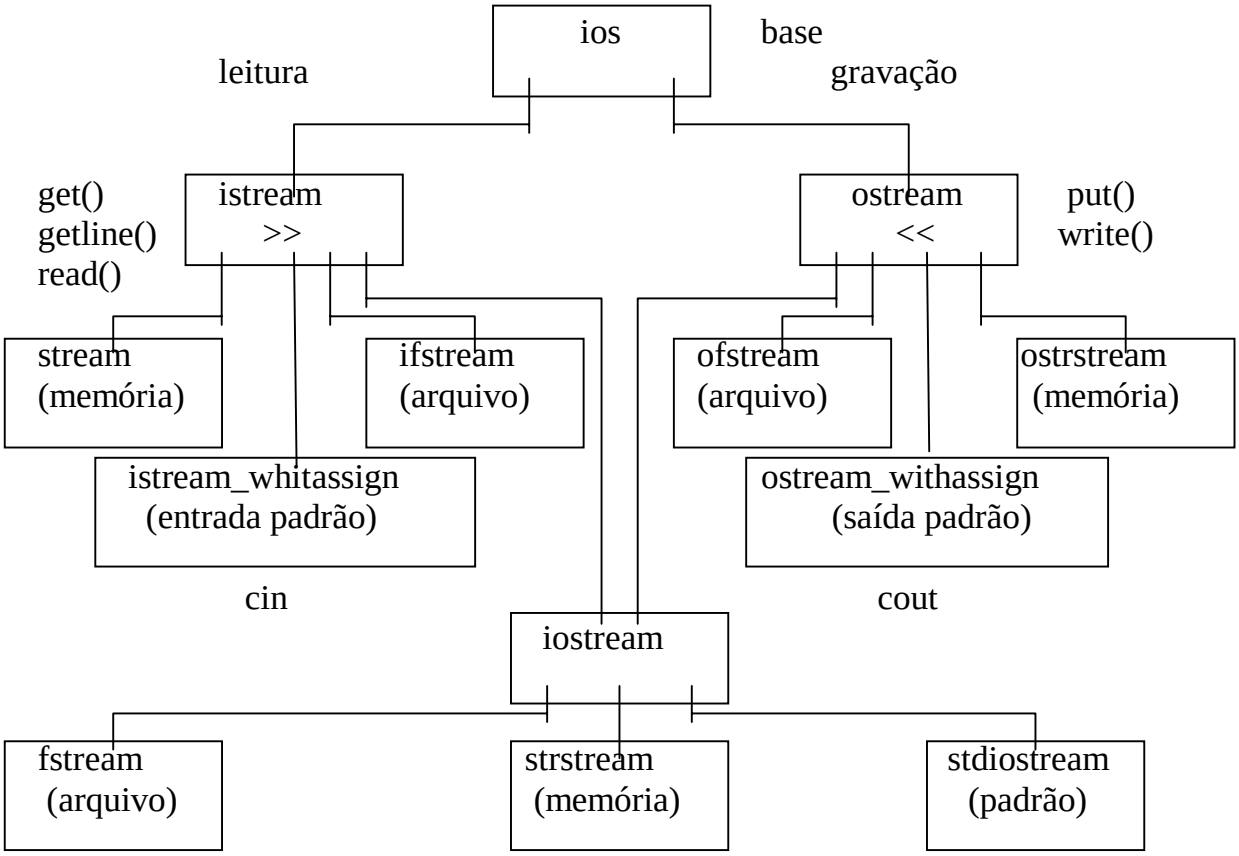
C++ - Operações com arquivos

Em C++, as classes **iostream** são usadas para executar operações de leitura e gravação em arquivos. Um objeto **stream** (fluxo de caracteres) pode ser pensado como um lugar de recebimento ou envio de bytes.

O conceito de arquivo é ampliado no sentido de considerar não somente os que existem em disco mas também o teclado, o vídeo, a impressora e as portas de comunicação. Ex.: o objeto **cout** representa o vídeo (recebimento de bytes) e o objeto **cin** representa o teclado (envio de bytes). As classes iostream interagem com esses arquivos.

