1.A hierarquia de dados

Basicamente, todos os itens de dados processados por um computador são reduzidos a combinações de 0s e 1s. Isso ocorre porque é simples e econômico para construir dispositivos eletrônicos que podem assumir dois estados estáveis. Os menores itens de dados em um computador podem assumir o valor 0 ou 1. Tal item de dados é chamado um bit. Os circuitos computacionais realizam várias manipulações simples de bit como determinar o valor de um bit, redefinir o valor de um bit e inverter o valor de um bit.

Da mesma forma que os caracteres são compostos de bits, os campos são compostos de caracteres. Um campo é um grupo de caracteres que possui um significado. Os itens de dados processados pelos computadores formam uma hierarquia de dados na qual os itens de dados se tornam maiores e mais complexos na estrutura à medida que evoluímos de bits para caracteres, campos e assim por diante.

Um registro (“struct” em C) é composto de vários campos. Em um sistema de folha de pagamento, por exemplo, um registro de um determinado empregado pode consistir nos seguintes campos: número de previdência social; nome; endereço; valor do salário hora; número de dispensas solicitadas; total de vencimento no presente ano; total de impostos federais retidos na fonte

Dessa forma, um registro é um grupo de dados relacionados entre si. Um arquivo é um grupo de registros relacionados. Um arquivo de folha de pagamento de uma companhia contém um registro para cada empregado.

Para facilitar a recuperação de registros específicos de um arquivo, pelo menos um campo em cada registro é escolhido como chave dos registros. A chave dos registros identifica um registro como pertencendo a uma determinada pessoa ou entidade.

Há muitas maneiras de organizar registros em um arquivo. O tipo mais popular de organização é chamado arquivo sequencial no qual normalmente os registros são armazenados em ordem segundo o campo-chave dos registros. Em um arquivo de folha de pagamentos, geralmente os registros são colocados em ordem segundo o número de inscrição na Previdência Social.

A maioria das empresas utiliza muitos arquivos diferentes para armazenar dados. As companhias podem ter arquivos de folhas de pagamento, arquivos de contas a receber, arquivos de contas a pagar, arquivos de inventário, características a respeito de todos os itens gerenciados pela empresa e muitos outros tipos de arquivos. Algumas vezes, um grupo de arquivos relacionados entre si é chamado de banco de dados. Um conjunto de programas que se destina a criar e gerenciar bancos de dados é chamado sistema de gerenciamento de banco de dados (SGBD)

2.Arquivos e fluxos

A linguagem C visualiza cada arquivo simplesmente como fluxo sequencial de bytes. Cada arquivo termina ou com um marcador de final de arquivo (EOF), ou usa em específico gravado em uma estrutura administrativa de dados, mantida pelo sistema. Quando um arquivo é aberto, um fluxo é associado àquele arquivo. Três arquivos e seus respectivos fluxos são abertos automaticamente quando a execução de um programa se inicia, o de entrada padrão, o de saída padrão e o de erro padrão. Os fluxos fornecem canais de comunicação entre arquivos e programas.

Abrir um arquivo retorna um ponteiro para uma estrutura "FILE", que contém informações usadas para processar o arquivo. Esta estrutura inclui um descritor de arquivo em um array do sistema operacional chamado tabela de arquivos abertos. Cada elemento do array contém um bloco de controle de arquivo (BCA) que o sistema operacional usa para administrar um arquivo em particular. O padrão de entrada, o padrão de saída e o padrão de erros são manipulados por meio dos ponteiros de arquivos ***stdin***, ***stdout***, e ***stderr***.

A biblioteca padrão fornece muitas funções para ler dados de arquivos e também para gravar dados em arquivos.

A função ***fget*** recebe como argumento um ponteiro **FILE** do arquivo para o qual o caractere será lido. A chamada do ***fget***(***stdin***) lê um caractere de ***stdin***. Essa chamada é equivalente à chamada ***getchar()***.

Função	Protótipo	Descrição
`fgetc();`	***`int fgetc`*** `(FILE * stream);`	`Retorna o caractere atualmente apontado pelo indicador interno de posição do arquivo do fluxo especificado. Se o fluxo estiver no fim do arquivo quando chamado, a função retornará EOF e definirá o indicador de fim de arquivo para o fluxo`
`fgets();`	***`char *fgets`*** `(char *str, int num, FILE * stream);`	`Lê caracteres do stream e os armazena como uma string C em str até que (num-1) caracteres tenham sido lidos ou uma nova linha ou o fim do arquivo seja alcançado, o que acontecer primeiro. Um caractere de nova linha faz com que fgets pare de ler, mas é considerado um caractere válido pela função e incluído na string copiada para str. Um caractere nulo de terminação é automaticamente anexado após os caracteres copiados para str`

A função ***fputc*** recebe como argumentos um caractere a ser escrito e um ponteiro para o arquivo ao qual o caractere será escrito.

Função	Descrição

fputc();	
fputs();	

3.Criando um arquivo de acesso sequencial

A linguagem C não impõe estrutura a um arquivo. Assim, noções como um registro de um arquivo não existem como parte da linguagem C. Portanto, o programador deve fornecer qualquer estrutura de arquivo para satisfazer as exigências de uma aplicação específica.

