Entrada e Saída

Prof. Edson Pedro Ferlin

Comandos do Pré-processador
Comando #define

- Usamos o comando **#define** para definir um identificador e uma string. O compilador substituirá o identificador pela string toda vez que ele for encontrado no arquivo fonte.
- **O identificador é chamado de nome de macro**, e o **processo de substituição é chamado de substituição de macro**.
- Forma geral: **#define identificador string**

Comandos do Pré-processador
Comando #define

- Exemplos :

```
#define verdadeiro 1
#define Max_pix 256
#define um 1
#define erro_1 "erro padrão na entrada\n"
#define PRN(n) printf("%f\n",n)
#define SOMA(x,y) (x + y)
#define Produto(x,y) ((x)*(y))
#define min(a,b) (a<b)?a:b
```


Comandos do Pré-processador
Comando #include

- O comando ***#include*** do pré-processador instrui o compilador a incluir um outro arquivo fonte com aquele que contém o comando ***#include***. Devemos envolver o arquivo-fonte lido entre aspas ou entre o sinal de maior e menor.

- Exemplos : #include "stdio.h"
  #include <stdio.h>

- **#include "stdio.h"** indica que o compilador procurará o arquivo **stdio.h** no diretório corrente.

- **#include <stdio.h>** indica que o compilador procurará o arquivo **stdio.h** nos diretórios especificados na linha de comando da opção -I do compilador, se não encontrar o arquivo, procurará nos diretórios padrões. **O diretório de trabalho corrente não será pesquisado em momento algum.**

Arquivo Cabeçalho stdio.h
Entrada e Saída Bufferizado

- É o arquivo-cabeçalho de entrada e saída bufferizado do C e se chama **stdio.h** (*standard input/output*) que corresponde no UNIX ao arquivo **io.h**.

- **Fila de bytes** : o sistema de arquivos bufferizado do **C** transforma cada dispositivo físico em um dispositivo lógico chamado **fila de byte**. Todas as filas de bytes se comportam de maneira similar. Como estas filas são independentes do dispositivo, as mesmas funções que podem escrever em um arquivo em disco podem também escrever no vídeo. Existem dois tipos de filas de bytes:

  - **Filas de texto**: sequência de caracteres que está organizada em linhas que são encerradas por caracteres \n.

  - **Filas binárias**: sequência de bytes que tem correspondência unívoca (um por um) com os bytes do dispositivo externo. Portanto, nenhuma tradução de caracteres ocorre.

Arquivos
Conceito Lógico

- Em C, um arquivo é um conceito lógico que o sistema pode aplicar a qualquer coisa, desde arquivos em disco até terminais;

- Todas as filas de bytes são iguais, mas nem todos os arquivos são iguais;

- O computador fechará automaticamente todos os arquivos quando seu programa terminar normalmente por um **main()** retornando ao sistema operacional ou chamando um **exit()**. Quando da queda do sistema, o computador não fechará os arquivos;

- No início da execução de um programa, são abertas três filas de texto predefinidas: **stdin**, **stdout** e **stderr**, que se referem ao dispositivo de entrada e saída padrão que está conectado ao sistema (normalmente o console) e que podem ser redirecionadas para um outro dispositivo (stdio.h).

Funções Básicas
Entrada e Saída do Console

FUNÇÃO OPERAÇÃO

- **getchar()** **lê um caractere do teclado; espera por <enter>**
- **getche()** **lê um caractere com eco; não espera por <enter>**
- **getch()** **lê um caractere sem eco; não espera por <enter>**
- **putchar()** **escreve um caractere na tela**
- **gets()** **lê uma string do teclado**
- **puts()** **escreve uma string na tela**

Função printf()
Saída Formatada

- Um *comando de formatação* contém primeiro um sinal de (%) e, depois, o *código de formatação*.

%c um único caracter
%d decimal
%i decimal
%e notação científica
%f ponto decimal flutuante
%g usa %e ou %f - aquele que for menor
%o octal
%s string de caracteres
%u decimal sem sinal
%x hexadecimal
%% imprime o sinal %
%p exibe um ponteiro
%n o argumento associado será um ponteiro no qual será colocado o número de caracteres escritos até o momento.

Função printf()
Observações

- **Para especificar o número de casas decimais em números de ponto flutuante**, usamos o formato : **%w.df** onde **w** é o campo total do número que queremos imprimir e **d** representa o número de casas decimais. Por exemplo, **%10.5f , %12.6f , %6.0f e %7.2f**.