Hierarquia de classes iostream



Gravando um caracter no arquivo :

```
#include <fstream.h>                // para funções de arquivo
void main()
{
   ofstream fout("a:teste.txt");     // Abre arquivo para gravação
                                                //em modo texto
   char ch;

   cout << "Digite um texto ";
   cout << "\nPressione CTRL_Z para encerrar ";
   while(cin.get(ch))                // Lê um caracter do teclado
        fout.put(ch);          // Grava o caracter no arquivo
}
```

Lendo um caracter do arquivo e exibindo no vídeo:

```
#include <fstream.h>
void main()
{
   ifstream fin("a:teste.txt");     // Abre arquivo para leitura
                                            //em modo texto
   char ch;
                                    // Enquanto não for fim de arquivo:
   while(fin.get(ch))               // lê um caracter do arquivo
        cout << ch;                 // imprime o caracter no vídeo
}
```

Gravando uma linha no arquivo :

```
#include <fstream.h>
void main()
{
   ofstream fout("a:teste1.txt");  // Cria arquivo para gravação
                                                //em modo texto
   fout << "Texto gravado no arquivo por meio de programa C++";      }
```

Lendo uma linha do arquivo e exibindo no vídeo:

```
#include <fstream.h>
void main()
{
        const int MAX=80;         char buff[MAX];
```

```
        ifstream fin("a:teste1.txt");       // Abre arquivo para leitura
                                                    //em modo texto
        while(fin)                                  // Enquanto não for fim de arquivo
        {       fin.getline(buff, MAX);   // Lê uma linha do arquivo
                cout << buff << '\n';       }       // Exibe no vídeo
}
```

A função open() : quando usamos objetos das classes **ofstream** ou **ifstream** é necessário associá-lo a um arquivo. Esta associação pode ser feita usando o **construtor** da classe, como nos exemplos anteriores, ou usando a função **open(),** membro da classe **fstream,** numa instrução após a criação do objeto.

Tanto o construtor como open() aceitam a inclusão de um segundo argumento indicando o modo de abertura do arquivo.

Modos de abertura	Descrição
ios::in	Abre para leitura (default de ifstream).
ios::out	Abre para gravação (default de ofstream),
ios::ate	Abre e posiciona no final do arquivo. (Este modo trabalha com leitura e gravação)
ios::app	Grava a partir do fim do arquivo
ios::trunc	Abre e apaga todo o conteúdo do arquivo
ios::nocreate	Erro de abertura se o arquivo não existe
ios::noreplace	Erro de abertura se o arquivo existir
ios::binary	Abre em binário (default é texto)

Modo texto e **modo binário** : numa operação de gravação em arquivos aberto em modo **texto** (default) o caracter '\n' é expandido em dois bytes, carriage-return e linefeed (CR/LF), antes de ser gravado. Em operações de leitura, o par de bytes CR/LF é convertido para um único byte '\n'. Quando o arquivo é aberto em modo **binário** não há esta conversão. Verifique a diferença de modo texto e modo binário através dos próximos exemplos .

```
#include <fstream.h>
#include <stdlib.h>         // para exit()
void main(int argc, char **argv)
{
   ifstream fin;
   char ch;
   int cont = 0;
   if(argc !=2)
   {
        cout << "\nForma de uso: c:\nome_programa nome_arquivo ";
```

```
        exit(1);
    }
    fin.open(argv[1]);        // Abre o arquivo em modo texto (default)

    while(fin.get(ch)) cont++;      // Lê um caracter do arquivo e conta
    cout << "\nNumero de caracteres : " << cont;      // Imprime contador
}
```

Compare o tamanho do arquivo apresentado pelo **programa** e o observado no **explorer.** A razão da diferença está no tratamento do '\n'.

