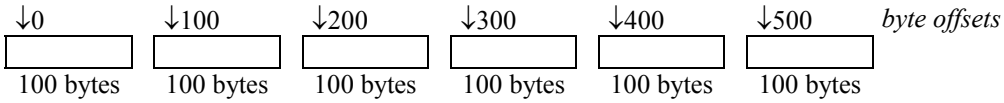
Manipulação de Arquivos Binários

Até agora vimos como criar arquivos de acesso seqüencial bem como realizar a busca por uma informação particular nesses arquivos. Arquivos seqüenciais são inadequados para aplicações ditas de *acesso instantâneo*. Algumas aplicações típicas de acesso instantâneo são os sistemas de reservas de vôos, sistemas de ponto-de-venda, caixas automáticos, sistemas bancários e outros tipos de sistemas de processamento de transações que requerem acesso rápido a um dado específico.

Por exemplo, no banco onde você possui sua conta corrente, mesmo quando você utiliza um caixa automático, sua conta é verificada se possui fundos em alguns segundos. Este tipo de acesso instantâneo é possível através dos *arquivos de acesso aleatório*. Os registros individuais em um arquivo de acesso aleatório podem ser acessados direta e rapidamente sem ser necessário procurar em todos os registros.

Como vimos em aulas anteriores, C++ não impõe estrutura para arquivos. Assim, a aplicação que deseja fazer uso de arquivos de acesso aleatório deve literalmente criá-los. Uma variedade de técnicas pode ser utilizada para criar arquivos de acesso aleatório e possivelmente a mais simples requer que todos os registros no arquivo possuam um tamanho fixo. A utilização de registros de tamanho fixo facilita o cálculo da localização exata de qualquer registro relativamente ao início do arquivo (como uma função do tamanho do registro e da chave).

Na figura seguinte é mostrado um esquema C++ de um arquivo de acesso aleatório, composto de registros de tamanho fixo (cada registro possui 100 bytes). Um arquivo de acesso aleatório é como um trem com vários vagões, alguns cheios, outros vazios.



Os dados podem ser inseridos em um arquivo de acesso aleatório sem destruir outros dados no arquivo. Os dados armazenados anteriormente também podem ser atualizados ou apagados sem a necessidade de reescrever o arquivo inteiro.

Criando um Arquivo Binário

No exemplo seguinte é criado um arquivo que pode ser utilizado para gerenciar um sistema de contas de clientes. Assume-se que cada cliente é identificado pelo número de sua conta, nome e saldo. A seguinte estrutura de dados é definida:

```
 1 // cliente.h
 2 // Definição da estrutura cliente
 3 #ifndef CLIENTE_H
 4 #define CLIENTE_H
 5
 6 struct cliente
 7 { int   numero;
 8   char  nome[30];
 9   float saldo;
10 };
11
12 #endif
```

Com base nisso, o programa seguinte cria o arquivo denominado “credito.dat” contendo 100 registros em “branco” ou seja, com informações que podem ser consideradas como indicadoras de um registro “vazio”

(linha 9). Na linha 17, os registros (todos iguais a **clienteVazio**) são gravados no arquivo através do comando:

```
outCredito.write((const char *)(&clienteVazio), sizeof(cliente));
```

O método **write** requer dois parâmetros, sendo o primeiro um ponteiro constante para o tipo de dados **char**. Assim, é necessário converter a variável **clienteVazio** (do tipo cliente) para o tipo (const char *). Em algumas implementações é necessário substituir o comando anterior por:

```
outCredito.write(reinterpret_cast<const char *>(&clienteVazio), sizeof(cliente));
```

O segundo parâmetro do método **write** é um inteiro do tipo **size_t** especificando o número de bytes a serem escritos. O tamanho do registro de cliente é dado pela função **sizeof(cliente)**, que determina quantos bytes a estrutura de dados (registro) cliente ocupa no computador em questão.

```
 1 // cliente1.cpp
 2 // Criação sequencial de um arquivo de acesso aleatório
 3 #include <iostream>
 4 #include <fstream>
 5 #include <cstdlib>
 6 using namespace std;
 7 #include "cliente.h"
 8 int main()
 9 { cliente clienteVazio = {0, "", 0.0};
10   Ofstream outCredito("credito.dat",ios::out);
11
12   if(! outCredito)
13   { cerr << "Arquivo credito.dat nao pode ser aberto." << endl;
14     exit(1);
15   }
16   for (int i = 0; i < 100; i++)
17     outCredito.write((const char *)(&clienteVazio),sizeof(cliente));
18   return 0;
19 }
```

Escrevendo Dados Aleatoriamente em um Arquivo

Uma vez criado o arquivo com o programa anterior, agora podemos proceder à atualização dos registros contidos nele no programa seguinte. Observe na linha 10: o modo de abertura **ios::ate** abre um arquivo para gravação e move o ponteiro de arquivo para o final-de-arquivo; os dados podem ser gravados em qualquer posição no arquivo neste modo de abertura de arquivo.