- **No caso dos strings**, a notação ***%x.ys, x representa o número mínimo e y o número máximo de caracteres a serem exibidos***. Por exemplo, **%6.12s**, exibirá uma string com pelo menos 6 caracteres e no máximo 12 caracteres. **Caso a string seja maior que a largura máxima do campo (y), o computador truncará os caracteres que estiverem além do limite especificado**.

Função scanf()
Entrada Formatada

- Colocando o sinal % na frente dos especificadores de formato de entrada, informamos a scanf() que o tipo de dado será lido a seguir.

%c lê um único caracter
%d lê um inteiro decimal
%i lê um inteiro decimal
%e lê um número de ponto flutuante
%flê um número de ponto flutuante
%h lê um inteiro curto
%o lê um número octal
%s lê uma string
%x lê um número hexadecimal
%p lê um ponteiro
%n recebe um valor inteiro igual ao número de caracteres lidos até o momento.

Função scanf()
Observações (1)

- 1. Um caractere branco na string de controle faz com que scanf() passe por cima de um ou mais caracteres brancos na string de entrada. Um caractere branco é um espaço, um tab ou um \n. Em resumo, um caractere branco na string de controle faz com que scanf() leia, mas não armazene, qualquer número (incluindo o zero) de caracteres brancos, até o primeiro caractere não-branco.
- 2. Um caractere não-branco na string de controle faz com que scanf() leia e desconsidere um caractere coincidente. Por exemplo, "%d,%d" faz com que scanf() primeiro leia um inteiro, depois leia e desconsidere uma vírgula e, finalmente, leia um outro inteiro.
- 3. Todas as variáveis usadas para receber valores através de scanf() devem ser passadas por seus endereços. Isto significa que todos os argumentos devem ser ponteiros para as variáveis usadas como argumentos.
- 4. No caso da leitura de strings para dentro de uma matriz de caracteres endereço, usamos: **scanf("%s",endereço);** pois neste caso endereço já é um ponteiro e não precisa ser precedido pelo operador &.

Função scanf()
Observações (2)

- 5. Devemos separar dados de entrada usando espaços, tabs ou \n. Sinais de pontuação como vírgula, ponto-e-vírgula e outros como separadores não são considerados.
- 6. Um * colocado depois de % e antes do código de formatação lerá dados do tipo especificado, mas suprimirá suas atribuições. Logo, o comando **scanf("%d%*c%d",&x,&y);** dada a entrada 10/20, coloca o valor 10 em x, desconsidera o sinal de divisão e atribui o valor 20 a y.
- 7. Os comandos de formatação podem especificar um modificador de comprimento máximo de campo. Esse modificador é um inteiro que colocamos entre o % e o código do comando de formatação que limita o número de caracteres lidos para qualquer campo. Por exemplo, para limitar a leitura de 20 caracteres para a string str, fazemos: **scanf("%20s", str);** caso a fila de entrada seja maior que 20 caracteres, então uma chamada subsequente à entrada começa onde essa chamada termina.

Função scanf()
Observações (3)

- 8. Apesar de usar espaços, tabs e \n como separadores de campo, quando estivermos lendo um único caracter, o computador lerá os separadores de campo como qualquer outro caracter. Por exemplo, com uma fila de entrada (**x y**), o **scanf("%c%c%c",&a, &b, &c);** retornará como caractere x em a, um espaço em b e o caractere y em c.

- 9. Qualquer outro caractere incluindo espaços, tabs e \n na string de controle, o computador usará esses caracteres para combinar e descartar caracteres da fila de entrada. O computador descartará qualquer caracter coincidente. Por exemplo, entrada (**10t20**), o **scanf("st%s", &x, &y);** atribuirá x=10 , y=20, e t será descartado. **scanf("%s",nome);** não retornará até que digitemos um caractere depois de digitar um caractere branco. Isto acontece porque o espaço depois de %s instrui scanf() para ler e desconsiderar espaços, tabs e \n.

- 10. Não podemos usar scanf() para exibir uma mensagem de aviso. Portanto, exiba todos os avisos explicitamente antes de chamar scanf().