Modifique o programa para que o arquivo seja aberto em modo binário substituindo a linha

fin.open(argv[1]);

por **fin.open(argv[1], ios::binary);**

e execute novamente. Observe que deste vez o tamanho do arquivo apresentado pelo **programa** coincide com o do **explorer.**

Gravando um objeto no arquivo :

```
#include <fstream.h>              // para funções de arquivo
#include <conio.h>                // para getche()
#include <stdio.h>          // para gets()
class Livro
{
        private :
                char titulo[50];
                char autor[50];
                int numreg;
                double preco;
        public :
                void novonome();
};

void Livro::novonome()
{
        cout << "\nDigite Titulo : ";      gets(titulo);
        cout << "\nDigite Autor : ";       gets(autor);
        cout << "\nDigite o Numero do Registro : "; cin >> numreg;
        cout << "\nDigite o Preco : "; cin >> preco;
}

void main()
{
```

```
    ofstream fout("a:lista.dat");
      Livro li;
      do
      {
            li.novonome();                                  // lê do teclado
            fout.write((char *)&li, sizeof(Livro)); // grava no arquivo
            cout << "\nInserir outro livro (s/n) ";
      }while(getche( ) == 's');
}
```

A função **write( )** recebe dois argumentos: o endereço do objeto a ser gravado e o tamanho do objeto em bytes.

Lendo um objeto do arquivo :

```
#include <fstream.h>
#include <stdio.h>
class Livro
{
      private :
            char titulo[50];
            char autor[50];
            int numreg;
            double preco;
      public :
            void print();
};

void Livro::print()
{
      cout << "\nTitulo : " << titulo;
      cout << "\nAutor : " << autor;
      cout << "\nNumero do Registro : " << numreg;
      cout << "\nPreco : " << preco;
}
void main()
{
    ifstream fin("a:lista.dat");          // abre arquivo para leitura
      Livro li;
      while(fin)                          // enquanto não for fim de arquivo...
    {
            fin.read((char*)&li,sizeof(Livro));     // lê do arquivo
            li.print();                             // imprime no vídeo
       }                                            }
```

Gravando e **lendo** **objetos** de um **mesmo** **arquivo** :

```
#include <fstream.h>
#include <conio.h>
#include <stdio.h>

class Livro
{
        private :
                char titulo[50];
                char autor[50];
                int numreg;
                double preco;

        public :
                void novonome();
                void print();
};

void Livro::novonome()
{
        cout << "\nDigite Titulo : ";      gets(titulo);
        cout << "\nDigite Autor : ";       gets(autor);
        cout << "\nDigite o Numero do Registro : "; cin >> numreg;
        cout << "\nDigite o Preco : "; cin >> preco;
}

void Livro::print()
{
        cout << "\nTitulo : " << titulo;
        cout << "\nAutor : " << autor;
        cout << "\nNumero do Registro : " << numreg;
        cout << "\nPreco : " << preco;
}
```

```
void main()
{
        fstream fio;                    // cria objeto de leitura e gravação
        Livro li;               // cria objeto Livro
        fio.open("a:lista.dat", ios::ate | ios::out | ios::in);
        // abre   e posiciona no final do arquivo, para operações de
        // gravação  e leitura
        do
        {
                li.novonome();                                  // lê do teclado
                fio.write((char *)&li, sizeof(Livro));   // grava no arquivo
                cout << "\nInserir outro livro (s/n) ";
        }while(getche( ) != 'n');                       // fim se 'n'
        fio.seekg(0);                   //coloca ponteiro no inicio do arquivo
        cout << "\nLista de Livros do arquivo ";
        cout << "\n=========================";
        while (fio.read((char *)&li, sizeof(Livro)))            // lê do arquivo
                li.print();                                       // imprime no vídeo
}
```

Operadores bit-a-bit : C++ tem seis operadores que atuam em bits individuais de variáveis, e que serão usados nos próximos exemplos. São eles :

Símbolo	**Operação**	**Exemplo** unsigned x = 5; 00000000 00000101 unsigned y = 9; 00000000 00001001
&	E (AND)	z = x & y; 00000000 00000001
|	OU (OR)	z = x | y; 00000000 00001101
^	OU exclusivo (XOR)	z = x ^ y; 00000000 00001100
>>	Deslocamento à direita	z = x >> 1; 00000000 00000010
<<	Deslocamento à esquerda	z = x << 1; 00000000 00001010
~	Complemento (unário)	~x 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0

Observ. : Cada objeto stream está associado a dois valores inteiros :

