdio.h>`. Função relacionada: `scanf()` e `fscanf()`.

```
#include<stdio.h>
main() {
   char st[80];
   int i;
   sscanf("Alo 1 2 3 4 5", "%s%d", st, &i);
   printf("%s %d",st, i);  // imprime Alo 1
}
```

9.77 strcat(*str1, *str2)

Concatena uma cópia de *str2 em *str1 e termina str1 com um nulo. O nulo que originalmente terminava str1 é sobreposto pelo primeiro caractere de str2. Str2 permanece inalterado na operação. Se eles se sobrepõem o resultado é indeterminado. A função devolve um pointer para str1. Não ocorre verificação de limites, ou seja, você deve garantir que str1 seja suficientemente grande para conter seu valor original mais o conteúdo de str2. Exige `#include<string.h>`. Funções relacionadas: `strchr()`, `strcmp()`, `strcpy()`.

```
#include<stdio.h>
#include<string.h>
void main(void) {
   char s1[80], s2[80];
   gets(s1);
   gets(s2);
   strcat(s2,s1);
   printf(s2);
}
```

9.78 strchr(*str, char ch)

Devolve um ponteiro para a primeira ocorrência de ch em str. Se não for encontrada nenhuma ocorrência volta um ponteiro nulo. Exige `#include<string.h>`. Funções relacionadas: `strpbrk()`, `strspn()`, `strsrt()`, `strok()`.

```
#include<stdio.h>
#include<string.h>
void main(void) {
   char *p;
   p = strchr("Isto e um teste",' ');
   printf(p); // imprime bebumbteste
}
```

9.79 strcmp(*str *str2)

Compara ambos e devolve negativo se $str1 < str2$, zero se iguais e positivo se $str1 > str2$. Exige `#include<string.h>`. Funções relacionadas: `strchr()`, `strcpy()`, `strncmp()`.

```
password(void) {
   char s[80];
   printf("digite a senha: ");
   gets(s);
   if (strcmp(s,"pass")) {
      printf("senha invalida\n");
  return 0;
}
    return 1;
}
```

9.80 strcoll(*str1, *str2)

Compara str1 com str2 de acordo com a localidade (veja setlocale()). Exige `#include<string.h>`. Funções relacionadas: `memcmp()`, `strcmp()`.

9.81 strcpy(*str1, *str2)

Copia str2 em str1. Str2 deve ser um ponteiro para uma string terminada por nulo. Devolve um ponteiro para str1. Se eles se sobrepõem o comportamente é indefinido. Str1 deve ser suficientemente grande. Exige `#include<string.h>`. Funções relacionadas: `memcpy()`, `strchr()`, `strcmp()` e `strncmp()`.

```
   char s[80];
   strcpy(s,"oba"); // copia oba em s
```

9.82 strcspn(*str1, *str2)

Devolve o comprimento da substring inicial de str1 formada apenas pelos caracteres inexistentes em str2, ou seja, devolve o indice do primeiro caractere de str1 que coincide com qualquer caractere de str2. Exige `#include<stdio.h>`. Funções relacionadas: `srtbrk()`, `strrchr()`, `strstr()` e `strok()`.

```
#include<stdio.h>
void main(void) {
   int onde;
   onde = strcspn("isto e um teste","uz");
   printf("%d",onde); // imprime 7
}
```

9.83 strerror(int num)

Devolve um ponteiro para a mensagem num, definida pela implementação. Exige `#include<string.h>`.

9.84 strlen(*str)

Devolve o comprimento de str, desconsiderando o nulo. Exige `#include<string.h>`. Funções relacionadas: `memcpy()`, `strchr()`, `strcmp()` e `strncmp()`.

```
printf("%d",strlen("oba")); // imprime 3
```

9.85 strncat(*str1, *str2, int tam)

Concatena não mais que tam caracteres de *str2 em *str1 e termina str1 com um nulo. O nulo que originalmente terminava str1 é sobreposto pelo primeiro caractere de str2. Str2 permanece inalterado na operação. Se eles se sobrepõem o resultado é indeterminado. A função devolve um pointer para str1. Não ocorre verificação de limites, ou seja, você deve garantir que str1 seja suficientemente grande para conter seu valor original mais o conteúdo de str2. Devolve str1. Exige `#include<string.h>`. Funções relacionadas: `strcat()`, `strnchr()`, `strncmp()`, `strncpy()`.


```
#include<stdio.h>
#include<string.h>
void main(void) {
   char s1[80], s2[80];
   int tam;
   gets(s1);  // supondo "oba"
   gets(s2);  // supondo "pizza"
   tam = 79-strlen(s2);
   strncat(s2,s1,tam);
   printf(s2); // "pizzaoba"
}
```