Os programas podem não processar nenhum arquivo, processar um arquivo ou processar vários arquivos. Cada arquivo usado em um programa deve ter um nome exclusivo e terá um ponteiro diferente de arquivo retornado por ***fopen***. Todas as funções subsequentes de processamento de arquivo depois do arquivo ser aberto devem se referir ao arquivo com o ponteiro de arquivo apropriado. Se ocorrer um erro durante a abertura de um arquivo de qualquer modo, ***fopen*** retorna **NULL**.

4.Lendo dados de um arquivo de acesso sequencial

Os dados são armazenados em arquivos de modo que possam ser recuperados para processamento quando necessário.

Modo	Descrição
r	Abre um arquivo para leitura
w	Cria um arquivo para gravação. Se o arquivo já existir, elimina o conteúdo atual
a	Anexa: abre ou cria um arquivo para gravação no final do arquivo
r+	Abre um arquivo para atualização (leitura e gravação)
w+	Cria um arquivo para atualização. Se o arquivo já existir, elimina o conteúdo atual.
a+	Anexa: abre ou cria um arquivo para atualização; a gravação é feita no final do arquivo

Para recuperar dados sequencialmente de um arquivo, normalmente um programa começa a leitura no início do arquivo e lê todos os dados consecutivamente até chegar ao dado procurado. Pode ser desejável processar os dados sequencialmente várias vezes durante a execução de um programa. Uma instrução como a abaixo, faz com que o ponteiro de posição do arquivo do programa - indicando o número do próximo byte do arquivo a ser lido ou gravado - seja reposicionado no início do arquivo apontado por “cfPtr”.

```
rewind(cfPtr);
```

O ponteiro de posição do arquivo não é realmente um ponteiro. Em vez disso, ele é um valor inteiro que especifica a posição do byte no qual ocorrerá a próxima leitura ou gravação.

Algumas vezes ele é chamado de offset de arquivo. O ponteiro de posição de arquivo é um membro da estrutura FILE associado a cada arquivo.

5.Arquivos de acesso aleatório

Os registros em um arquivo craid com a função de saída formatado ***fprintf*** não possuem necessariamente o mesmo tamanho. Entretanto, normalmente os registros de um arquivo de acesso aleatório possuem o mesmo tamanho e podem ser acessados diretamente sem passar por outros registros. Isso faz com que os arquivos de acesso aleatórios sejam apropriados para sistemas de reservas de vôos, sistemas de bancos, sistemas de ponto de venda e outros tipos de sistemas de processamento de transações que exigem acesso rápido a dados específicos. Há outras maneiras de implementar arquivos de acesso aleatório.

Como todos os registros de um arquivo de acesso aleatório possuem o mesmo comprimento, a localização exata de um registro relativo ao início do arquivo pode ser calculada como uma função da chave dos registros.

Os dados podem ser inseridos em um arquivo de acesso aleatório sem que sejam destruídos outros dados no arquivo. Os dados armazenados previamente podem ser atualizados ou excluídos sem reescrever todo o arquivo.

6.Criando um arquivo de acesso aleatório

A função ***fwrite*** transfere para um arquivo um número especificado de bytes, iniciando em um determinado local da memória. Os dados são gravados iniciando no local do arquivo indicado pelo ponteiro de posição do arquivo. A função ***fread*** transfere um número determinado de bytes de um local específico da memória, definido pelo ponteiro de posição do arquivo, para uma área da memória, iniciando em um endereço indicado.

Agora, ao gravar um inteiro, ao invés de usar a função abaixo, que poderia imprimir o mínimo de 1 dígito ou o máximo de 11 dígitos para um inteiro de 4 bytes, podemos usar a função abaixo dela que sempre grava 4 bytes da variável número no arquivo representado por fPtr.

```
fprintf(fPtr, "%d", numero);
fwrite(Snumero, sizeof(int), 1, fPtr);
```

As funções ***fread*** e ***fwrite*** são capazes de ler e gravar ***arrays*** de dados de e para um disco. O terceiro argumento, tanto de ***fread*** quanto de ***fwrite***, é o número de elementos do ***array*** que deve ser lido do disco ou gravado nele. A chamada anterior à função ***fwrite*** grava um inteiro simples no disco, portanto o terceiro argumento é 1.

Os programas de processamento de arquivos raramente gravam um campo isolado em um arquivo. Normalmente eles gravam uma ***struct*** de cada vez.

As várias seções que se seguem apresentam as técnicas necessárias para criar o programa de processamento de créditos.

A função ***fwrite*** pode até ser usada para gravar vários elementos de um ***array*** de objetos. Para gravar vários elementos de um ***array***, o programador deve fornecer, como primeiro argumento da chamada a ***fwrite***, um ponteiro para um ***array*** e especificar como terceiro argumento daquela chamada o número de elementos a serem gravados.

7.Lendo dados aleatoriamente em um arquivo de acesso aleatório

A função ***fread*** lê um número especificado em bytes de um arquivo para a memória. Por exemplo, a instrução abaixo lê o número de bytes determinado por ***sizeof(struct dadosCliente)***. Os bytes são lidos do local no arquivo especificado pelo ponteiro de posição do arquivo.

```
fread(&cliente, sizeof(struct dadosCliente), 1, cfPtr);
```

A função ***fread*** pode ser usada para ler vários elementos, com tamanho fixo, de um ***array***, fornecendo um ponteiro para o ***array*** no qual os elementos serão armazenados e indicando o número de elementos a ser lido. A instrução anterior especifica que um elemento deve ser lido. Para ler mais de um elemento, especifique o número de elementos no terceiro argumento da instrução ***fread***.