- Posição atual de leitura
- Posição atual de gravação

Seus valores especificam o número do byte do arquivo onde ocorrerá a próxima leitura ou gravação. As funções a seguir trabalham com estes valores :

Função	Descrição
seekg( )	Movimenta a posição atual de leitura (get)
seekp( )	Movimenta a posição atual de gravação (put)
tellg( )	Retorna a posição atual de leitura (em bytes), a partir do início do arquivo
tellp( )	Retorna a posição atual de gravação (em bytes), a partir do início do arquivo

Protótipo de seekg( ) (e seekp( )) :

```
istream& seekg( long pos, seek_dir posicao= ios::beg);
```

Possíveis valores para o segundo argumento :

- ios::beg a partir do início do arquivo
- ios::cur a partir da posição corrente (atual)
- ios::end a partir do fim do arquivo

Calculando o **número** de **registros** de um **arquivo** : class Livro anterior

```
void main( )
{
      ifstream fin;                              // cria objeto de leitura
      Livro li;                                  // cria objeto Livro
      fin.open("a:lista.dat");                   // abre arquivo para leitura
      fin.seekg(0,ios::end);                     //coloca ponteiro no fim do arquivo
      long nrec = (fin.tellg( ))/sizeof(Livro);  // calcula num. reg.
      cout << "\nNumero de registros : " << nrec;  // imprime
      cout << "\n\nInsira o numero do registro a exibir ";
      cin >> nrec;
      int posicao = (nrec-1)*sizeof(Livro);      // calcula posição
      fin.seekg(posicao);          // posiciona no registro solicitado
      fin.read((char *)&li, sizeof(Livro));      // lê do arquivo
     li.print();                                 // imprime no vídeo
}
```

Condições de **erro** : Situações de erro ao tratar com arquivos podem ser analisadas através da palavra (int) de status da classe ios, obtida pela função **rdstate( )**. Os bits individuais do valor encontrado podem ser testados pelo operador AND bit-a-bit (&) e os seguintes valores enumerados:

- ios::goodbit Nenhum bit setado. Sem erros
- ios::eofbit Encontrado o fim de arquivo
- ios::failbit Erro de leitura ou gravação
- ios::badbit Erro irrecuperável

A função **clear( )** de protótipo :

void clear(int status=0);

modifica a palavra de status. Se usada sem argumentos, todos os bits são limpos. Do contrário, os bits são setados de acordo com os valores enumerados escolhidos e combinados pelo operador OR (|). Ex.:

clear (ios::eofbit | ios::failtbit);

Outras funções que retornam o status de um bit individual : good(), eof(), fail(), bad(). Veja o exemplo:

```
#include <fstream.h>
void main()
{
        ifstream fin;
        fin.open("xxx.yyy");
        if(!fin)
                cout << "\nNao posso abrir arquivo xxx.yyy\n";
        else
                cout << "\nArquivo aberto com sucesso\n";
        cout << "\nrdstate() = " << fin.rdstate();
        cout << "\ngood() = " << fin.good();
        cout << "\neof() = " << fin.eof();
        cout << "\nfail() = " << fin.fail();
        cout << "\nbad() = " << fin.bad();
 }
```

Execute e observe a saída.

Lendo e gravando de e para a **memória** : As classes iostream interagem com a memória pela mesma sintaxe com a qual interagem com arquivos em disco. Ex.:

```
#include <strstrea.h>              // para funções de strings
```

```
void main()
{
  char buff[200];
  ostrstream str(buff, sizeof(buff));              // cria buffer para gravação
  str << "Este texto esta sendo gravado diretamente na memoria\n";
  str << "por meio de um programa c++\n" << ends;
  cout << buff;                                    // exibe buffer no vídeo
}
```

Observ. : O manipulador **ends** acrescenta o terminador '\0' .

O próximo exemplo cria buffer para leitura e gravação, manipulado com a mesma sintaxe de **cin** e **cout** :

```
#include <strstrea.h>
void main()
{
        strstream str;        // cria buffer para leitura e gravação
        double x, y;
        int i;
        str << "123.45 67.89 123";       // grava
        str >> x;                        // lê
        str >> y;                        // lê
        str >> i;                                // lê
        cout << x << "\n" << y << "\n" << i;   // exibe no vídeo
}
```

Execute e observe a saída.