9.86 strncmp(*str, *str2, int tam)

Compara não mais do que tam caracteres de ambos e devolve negativo se $str1 < str2$, zero se iguais e positivo se $str1 > str2$. Se há menos do que tam caracteres em uma das strings, a comparação termina quando o primeiro nulo é encontrado; Exige `#include<string.h>`. Funções relacionadas: `strnchr()`, `strncpy()`, `strcmp()`.

```
#incluide<stdio.h>
#include<string.h>
void main(int ac, char *av[]) {
   if (ac != 3) {
      printf("numero incorreto\n")
  exit(1);
}
   if (strncmp(av[1], av[2], 8))
      printf("arquivos sao iguais\n");
}
```

9.87 strncpy(*str1, *str2, int tam)

Copia até tam caracteres de str2 em str1. Str2 deve ser um ponteiro para uma string terminada por nulo. Devolve um ponteiro para str1. Se eles se sobrepõem o comportamente é indefinido. Str1 deve ser suficientemente grande. Se str2 é maior do que tam, a string resultante, apontada por str1 não termina em nulo. Exige `#include<string.h>`. Funções relacionadas: `memcpy()`, `strchr()`, `strncat()` e `strncmp()`.

```
   char s1[128], s2[80];
   strncpy(s2, s1, 79); // copia no
           // máximo 79 de s1 em s2
```

9.88 strpbrk(*str1, *str2)

Devolve um ponteiro para o primeiro caractere de str1 que coincide com qualquer caractere de str2. Os nulos não são incluídos. Se não ocorre coincidência volta um nulo. Exige `#include<string.h>`. Funções relacionadas: `strrchr()`, `strspn()`, `strstr()` e `strok()`.

```
#include<stdio.h>
#include<string.h>
void main(void) {
   char *p;
   p = strpbrk("isto e um teste","obkj");
   printf(p);  // imprime "o e um teste"
}
```

9.89 strrchr(*str, int ch)

Devolve um ponteiro para a última ocorrência de ch na string apontada por str. Se não há, volta um ponteiro nulo. Exige `#include<string.h>`. Funções relacionadas: `strpbrk()`, `strspn()`, `strstr()`, `strtok()`.

```
#include<stdio.h>
#include<string.h>
main() {
   char *p;
   p = strrchr("vamos ver se a coisa vai...,'o');
   printf(p);   // imprime oisa vai...
}
```

9.90 strspn(*str1, *str2)

Devolve o índice de str1 que aponta para o primeio caractere de str1 que não está em str2. Exige `#include<string.h>`. Funções relacionadas: `strpbrk()`, `strchr()`, `strstr()`, `strtok()`.

```
#include<stdio.h>
#include<string.h>
main() {
   int len;
   len = strspn("isto e um teste","otsi ");
   printf("%d",len);   // imprime 5
}
```

9.91 strstr(*str1, *str2)

Devolve o índice de str1 que aponta para a ocorrência de str2 em str1. Exige `#include<string.h>`. Funções relacionadas: `strchr()`, `strcspn()`, `strpbrk()`, `strrchr()` `strspn()`, `strtok()`,.

```
#include<stdio.h>
#include<string.h>
main() {
   char *p;
   p = strstr("isto e um teste","um ");
   printf(p);   // imprime um teste
}
```

9.92 strtod(*str, **fim)

Converte um string em um double. Exige `#include<stdlib.h>`. Função relacionada: `atof()`.

```
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>
void main(void) {
   char *end, *sta = "208.00 uvas e 303.00 pes";
   end = sta;
   while (*sta) {
      printf("%f, ",strtod(sta,&end));
  printf("sobra: %s\n",end);
  sta=end;
  while(!isdigit(*sta) && *sta) sta++;
  } // deu 208.000000, sobra:  uvas e 303.00 pes
}       // deu 303.000000, sobra:  pes
```


9.93 strtok(*str1, *str2)