O programa então lê um número de conta, **c.numero** (linhas 17-18) entre 1 e 100 (que é o total de registros existentes no arquivo criado com o programa anterior). Em seguida o nome e saldo do cliente são lidos (linhas 20-21). Na linha 24 o ponteiro de arquivo é posicionado na conta desejada (lembre-se que o ponteiro de arquivo indica o número de bytes deslocados em relação ao início do arquivo e que o arquivo começa no byte de número zero. Assim, para o primeiro registro, devemos posicionar no byte zero; para o segundo registro devemos posicionar no byte sizeof(cliente) e assim por diante. Para o n-ésimo registro, devemos posicionar no byte (n-1)*sizeof(cliente).

Uma vez o ponteiro de arquivo na posição correta, na linha 26 os dados fornecidos pelo usuário são regravados por cima dos existentes, atualizando as informações.

```
 1 // cliente2.cpp
 2 // Escrevendo Dados de modo aleatorio
 3 #include <iostream>
 4 #include <fstream>
 5 #include <cstdlib>
 6 using namespace std;
 7 #include "cliente.h"
 8 int main()
 9 { cliente c;
10   ofstream outCredito("credito.dat",ios::ate);
```

```
11
12    if(! outCredito)
13    { cerr << "Arquivo credito.dat nao pode ser aberto." << endl;
14      exit(1);
15    }
16
17    cout << "Enter numero da conta de 1 a 100 (0 termina) ";
18    cin  >> c.numero;
19    while(c.numero > 0 && c.numero <= 100)
20    { cout << "Entre nome, saldo\n? ";
21      cin >> c.nome >> c.saldo;
22
23      // posicionar no cliente desejado
24      outCredito.seekp((c.numero - 1) * sizeof(cliente));
25      // atualizar dados
26      outCredito.write((const char *)(&c),sizeof(cliente));
27
28      cout << "Enter numero da conta de 1 a 100 (0 termina) ";
29      cin  >> c.numero;
30    }
31    return 0;
32  }
```

Lendo Dados Seqüencialmente a partir de um Arquivo de Acesso Aleatório

Considerando o próximo programa, nas linhas 24 e 32, os registros são lidos do arquivo através do comando:

```
inCredito.read((char *)(&c), sizeof(cliente));
```

O método **read** requer dois parâmetros, sendo o primeiro um ponteiro para o tipo de dados **char**. Assim, é necessário converter a variável **c** (do tipo cliente) para o tipo (char *). Em algumas implementações é necessário substituir o comando anterior por:

```
inCredito.read(reinterpret_cast<char *>(&c), sizeof(cliente));
```

O segundo parâmetro do método **read** é um inteiro do tipo **size_t** especificando o número de bytes a serem lidos. Como antes mencionado, o tamanho do registro de cliente é dado pela função **sizeof(cliente)**, que determina quantos bytes a estrutura de dados (registro) cliente ocupa no computador em questão.

Como o arquivo foi aberto no modo leitura (ios::in), observe que não há nenhuma chamada ao método **seekg** para posicionamento do ponteiro de arquivo. Ao ler um registro, o ponteiro de arquivo é automaticamente incrementado pela quantidade de bytes lidos, posicionando o ponteiro de arquivo no próximo registro do arquivo. Assim, mesmo o arquivo sendo de acesso aleatório, os dados podem ser lidos seqüencialmente.