Sobrecarga dos **operadores** << e >> : Os objetos stream usam os operadores de inserção **(<<)** e de extração **(>>)** com tipos básicos. Vamos sobrecarregá-los para que trabalhem com tipos criados pelo usuário.

O primeiro exemplo sobrecarrega os operadores << e >> para classe **string**:

```
#include <iostream.h>
#include <string.h>
const MAX=80;

class string
{       private :
                char str[MAX];
        public :
                string( )       {       str[0] = '\0';  }
                string (char s[])      {strcpy(str, s);}
```

```
            friend ostream& operator<<(ostream& os, string& s);
            friend istream& operator>>(istream& is, string& s);
  } ;

  ostream& operator<<(ostream& os, string& s)
  {
        os << s.str;
        return os;                                          }
istream& operator>>(istream& is, string& s)
  {
        is.getline(s.str,MAX);
        return is;
  }

  void main( )
  {
        string s;
        cout << "\nLê string com cin e imprime com cout “;
        cout << “\nDigite a string : ";
        cin >> s;
        cout << s;
}
```

Observ.: As funções **operator<<()** e **operator>>()** devem ser **friends** da classes string, visto que os objetos **ostream** (cout) e **istream** (cin) aparecem à esquerda do operador. O valor de retorno é para que o operador possa ser concatenado e imprima (leia) vários objetos de uma única vez.

O segundo exemplo sobrecarrega os operadores << e >> para classe **data** modificada :

```
#include <iostream.h>
class data
{
        private :
                int dia, mes, ano;
        public :
                data() {dia=mes=ano=1;}
                data(int d, int m, int a);

                friend istream& operator>>(istream& is, data& d);
                friend ostream& operator<<(ostream& os, data& d);       } ;
```

```
data::data(int d, int m, int a)
{
        int dmes[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
        ano = a > 0 ? a : 1;
        dmes[2]+=(a%4==0 && a%100 || a%400==0);
        mes = m >= 1 && m <=12 ? m : 1;
        dia = d >= 1 && d <=dmes[mes] ? d : 1;
}

istream& operator>>(istream& is, data& d)
{
        cout << "\nDigite o dia : "; is >> d.dia;
        cout << "\nDigite o mes : "; is >> d.mes;
        cout << "\nDigite o ano : "; is >> d.ano;
  // executa construtor para verificação da entrada
        data d1(d.dia, d.mes, d.ano);
        d=d1;
        return is;
}

ostream& operator<<(ostream& os, data& d)
{
        char nome[13][10]=
        {"Zero","Janeiro","Fevereiro","Marco","Abril","Maio","Junho",
      "Julho","Agosto","Setembro","Outubro","Novembro","Dezembro"};

        os <<d.dia << " de " << nome[d.mes] << " de " << d.ano;
        return os;
}

void main()
{
        data x, y, z(25,12,1999);
        cin >> x >> y;
        cout << x << '\n' << y << '\n' << z;                    }
```

Imprimindo na **impressora** : é semelhante a gravar dados num arquivo em disco. Veja o exemplo :

```
#include <fstream.h>
#include <stdlib.h>

void main(int argc, char **argv)
{
```

```
  const int MAX=80;
   char buff[MAX];

   if(argc != 2)
   {
        cout << "\nForma de uso : C:\\Nome_programa nome_arquivo ";
        exit(1);
   }
        ifstream fin(argv[1]);       // abre arquivo para leitura
         if(!fin)
        {
                cout << "\nNao posso abrir " << argv[1];
                exit(1);
        }

        ofstream oprn("PRN");              // abre arquivo impressora
        while(fin)
        {
                fin.getline(buff, MAX);   // lê do disco
                oprn << buff << '\n';                // imprime na impressora
        }
}
```

Manipuladores e **flags** da **classe ios** : O controle do formato da impressão e leitura pode ser executado por meio dos manipuladores e flags da classe ios. São eles :