Entrada e Saída Bufferizado
Funções mais utilizadas (stdio.h)

- **NOME FUNÇÃO**

fopen()	**abre uma fila**
fclose()	**fecha uma fila**
putc()	**grava um caractere na fila**
getc()	**ê um caractere da fila**
fseek()	**procura byte especificado em uma fila**
fprintf()	**é para uma fila o que printf() é para o console**
fscanf()	**é para uma fila o que scanf() é para o console**
feof()	**devolve verdadeiro se a marca de EOF for alcançada**
ferror()	**devolve verdadeiro se um erro tiver ocorrido**
rewind()	**restabelece o localizador de posição no início do arquivo.**
remove()	**apaga um arquivo**

Ponteiro do Arquivo

- É um ponteiro para a informação que **define vários aspectos do arquivo, incluindo seu nome, status e posição corrente**.
- **Um ponteiro de arquivo é uma variável do tipo FILE, que é definida em stdio.h.**

Funções com Arquivos
Função fopen() – Parte 1

- Esta função abre uma fila de bytes para ser usada e liga um arquivo a fila aberta.
- Forma geral: **FILE *fopen(char *_nome_arquivo_, char *_modo_);**
  - ***modo*** é uma string que contém o status de abertura desejado.
  - ***nome_arquivo*** é uma string que define o nome do arquivo.
- **Observação** : podemos abrir dois tipos de arquivos, um no **modo texto** e outro no **modo binário**.

Funções com Arquivos
Função fopen() – Parte 2

- **Modo Significado**
  - **"r" abre um arquivo-texto para leitura (read)**
  - **"w" cria um arquivo-texto para gravação (write)**
  - **"a" anexa a um arquivo-texto (append)**
  - **"rb" abre um arquivo-binário para leitura (read-binário)**
  - **"wb" cria um arquivo-binário para gravação (write-binário)**
  - **"ab" anexa um arquivo binário (append-binário)**
  - **"r+" abre um arquivo texto para leitura/gravação**
  - **"w+" cria um arquivo-texto para leitura/gravação**
  - **"a+" abre ou cria um arquivo-texto para leitura/gravação**
  - **"r+b" abre um arquivo-binário para leitura/gravação**
  - **"w+b" cria um arquivo-binário para leitura/gravação**
  - **"a+b" abre um arquivo binário para leitura/gravação**
  - **"rt" abre um arquivo texto para leitura (read-text)**
  - **"wt" cria um arquivo texto para gravação (write/text)**
  - **"at" anexa uma arquivo-texto (append-text)**
  - **"r+t" abre um arquivo-texto para leitura/gravação**
  - **"w+t" cria um arquivo-texto para leitura/gravação**
  - **"a+t" abre um arquivo-texto para leitura/gravação**

Funções com Arquivos
Função fopen() – Parte 3

- Exemplo : fp = fopen("A:\arq\teste.txt","w");

  fp é uma variável do tipo **FILE *,** que é o ponteiro do arquivo.

- A notação mais usual é escrita na forma :

```
if ( ( fp = fopen( "A:\arq\teste.txt","w") ) == NULL ) {
    puts("NÃO POSSO ABRIR O ARQUIVO");
    exit(1); }
```

- **Vantagem de detectar qualquer erro na abertura de um arquivo**, por exemplo : a tentativa de abrir um disco protegido contra gravação ou um disco cheio, antes de tentar gravar. Esse método usa **NULL**, que é zero, porque nenhum ponteiro de arquivo terá aquele valor. **NULL** é uma macro definida em stdio.h.

Funções com Arquivos
Função fopen() – Parte 4

- 1. A utilização de fopen() na abertura de um arquivo para gravação, criará um arquivo com o nome definido e se existir qualquer arquivo preexistente com este nome ele o apagará.
- 2. Quando queremos anexar outras informações ao final do arquivo, usamos o *modo a*. Quando usamos o *modo a* e o arquivo não existe, o computador devolverá um erro.
- 3. A abertura de um arquivo para operações de leitura, requer que o arquivo exista. Se não existir, fopen() devolverá um erro.
- 4. Quando abrimos um arquivo para as operações leitura/gravação, o computador não apagará se ele existir; e se ele não existir, o computador criará.

Funções com Arquivos
Função putc()

- É usada para ***gravar caracteres em uma fila*** previamente aberta para gravação através da função fopen().
- Forma geral: **int putc (int ch, FILE *fp);**

  fp é o ponteiro do arquivo devolvido por fopen();
  ch é o caractere a ser gravado (usa apenas um byte);

- Exemplo: putc(ch,fp);
- Observações:
  - 1. O ponteiro do arquivo informa a putc() em qual arquivo em disco gravar.
  - 2. Se uma gravação putc() for bem-sucedida, então ela devolverá o caractere gravado, caso contrário, devolverá um EOF(end-of-file).