Retorna um ponteiro para a próxima palavra na string apontada por str1. Os caracteres de str2 definem os delimitadores que separam cada palavra da seguinte. Na primeira vez que strtok é chamada, str1 é realmente utilizada. Chamadas subsequentes devem usar um ponteiro nulo como primeiro argumento. A função strtok altera o argumento str1. Toda vez que uma palavra é encontrada é colocado um '\0' onde o delimitador foi encontrado. Assim, `strtok()` pode avançar pela string. Nada impede o uso de delimitadores diferentes a cada chamada. Exige `#include<string.h>`. Funções relacionadas: `strchr()`, `strcspn()`, `strpbrk()`, `strrchr()` `strspn()`.

```
#include<stdio.h>
#include<string.h>
main() {
   char *p;
   p = strtok("isto e um teste, mas nunca", " ");
   printf(p);
   do {
      p = strtok('\0', ", ");
  if(p) printf("-%s",p); //isto-e-um-teste-mas-nunca
} while (p);
}
```

9.94 strtol(*str, **fim, int base)

Converte um string em um long. Base deve estar entre 2 e 36 ou ser zero. Exige `#include<stdlib.h>`. Função relacionada: `atol()`.

```
#include<stdlib.h>
#include<stdio.h>
long main(void) {
   char sta[80], *end;
   printf("Digite um numero: ");//digitando 123.45
   gets(sta);
   printf("%ld \n",strtol(sta,&end,10));//volta 123
}
}
```

9.95 strtoul(*str, **fim, int base)

Converte um string em um unsigned long. Base deve estar entre 2 e 36 ou ser zero. Exige `#include<stdlib.h>`. Função relacionada: `strtol()`.

9.96 tan(double arg)

Devolve a tangente de arg. Arg está em radianos. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas.

```
#include<stdio.h>
#include<math.h>
main() {
   double val = -1.0;
   do {
      printf("Arco=%f, tan=%f\n", val, tan(val));
      val = val + 0.1;
    } while (val <= 1.0);
}
```

9.97 tanh(double arg)

Devolve a tangente hiperbólica de arg. Arg está em radianos. Exige `#include<math.h>`. Funções relacionadas: todas as trigonométricas. Sua fórmula

$$tanh(t) = \frac{senh(t)}{cosh(t)}$$

ou

$$tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$$

.

9.98 tolower(char ch)

Devolve ch em minúsculo, se ch for uma letra. Senão, devolve inalterado. Exige `#include<ctype.h>`. Função relacionada: `toupper()`.

```
putchar(tolower('Z')); // mostra z
```

9.99 toupper(char ch)

Devolve ch em maiúsculo, se ch for uma letra. Senão, devolve inalterado. Exige `#include<ctype.h>`. Função relacionada: `tolower()`.

```
putchar(toupper('b')); // mostra B
```

10 Alguns programas selecionados

10.1 passeio do cavalo

Recursos: recursividade, backtracking.

```
#include<stdlib.h>
#include<stdio.h>
int MAT[9][9];
int DLIN[9]={0,-2,-1,1,2,2,1,-1,-2};
int DCOL[9]={0,1,2,2,1,-1,-2,-2,-1};
int MAIOR=-99999;
int andacavalo (int NUMMOV, int LCAV, int CCAV) {
  int LP,CP,RESP,K;
  K=1;
  do {
    RESP=0;
    LP = LCAV+DLIN[K];
    CP = CCAV + DCOL[K];
    if ((LP>=1)&&(LP<=8)&&(CP>=1)&&(CP<=8)){
      if (MAT[LP][CP]==0) {
        MAT[LP][CP]=NUMMOV;
        if (NUMMOV < 56) {  //colocar aqui 56 ou 63 (era 64???)
          RESP = andacavalo ((NUMMOV+1),LP,CP);
          if (RESP == 0) {
            MAT[LP][CP] = 0;
          }
        }
        else {
          RESP = 1;
        }
      }
    }
    K++;
  } while ((K<=8) && (RESP==0));
  return RESP;
}

main() {
  int I;
  int J;
  int LI,CI;
  for (I=0;I<9;I++) {
    for (J=0;J<9;J++) {
      MAT[I][J]=0;
    }
  }
```



```
  LI = 6; // jogada inicial (pegar da folha)
  CI = 6; // jogada inicial (pegar da folha)
  MAT[LI][CI] = 1;
  if (andacavalo (2,LI,CI)==1) {
    for (I=1;I<9;I++) {
      for (J=1;J<9;J++) {
        printf("%3i  ",MAT[I][J]);
      }
      printf("\n");
    }
  }
  else {
    printf ("Nao e possivel \n");
  }
}
```