O laço na linha 25

```
while (inCredito && ! inCredito.eof())
```

utiliza o método **eof** para determinar quando o final-de-arquivo é atingido e termina a execução do laço. Além disso, se houver um erro de leitura do arquivo, o laço termina uma vez que **InCredito** será **false**.

```
 1  // cliente3.cpp
 2  // Lendo Dados Sequencialmente a partir de um Arquivo de Acesso Aleatório
 3  #include <iostream>
 4  #include <iomanip>
 5  #include <fstream>
 6  #include <cstdlib>
 7  using namespace std;
 8  #include "cliente.h"
 9  int main()
10  { cliente c;
11    ifstream inCredito("credito.dat", ios::in);
12
13    if(! inCredito)
14    { cerr << "Arquivo credito.dat nao pode ser aberto." << endl;
```

```
15     exit( 1 );
16   }
17
18   cout << setiosflags(ios::left)
19        << setw(10) << "Conta"
20        << setw(30) << "Nome"
21        << resetiosflags(ios::left)
22        << setw(10) << "Saldo" << endl;
23
24   inCredito.read((char *)(&c),sizeof(cliente));
25   while (inCredito && ! inCredito.eof())
26   { if(c.numero != 0)
27       cout << setiosflags( ios::left )
28           << setw(10) << c.numero
29           << setw(30) << c.nome
30           << setw(10) << setprecision( 2 ) << resetiosflags(ios::left)
31           << setiosflags(ios::fixed | ios::showpoint) << c.saldo << '\n';
32     inCredito.read((char *)(&c),sizeof(cliente));
33   }
34   return 0;
35 }
```

