Entrada/Saída em C++

Aluno: Marcelo Giesel
Professores: Humberto Longo e Vagner Sacramento
Universidade Federal de Goiás

Entrada/Saída em C++

- C++ iostream.h no lugar de stdio.h
- Porque mudar?
  - Rotinas de entrada de saída podem ser extendidas para novos tipos declarados pelo usuário.
  - As rotinas são em sua maioria mais fáceis de usar.
  - Alguns aspectos de rotinas podem ser configuradas sem a necessidade e repeti-las.

Visão geral

- Entrada/Saída simples (iostream.h)
  - cout, cin, cerr
  - saída
    - operador de inserção (<<) e encadeamento
  - entrada
    - operador de extração (>>) e encadeamento
  - outras funções de entrada e saída
- Entrada/Saída avançada
  - flags de objetos (setf, unsetf)
  - bits de estatus da entrada
  - manipuladores (iomanip.h)
  - entrada/saída com arquivos (fstream.h)
    - abrindo/fechando arquivos

Utilizando iostream.h

- Inclusão de iostream.h ao invés de stdio.h
- Objetos padrões de iostream:

  cout – conexão com saída padrão

  cin – conexão com entrada padrão

  cerr – conexão com a stream de erro
- Entrada/saída são feitas através destes objetos (ou de um arquivo).

O operador de inserção (<<)

- Para enviarmos algo à saída padrão utilizamos o operador de inserção e o objeto *cout*
- Formato: `cout << Expressao;`
- O compilador reconhece o tipo automaticamente

O operador de inserção (<<) (cont.)

- Em C++

```
cout << 5;  // Outputs 5
cout << 4.1; // Outputs 4.1
cout << "String"; // Outputs String
cout << '\n'; // Outputs a newline
```

- Em C

```
printf("5"); // Outputs 5
printf("4.1"); // Outputs 4.1
printf("String"); // Outputs String
printf('\n'); // Outputs a newline
```

O operador de extração (>>)

- Para receber dados da entrada padrão utilizamos o operador de extração e o objeto *cin*.
- Formato: `cin >> Variavel;`
- Não necessita do &
- O compilador reconhece o tipo faz a leitura automaticamente

O operador de extração (>>) (cont.)

```
int X;
float Y;
```

- Em C++

```
cin >> X; // Leitura de um inteiro
cin >> Y; // Leitura de um float
```

- Em C

```
scanf(“%d”,&X); //Leitura de um inteiro
scanf(“%d”,&Y); //Leitura de um float
```

Operadores << e >>

- Inserções e extrações podem ser encadeadas

```
cout << E1 << E2 << E3 << … ;
cin >> V1 >> V2 >> V3 >> …;
```

- Equivale a realizar várias inserções e extrações seguidas
- Exemplo

```
cout << "Total de vendas $" << sales << '\n';
cin >> Sales1 >> Sales2 >> Sales3;
```

Operadores << e >>

- **>>** e **<<** possuem precedência relativamente alta

  Expressões condicionais e aritméticas devem estar entre parênteses

- Comumente utilizadas como condição de interrupção de laços

```
while (cin >> grade)
```

  Extração retorna 0 (falso) quado EOF é encontrado, e o laço termina

Operadores << e >>

- A função *scanf()* em C pode ler caractere por caractere, seja ele qual for, mas << e >> desconsideram espaços em branco (espaços, caractere de nova linha, tabulações)
- Então o que fazer se preciso destes caracteres?
- Como eu sei se cheguei ou não ao final de uma linha ou do arquivo?

Função *get()*

- Uma solução é o uso de *cin.get()*

  Utilizada sem parâmetros retorna um inteiro, referente ao código ASCII do caractere lido. Lê espaços em branco.

  Semelhante a *getchar()* em C

- O código abaixo lê e imprime quatro caracteres seguidos, independente de quais sejam

```
char c;
  for(int i = 4; i ; i--)    {
        c = cin.get();
        cout << c;    }
```

Função *get()* - usando parâmetros

- *cin.get(caractere)*

  Extrai um caractere da entrada e o armazena em *c*

- *cin.get(array, size, delimitador)*

  Aceita 3 argumentos: array de caracteres, o tamanho limite e um delimitador (opcional, '\n' por padrão)

  Utiliza o *array* como buffer

  O *delimitador* não é consumido da entrada

  O caractere nulo é incluído no final do *array*

Função *getline()*

- *cin.getline(array, size)*

  Funciona como *cin.get(buffer, size)* mas descarta o *delimitador* da stream de entrada e não o armazena no *array*

  Caractere nulo também é inserido no final

- O exemplo abaixo recebe e imprime a linha inteira, juntamente com espaços, e descarta o '\n'

```
char buffer[80];
cout << "Digite seu nome completo:\n";
cin.getline( buffer, 80 );
cout << "\nSeu nome completo eh:\n" << buffer << endl;
```

Funções *peek()*, *putback()* e *ignore()*

- *cin.peek()*

  Não possui parâmetro, e retorna o próximo caractere de entrada sem o excluir da stream

  O ponteiro da stream de entrada não é alterado

- *cin.putback(caract)*

  Coloca *caract* na entrada, tornando-o o próximo caractere a ser recebido

- *cin.ignore(size, delimitador)*

  Extrai da entrada e descarta *size* caracteres ou até encontrar o *delimitador*, que também será descartado

  Por padrão, *size* é 1 e *delimitador* é '\n'