10.2 TEA

Algoritmo de criptografia simétrico. Retirado do livro do Couloris, pág. 258

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <io.h>
#include <string.h>
void encript(unsigned long k[], unsigned long text[]) {
  unsigned long y = text[0], z=text[1];
  unsigned long delta = 0x9e3779b9, sum =0; int n;
  for (n=0; n<32; n++) {
    sum += delta;
    y += ((z<< 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);
    z += ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
//  printf ("%4X %4X - %4X %4X\n",ya,y,za,z);
  }
  text[0] = y; text[1]=z;
}
void decript (unsigned long k[], unsigned long text[]) {
  unsigned long y = text[0], z=text[1];
  unsigned long delta = 0x9e3779b9, sum =delta << 5; int n;
  for (n=0; n<32; n++) {
    z -= ((y<< 4) + k[2]) ^ (y+sum) ^ ((y >> 5) + k[3]);
    y -= ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
    sum -= delta;
  }
  text[0] = y; text[1]=z;
}
void main(ac,av)
  int ac;
  char *av[]; {
  FILE *infile;
  FILE *outfile;
  FILE *senha;
  int tipo, i,j ;
  unsigned char Text[8];   // nao tinha o unsigned
  unsigned long  int k[4];
  if (ac != 5) {
printf("Erro: a chamada certa: tea 1/2 arq-ent arq-sai
     arq-senha - P.Kantek 08/2010\n");
exit(1); }
  else {tipo = atoi(av[1]);    // qual algoritmo ?
if (tipo == 1) {printf("Criptografando...\n");}
else {printf("Decriptografando...\n");} }
    if ((outfile = fopen(av[3],"wb")) == NULL)   {
      printf("Erro na criacao da saida\n");
      exit(1); }
    if ((senha = fopen (av[4],"rb"))==NULL) {
      printf("Erro: nao foi encontrado o arquivo senha \n");
  exit(1); }
    i = fread(k,8,4,senha);
    if ((infile = fopen (av[2],"rb"))==NULL) {
      printf("Erro: nao foi encontrado o arquivo entrada \n");
  exit(1);  }
   //    encript(k, (unsigned long*) Text);
    while (!feof(infile)) {
    i = fread(Text,1,8,infile);
    if (i<=0) break;
    while (i<8){Text[i++]=' ';}
    switch (tipo) {
      case 1: encript(k,(unsigned long *) Text); break;
      case 2: decript(k,(unsigned long *) Text); break;   }
    j = fwrite(Text,1,8,outfile);
  }
  i= fclose(infile);
  i= fclose(senha);
  i= fclose(outfile);
}
```

10.3 War

programa tirado do livro de maratonas em espanhol.

```
#include<stdio.h>
#define NCARDS 52
#define NSUITS 4
#define QUEUESIZE 300
#define TRUE 1
#define FALSE 0
#define MAXSTEPS  1000000
typedef struct {
   int q[QUEUESIZE+1];
   int first;
   int last;
   int count;
} queue;
init_queue(queue *q) {
   q->first = 0;
   q->last = QUEUESIZE-1;
   q->count = 0;
}
enqueue(queue *q, int x) {
   if (q->count >= QUEUESIZE)
      printf("fila cheia\n");
   else {
      q->last = (q->last+1) % QUEUESIZE;
      q->q[q->last] = x;
      q->count = q->count + 1;
   }
}
int dequeue(queue *q) {
   int x;
   if (q->count <= 0 ) printf("fila vazia\n");
   else {
      x = q->q[q->first];
      q->first = (q->first + 1)% QUEUESIZE;
      q->count = q->count - 1;
   }
   return (x);
}
int empty(queue *q) {
   if (q->count <= 0) return (TRUE);
     else return(FALSE);
     }
char values[]="23456789TJQKA";
char suits[] = "cdhs";
int rank_card(char value, char suit) {
   int i,j;
   for (i=0;i<(NCARDS/NSUITS);i++)
      if (values[i]==value)
         for (j=0;j<NSUITS;j++)
            if (suits[j]==suit)
               return(i*NSUITS+j);
   printf("carta errada\n");
}
char suit(int card){
   return(suits[card%NSUITS]);
}
char value(int card){
   return(values[card/NSUITS]);
}
main() {
   queue decks[2];
   char value,suit,c;
   int i;
   printf ("entrando\n");
   while (TRUE) {
      for (i=0;i<=1;i++) {
         init_queue(&decks[i]);
         while ((c=getchar()) != '\n') {
            if (c == EOF) return;
```