Nome	Descrição
skipws	Pula brancos na entrada.
left	Ajusta a saída à esquerda.
right	Ajusta a saída à direita.
internal	Adiciona o caracter de preenchimento do campo após o sinal de menos ou indicação de base, mas antes do valor.
dec	Formato decimal para números.
oct	Formato octal para números.
hex	Formato hexadecimal para números.
showbase	Imprime a base indicada (0x para hex e 0 para octal).
showpoint	Força ponto decimal em valores float e preenche com zeros à direita para completar número de casas decimais.
uppercase	Imprime maiúsculo de A a F para saída hexadecimal e E para notação científica.
showpos	Imprime ‘+’ para inteiros positivos.
scientific	Imprime notação científica para float
fixed	Imprime ponto decimal em valores float

unitbuf	Descarrega stream após cada inserção
stdio	Descarrega stdout e stderr após cada inserção

O próximo exemplo mostra o uso dos flags :

```
#include <iomanip.h>
void main()
{
	int x=0x5fa2;
	float f=123.45;

	for( int i=0; i<2 ; i++)
	{
		cout << "\n\n" << setiosflags(ios::internal) << setw(10) << -x;
		cout << "\n" << resetiosflags(ios::internal) << setw(10) << -x;
		cout << "\n\n" <<setiosflags(ios::hex) << setw(10) << x;
		cout << "\n" <<setiosflags(ios::showbase) << setw(10) << x;
		cout << "\n\n" << setiosflags(ios::dec|ios::fixed) << f;
		cout << setprecision(5);
		cout << "\n" << setiosflags(ios::showpoint) << f;
		cout << "\n----------\n";
		cout.fill('0');
	}
}
```

Execute e observe a saída.

Lendo caracter com **cin** e **gravando** em **disco** : O problema de ler um caracter por vez com **cin** é que cin pula a leitura de espaços em branco que incluem caracteres de tabulação, nova linha e fim de arquivo. Se quisermos ler todos esses caracteres, devemos modificar o flag **skipws**. Ex.:

```
#include <fstream.h>
#include <iomanip.h>
void main()
{
	ofstream fout("a:teste2.txt");
	char ch;

	cout << "Digite o texto (CTRL_Z para encerrar)\n";
	while (!cin.eof())
	{
		cin >> resetiosflags(ios::skipws) >> ch;
```

```
            fout.put(ch);
        }
}
```

Examine o conteúdo do arquivo teste2.txt usando um editor.

Criando nossos próprios manipuladores : Podemos criar novos manipuladores para objetos **stream**. Por exemplo, se o programa for imprimir vários números num único formato, em vez de escrever o formato em cada instrução de impressão, podemos criar um manipulador como mostra o programa :

```
#include <fstream.h>
#include <iomanip.h>

ostream& form(ostream& os)
{
  os.fill('*');
  return os << setiosflags(ios::fixed)
              << setprecision(2)
            << setw(10);
}

void main()
{
        double x=3456.789678, y=9876.543278;
        cout << '\n' << form << x << '\n'<< y;
}
```

Execute e observe a saída.

Exercícios

1) Escreva instruções para:

a) abrir o arquivo ARQ.DOC para **gravação** em **binário**

b) ler um objeto da classe A chamado objA de um objeto de um objeto da classe **ifstream** chamado **fin**.

c) movimentar a posição corrente do objeto **fin** da classe **ifstream** 20 bytes acima.

d) obter a posição corrente do objeto **fin** da classe **ifstream**.

2) Escreva a função de sobrecarga do operador >> que tome dados de um objeto da classe istream e os grave num objeto da classe Facil.

3) Crie um manipulador para objetos ostream que imprima um número float num campo de tamanho 10, três casas decimais e preenchendo os campos não-ocupados com bancos.

4) Escreva um programa que imprima um arquivo na tela de 20 em 20 linhas. O arquivo de entrada deve ser fornecido na linha de comando. A cada impressão de 20 linhas o programa aguarda o pressionamento de uma tecla.

5) Modifique o programa anterior para que aceite mais dois argumentos na linha de comandos, indicando a primeira e a última linha a ser impressa.

Prática XIII

Escreva um programa que criptografa um arquivo usando o operador de complemento bit-a-bit (~). Quando o programa é executado para um arquivo já criptografado, o arquivo é recomposto e volta ao original.