Funções com Arquivos
Função getc() – Parte 1

- É usada para ***ler caracteres de uma fila*** que fopen() abriu no modo de leitura.
- Forma geral: **int getc(FILE *fp);**

  fp é um ponteiro de arquivo do tipo **FILE;**
  Devolve um inteiro, mas o byte de ordem superior é zero.
- Observações :

  1. A função getc() devolve uma marca EOF quando alcança o final de arquivo. **Logo, para ler um arquivo-texto até a marca de fim de arquivo**.

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


Funções com Arquivos
Função getc() – Parte 2

- 2. Quando **abrimos um arquivo para entrada binária, é possível ler um valor inteiro igual a marca EOF**. Se isto acontecer, a rotina que mostramos, indicará uma condição de fim de arquivo, embora não tenha atingido o fim físico do arquivo. Para resolver este problema, usamos a **função feof()**; que determina onde está a marca de fim de arquivo, quando da leitura de arquivos de dados binários. A função feof() aceita um argumento de ponteiros de arquivo; e retorna zero se não tiver chegado ao fim do arquivo.
- Exemplo: while (!feof(fp)) ch = getc(fp);
- Podemos aplicar este mesmo método a arquivos-textos.

Funções com Arquivos
Função fclose() – Parte 1

- É usada para ***fechar uma fila*** que foi aberta com fopen().
- Forma geral: **int fclose(FILE *fp);**

  fp é o ponteiro do arquivo devolvido pela fopen().
- Exemplo: fclose(fp);

Observações :

1. Um valor de retorno zero significa uma operação de fechamento bem-sucedida.

2. Podemos usar a função ferror() para determinar e relatar problemas.

3. Geralmente, fclose() fracassará somente quando removemos um disquete prematuramente ou quando não houver mais espaço no mesmo.

Funções com Arquivos
Função fclose() – Parte 2

Observações :

5. A função fclose() grava em um arquivo os dados que ainda permanecem no buffer do disco e efetua no arquivo um fechamento formal a nível do sistema operacional.

6. O fracasso em fechar uma fila é um convite a todos os tipos de problemas, incluindo perda de dados, destruição de arquivos e possíveis erros intermitentes no programa.

7. O uso de fclose() libera o bloco de controle do arquivo associado a fila e torna-o disponível para ser reutilizado.

8. O sistema operacional limita o número de arquivos abertos que podemos usar ao mesmo tempo, portanto podemos fechar um arquivo antes de poder abrir um outro.

Funções com Arquivos
Função ferror()

- É usada para ***determinar se uma operação de arquivo produziu um erro***.
- Forma geral: **int ferror(FILE *fp) ;**

  fp é o ponteiro do arquivo válido.
- Exemplo: ferror(fp);
- A função ferror() retorna 1 se houver ocorrido um erro durante a última operação de arquivo; devolve zero se a operação de arquivo foi bem-sucedida.

Funções com Arquivos
Função rewind()

- É usada para ***restabelecer o localizador de posição no arquivo para o início do arquivo*** especificado no seu argumento.
- Forma geral: **void rewind(FILE *fp);**

  fp é um ponteiro de um arquivo válido.
- Exemplo:

  rewind(fp);

Funções com Arquivos
Função putw() e getw()

- São usadas par ***ler e gravar inteiros de e para um arquivo em disco***. Estas funções não fazem parte do da padronização ANSI C. Elas trabalham exatamente como putc() e getc() só que em vez de ler ou gravar um único caracter, putw() e getw() lêem e gravam um inteiro.
- Forma geral: **int putw(int i, FILE *fp);**
- Forma geral: **int getw(FILE *fp);**
- Exemplo :

```
putw(100, fp);
inteiro=getw(fp);
```


Funções com Arquivos
Função fputs() e fgets()

- São usadas par ***ler e gravar filas a partir de strings***.
- Forma geral: **char fputs(char *_str_, FILE *fp);**
- Forma geral: **char fgets(char *_str_, int _tamanho_, FILE *fp);**
- A função **fputs() funciona como o puts(), só que ela escreve a string em um fluxo especificado**.
- A função **fgets() lê uma string de um fluxo especificado até que um caractere de nova linha seja lido ou tamanho-l caracteres tenham sido lidos**. Se uma nova linha é lida, ela fará parte da string (**ao contrário do gets()**). A string resultante terminará com nulo.