```
            if (c != ' ' ) {
               value = c;
               suit = getchar();
               enqueue(&decks[i],rank_card(value,suit));
            }
         }
      }
      printf("armazenou\n");
      war(&decks[0],&decks[1]);
   }
}
war(queue *a, queue *b) {
   int steps=0;
   int x,y;
   queue c;
   int inwar;
   inwar = FALSE;
   init_queue(&c);
   while ((!empty(a)) && (!empty(b) && (steps < MAXSTEPS))) {
      steps=steps+1;
      x=dequeue(a);
      y=dequeue(b);
      enqueue(&c,x);
      enqueue(&c,y);
      if (inwar) {
         inwar=FALSE; }
      else {
         if (value(x)>value(y)) {
            clear_queue(&c,a); }
         else  if  (value(x)<value(y)) {
            clear_queue(&c,b); }
         else  if (value(y) == value(x)) {
            inwar=TRUE; }
       }
   }
   if (!empty(a) && empty(b))
      printf("a ganha em %i etapas \n",steps);
   else if (empty(a) && !empty(b))
      printf("b ganha em %i etapas \n",steps);
   else if (!empty(a) && !empty(b))
      printf("empatou apos %i etapas\n",steps);
   else
      printf("a e b empataram apos %i etapas\n",steps);
}

clear_queue(queue *a, queue *b) {
   while (!empty(a)) enqueue(b,dequeue(a));
   }
```

10.4 Mínimos quadrados

Retirado da folha vivo245, é o método dos mínimos quadrados

```
#include<stdio.h>
#include<stdlib.h>
float J,K,L,M,N,X,Y;
int IND,CT;
float VX[1000];
float VY[1000];
float A,B;
int main() {
printf("Informe o valor de X \n");
scanf("%f",&X);
CT = 0;
while (X != -1) {
VX[CT]=X;
printf ("X=%f e Y equivalente \n",X);
scanf("%f",&Y);
VY[CT]=Y;
printf("Y=%f e Novo X ou -1 para encerrar\n",Y);
scanf("%f",&X);
CT++;
}
printf("CT=%i\n",CT);
    J = 0;
    K = 0;
    L = 0;
    M = 0;
    N = 0;
    for (IND=0;IND<=CT;IND++) {
        J =J + VX[IND];
    K = K + VY[IND];
L = L + (VX[IND] * VX[IND]);
M = M + (VX[IND] * VY[IND]);
N++;
}
N--;
printf("J=%f\n",J);
printf("K=%f\n",K);
printf("L=%f\n",L);
printf("M=%f\n",M);
printf("N=%f\n",N);
B = ((K*J) - (N*M))/((J*J)-(L*N));
A = (K-(J*B))/N;
printf("A=%f      B=%f\n",A,B);
printf("Novo X ou -1 para encerrar\n");
scanf("%f",&X);
while (X != -1) {
Y = A + (B*X);
printf ("Par X=%f, Y=%f\n",X,Y);
printf(" - Novo X ou -1 para encerrar\n");
scanf("%f",&X);
}
}
```

10.5 Mochila

Problema da mochila binária (folha vivo240).

```
#include<stdio.h>
#include<stdlib.h>
main(){
   int w[13]={0,5,3,9,10,1,3,6,4,1,1,4,3};
   int v[13]={0,6,8,4,22,20,15,12,4,19,11,15,23};
//   int w[9]={0,1,5,1,1,6,6,8,8};   exemplo do pe da pagina
//   int v[13]={0,2,21,3,13,13,11,19,4}; idem
   int t[13][101];  //colunas igual a no minimo W.
   //   Pode colocar um valor bem grande...
   int x[13]={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
   int n=12;
   int W=27;
   int Y,i,A,B;
   for(Y=0;Y<=W;Y++){
      t[0][Y]=0;
      for (i=1;i<=n;i++){
         A = t[i-1][Y];
         if (w[i]>Y) {
            B = 0;
         }
         else {
            B=t[i-1][Y-w[i]]+v[i];
         }
         if (A>=B) {
            t[i][Y]=A;
         }
         else {
            t[i][Y]=B;
         }
      }
   }
   Y = W;
   for (i=n;i>=1;i--){
      if (t[i][Y]==t[i-1][Y]) {
         x[i]=0;
      }
      else {
         x[i]=1;
         Y=Y-w[i];
      }
   }
   for (i=1;i<=n;i++){
      printf("%i ",x[i]);
   }
   printf("\n ====>%i", t[n][W]);
}
```

```
A
    abs, 6
    acos, 7
```