Exemplo de Programa de Processamento de Transações

O programa seguinte utiliza um arquivo de acesso aleatório para obter acesso "instantâneo" aos registros de um arquivo, simulando contas de clientes em um banco. A opção número 1 deve ser utilizada somente uma única vez, já que ela cria o arquivo com 100 registros "em branco" ou sobrepõe os 100 registros existentes com valores que consideramos como registros vazios. Note que, embora interpretemos os registros como vazios, eles contêm dados, ou seja, zero para o número de conta e saldo e um vetor de caracteres de tamanho zero para o nome.

```
 1 #include <iostream>
 2 #include <fstream>
 3 #include <iomanip>
 4 #include <cstdlib>
 5 #include <string>
 6 using namespace std;
 7 #include "cliente.h"
 8 enum Escolhas {CRIAR=1, TELA, ARQUIVOTEXTO, ATUALIZAR, NOVO, APAGAR, FIM};
 9 //-------------------------------------------------------------------------
10 Escolhas enterChoice()
11 // pre: nenhuma
12 // pos: exibe menu na tela e solicita acao a ser tomada ao usuario
13 { int menuChoice;
14   cout << "\nMenu:" << endl
15        << "1 - cria registros vazios no arquivo\n"
16        << "2 - lista os dados na tela\n"
17        << "3 - armazena os dados no arquivo texto \"print.txt\"\n"
18        << "4 - atualiza uma conta que ja contenha informacoes\n"
19        << "5 - insere informacoes em uma nova conta\n"
20        << "6 - apaga informacoes de uma conta\n"
21        << "7 - fim do programa\n"
22        << "Opcao: ";
23    cin >> menuChoice;
24    return (Escolhas) menuChoice;
25 }
26 //-------------------------------------------------------------------------
27 void create(fstream &f)
28 // pre: arquivo f aberto
29 // pos: insere/sobrepoe 100 registros no arquivo, destruindo
30 //      qualquer informacao anterior
31 { cliente clienteVazio = {0, "", 0.0};
32
33   f.seekg(0);
34   for (int i = 0; i < 100; i++)
35     f.write((const char *)(&clienteVazio),sizeof(cliente));
36 }
37 //-------------------------------------------------------------------------
38 void outputLine(ostream &output, const cliente &c)
39 // pre: arquivo output aberto para escrita, cliente c contendo dados de cliente
40 // pos: escreve dados do cliente c no arquivo output
41 { output << setiosflags(ios::left)
42        << setw(10) << c.numero
```

```
        << setw(30) << c.nome
        << setw(10) << setprecision(2) << resetiosflags(ios::left)
        << setiosflags(ios::fixed | ios::showpoint) << c.saldo << '\n';
}
//-------------------------------------------------------------------------
void screen(fstream &f)
// pre: Arquivo f contendo dados de clientes aberto
// pos: Cria um arquivo texto formatado para impressao
{ cliente c;

  cout << setiosflags(ios::left)
       << setw(10) << "Conta"
       << setw(30) << "Nome"
       << resetiosflags(ios::left) << setw(10) << "Saldo" << endl;

  f.seekg(0);
  f.read((char *)(&c),sizeof(cliente));
  while(! F.eof())
  { if(c.numero != 0)
      outputLine(cout,c);
    f.read((char *)(&c),sizeof(cliente));
  }
}
//-------------------------------------------------------------------------
void textFile(fstream &f)
// pre: Arquivo f contendo dados de clientes aberto
// pos: Cria um arquivo texto formatado para impressao
{ cliente c;
  ofstream outPrintFile("print.txt",ios::out);

  if(! outPrintFile)
  { cerr << "Arquivo print.txt nao pode ser aberto." << endl;
    exit(1);
  }

  outPrintFile << setiosflags(ios::left)
               << setw(10) << "Conta"
               << setw(30) << "Nome"
               << resetiosflags(ios::left) << setw(10) << "Saldo" << endl;

  f.seekg(0);
  f.read((char *)(&c),sizeof(cliente));
  while(! f.eof())
  { if(c.numero != 0)
      outputLine(outPrintFile,c);
    f.read((char *)(&c),sizeof(cliente));
  }
  outPrintFile.close();
}
//-------------------------------------------------------------------------
int getAccount(string msg)
// pre: nenhuma
// pos: imprime msg, obtem do teclado um numero de conta entre 1 e 100 e retorna-o
{ int conta;

  do
  { cout << msg << " (1 - 100): ";
    cin >> conta;
  } while (conta < 1 || conta > 100);

  return conta;
}
//-------------------------------------------------------------------------
void updateRecord(fstream &f)
// pre: Arquivo f aberto
// pos: Atualiza o saldo de uma conta
{ int conta;
  cliente c;
  float transacao;

  conta = getAccount("Conta a ser atualizada");
  f.seekg((conta – 1) * sizeof(cliente)); // posicionar na conta desejada
  f.read((char *)(&c),sizeof(cliente));   // ler dados da conta
  if(c.numero != 0) // conta contem informacao?
  { outputLine(cout,c);
    cout << "\nEntre deposito (+) ou retirada (-): ";
    cin  >> transacao;
    c.saldo += transacao;
    outputLine(cout,c);
    f.seekp((conta - 1) * sizeof(cliente));      // posicionar na conta desejada
```

```
123     f.write((const char *)(&c),sizeof(cliente)); // atualizar
124   }
125   else
126     cerr << "Conta #" << conta  << " nao possui informacao." << endl;
127 }
128 //-------------------------------------------------------------------------
129 void newRecord(fstream &f)
130 // pre: arquivo f aberto
131 // pos: insere informacao em um registro de conta que esteja vazio (sem informacao)
132 { int conta;
133   cliente c;
134
135   conta = getAccount("Numero da nova conta");
136   f.seekg((conta-1) * sizeof(cliente)); // posicionar na conta desejada
137   f.read((char *)(&c),sizeof(cliente)); // ler dados da conta
138   if(c.numero == 0)
139   { cout << "Entre nome, saldo\n? ";
140     cin >> c.nome >> c.saldo;
141     c.numero = conta;
142     f.seekp((conta - 1) * sizeof(cliente));      // posicionar na conta desejada
143     f.write((const char *)(&c),sizeof(cliente)); // atualizar
144   }
145   else
146     cerr << "Conta #" << conta << " ja possui informacao." << endl;
147 }
148 //-------------------------------------------------------------------------
149 void deleteRecord(fstream &f)
150 // pre: arquivo f aberto
151 // pos: conta fornecida pelo usuario e apagada
152 //      (informação em branco)
153 { int conta;
154   cliente c, clienteVazio = {0, "", 0.0};
155
156   conta = getAccount("Conta a ser apagada");
157   f.seekg((conta-1) * sizeof(cliente));
158   f.read((char *)(&c),sizeof(cliente));
159   if(c.numero != 0)
160   { f.seekp((conta - 1) * sizeof(cliente));
161     f.write((const char *)(&clienteVazio), sizeof(cliente));
162     cout << "Conta #" << conta << " apagada." << endl;
163   }
164   else
165     cerr << "Conta #" << conta << " ja esta apagada." << endl;
166 }
167 //-------------------------------------------------------------------------
168 void main()
169 { Escolhas opcao;
170   fstream inOutCredito("credito.dat", ios::in | ios::out);  // leitura e escrita
171
172   if(! inOutCredito)
173   { cerr << "Arquivo credito.dat nao pode ser aberto." << endl;
174     exit (1);
175   }
176
177   while ((opcao = enterChoice()) != FIM)
178   { switch (opcao)
179     { case CRIAR:
180         create(inOutCredito);
181         break;
182       case TELA:
183         screen(inOutCredito);
184         break;
185       case ARQUIVOTEXTO:
186         textFile(inOutCredito);
187         break;
188       case ATUALIZAR:
189         updateRecord(inOutCredito);
190         break;
191       case NOVO:
192         newRecord(inOutCredito);
193         break;
194       case APAGAR:
195         deleteRecord(inOutCredito);
196         break;
197       default:
198         cerr << "Opcao incorreta\n";
199         break;
200     }
201     inOutCredito.clear();  // limpa o indicador de final-de arquivo
202   }
```

```
203   return 0;
204 }
```