Funções com Arquivos
Função fread() e fwrite()

- São usadas para ***leitura e a escrita em blocos de dados.***
- Forma geral:

unsigned fread(void *_buffer_, int _num_bytes_, int _count_, FILE *_fp_);

unsigned fwrite(void *_buffer_, int _num_bytes_, int _count_, FILE *_fp_);

- No caso da função **fread()**, ***buffer*** é um ponteiro para uma região de memória que **receberá dados lidos do arquivo**. Para **fwrite()**, ***buffer*** é um ponteiro para a informação que será **escrita no arquivo. O número de bytes para ler ou escrever é especificado por *num_bytes***. O argumento ***count*** **determina quantos itens** (cada um tendo num_bytes de tamanho) **serão lidos ou escritos**. Finalmente, ***fp*** é um ponteiro para um arquivo de um fluxo previamente aberto.

Funções com Arquivos
Exemplo de Escrita em Arquivo

```
main()
{
  FILE *fp;
  float mat[100];
  int i;
  if( (fp=fopen("nome.dat","wb") ) ==NULL )
  { printf("Arquivo não pode ser aberto");
    exit(1);}
  for( i=0; i<100; i++) mat[i] = (float) i; /* salvar a matriz inteira */
  if( fwrite(mat, sizeof(mat),1,fp) != 1 )  printf("erro no arquivo");
  fclose(fp);
}
```

Funções com Arquivos
Exemplo de Leitura de Arquivo

```
main()
{
   FILE *fp;
   float mat[100];
   int i;
  if( (fp=fopen("nome.dat","rb") ) = =NULL )
  { printf("Arquivo não pode ser aberto");
  exit(1);}
  /* ler a matriz inteira */
  if( fread(mat, sizeof(mat),1,fp) != 1 )  printf("erro no arquivo");
  for( i=0; i<100; i++) printf("%03d ==> %05.1f\n",i, mat[i]);
  fclose(fp);
}
```

```
000 ==> 000.0
001 ==> 001.0
002 ==> 002.0
003 ==> 003.0
.....
095 ==> 095.0
096 ==> 096.0
097 ==> 097.0
098 ==> 098.0
099 ==> 099.0
```


Funções com Arquivos
Acesso Aleatório

- Podemos efetuar operações de leitura e gravação aleatórias sob o sistema de entrada e saída bufferizado com a ajuda da função fseek(), que ajusta o localizador de posição no arquivo.
- Forma geral: **int fseek(FILE **fp*, long int *num_bytes*, int *origem*);**

fp é um ponteiro para o arquivo retornado por uma chamada **fopen()**; ***um_bytes***, que é um inteiro longo, é o número de bytes a partir da ***origem***, necessários para se conseguir a posição corrente; e ***origem*** é uma das seguintes macros definidas em "stdio.h".

Origem	Nome	Valor atual
início do arquivo	SEEK_SET	0
posição corrente	SEEK_CUR	1
fim do arquivo	SEEK_END	2

- Para procurar ***num_bytes*** do **começo do arquivo**, ***origem*** deve ser indicada como **SEEK_SET**. Para procurar da **posição corrente**, usamos **SEEK_CUR**; do **final do arquivo**, usamos **SEEK_END**.

Funções com Arquivos
Acesso Aleatório - Exemplo

- O trecho de programa, mostra como ler o 235º byte do arquivo x.dat.

```
FILE *fp;
char ch;
if( ( fp = fopen("x.dat","rb") ) == NULL )
{
   printf("O arquivo não pode ser aberto\n");
   exit(1);
}
fseek(fp, 234, 0);
ch = getc(fp) ;  /* le um caractere na posição 235*/
```


Funções com Arquivos
Funções fprinf() e fscanf()

- Estas funções comportam-se exatamente como as funções **printf() e scanf()**, ***só que operam em arquivos em disco***.
- Forma geral:

```
int fprintf (FILE *fp , char *string_de_controle, lista_argumentos);

int fscanf (FILE *fp , char *string_de_controle, lista_argumentos);
```

- Exemplo:
  - fscanf (stdin,"%s %d", nome, &numero);
  - fprinf (stdin,"%s %d\n", nome, numero);

Funções com Arquivos
Função remove()

- Para apagar um arquivo especifico, usamos a função remove().
- Forma geral: **int remove (char *nome_arquivo);**
- Exemplo:

```
if (remove(fp)==0) printf("Arquivo foi deletado com sucesso!\n");
```


Contato

eferlin@live.com

(BLOG) professorferlin.blogspot.com

(SITE) professorferlin.webnode.com.br

(YOUTUBE) ProfEdsonPedroFerlin