Verificação da stream de entrada

- Em C tenho a palavra reservada EOF para saber que cheguei ao final de um arquivo
- Em C++ temos os bits de controle de streams, que nos informam sobre os estados de erros das mesmas
- É por meio destes bits que podemos saber se uma stream chegou ao final ou não

Estados de erros de streams

- eofbit

  É setado quando um final de arquivo é encontrado numa stream de entrada

  cin.eof() retorna verdadeiro quando final de arquivo for encontrado

- goodbit

  Está setado quando nem *eofbit, badbit* ou *failbit* estão setados

  cin.good() retorna verdadeiro quando as funções *bad, fail* e *eof* retornam falso

  Muito utilizado em condição de interrupção

```
while( cin.good() )
```

Funções de saída

- Algumas funções de saída também são bem úteis
- *put(caract)*

  Utilizando *cout.put(caract)* o caractere passado como parâmetro é colocado na stream de saída

  O ponteiro da saída é incrementado

- *write(buffer, size)*

  O valor em *buffer* (array de caracteres) é escrito na saída até *size* caracteres escritos

  Se encontrar um valor nulo, ele também é escrito na saída

Configurando as flags de formatação

- O objeto *cout* possui *flags* que definem como as informações devem ser exibidas. Tais *flags* podem ser configuradas
- Utilizamos *setf* para configuras essas *flags*. O mesmo é válido para *cin*.

- Chamando *setf:*

```
cout.setf(flags)
```

A função *setf* é um campo do objeto *cout* e *cin*

Base de inteiros e flags de formato

Escolhendo a base para imprimir um inteiro

- Flags a serem usadas:
  - ios::dec – mostra em decimal (padrão)
  - ios::oct – mostra em octal
  - ios::hex – mostra em hexadecimal
- Apenas uma deve estar setada a cada vez
- Para mudar, use:

```
cout.unsetf(ios::dec);
cout.unsetf(ios::oct);
cout.unsetf(ios::hex);
cout.setf(ios::oct);
```

Base de inteiros e flags de formato

Podemos combinar as flags usando o operador |

```
cout.unsetf(ios::dec | ios::oct | ios::hex);
cout.setf(ios::oct);
```

ou

```
cout.setf(ios::oct, ios::dec | ios::oct | ios::hex);
```

C++ ainda inclui um shorthand para a segunda combinação de flags: ios::basefield:

```
cout.setf(ios::oct,ios::basefield);
```

Desliga todas as flags e liga apenas a octal

Exemplo de base de inteiros

```
int x = 42;

cout.setf(ios::oct,ios::basefield);
cout << x << '\n'; // Outputs 52\n
cout.setf(ios::hex,ios::basefield);
cout << x << '\n'; // Outputs 2a\n
cout.setf(ios::dec,ios::basefield);
cout << x << '\n'; // Outputs 42\n
```

Mostrando o sinal de adição

A flag ios::showpos pode ser ligada para imprimir o sinal + caso um número seja positivo

```
int x = 42;
int y = 3.1415;

cout.setf(ios::showpos);
cout << x << '\n'; // Outputs +42\n
cout << y << ‘\n’; // Outputs +3.1415\n
```

Configurando a largura

- Você pode utilizar a função *width(int)* para configurar a largura de uma área a ser impressa

```
int x = 42;

cout.width(5);
cout << x << '\n';  // Outputs    42
cout << x << '\n';  // Outputs 42
```

Formato do float

Valores de ponto flutuante são impressos nas formas fixas e científicas

```
cout << 2.3;  // Outputs 2.3
cout << 5.67e8; // Outputs 5.67e+08
cout << 0.0;  // Outputs 0
```

Dígitos significantes em Float

Use a função *precision(int)* para configurar o número de dígitos significativos impressos

```
float y = 23.1415;
cout.precision(1);
cout << y << '\n'; // Outputs 2e+01
cout.precision(2);
cout << y << '\n'; // Outputs 23
cout.precision(3);
cout << y << '\n'; // Outputs 23.1
```

Formato de pontos flutuantes

- Podemos utilizar as flags ios::scientific e ios::fixed para forçar a saída de um ponto flutuante para a forma fixa ou científica
- Apenas uma flag por vez

```
cout.setf(ios::scientific,ios::floatfield);
cout << 123.45 << '\n'; // Outputs 1.2345e+02
cout.setf(ios::fixed,ios::floatfield);
cout << 5.67E1 << '\n'; // Outputs 56.7
```

- O efeito da precisão depende do formato
  - scientific (total de dígitos significativos)
  - fixed (dígitos após o ponto)

Manipulators

- Isn't that all kind of involved??
  - Plus, what's that with width only counting for one arg?
- A solution - manipulators
  - A manipulator is a simple function that can be included in an insertion or extraction chain
- C++ manipulators
  - must include iomanip.h to use
  - several are provided to do useful things
  - you can also create your own (see 17.3, 17.5, 17.6, 17.8)

Manipuladores de saída (sem argumentos)

Manipuladores são incluídos como argumentos na extração

- endl – imprime um caractere de nova linha, flushes output
- dec – configura a saída de inteiros para decimal
- hex – configura a saída de inteiros para hexadecimal
- oct – configura a saída de inteiros para octal

Exemplo:

```
#include <iostream.h>
#include <iomanip.h>
int x = 42;
cout << oct << x << endl; // Outputs 52\n
cout << hex << x << endl; // Outputs 2a\n
cout << dec << x << endl; // Outputs 42\n
```

Manipuladores de saída (1 arg)

Manipuladores com 1 argumento

- setw(*int*) - seta o tamanho para o valor de *int*
- setfill(*char*) - seta o caractere de preenchimento *char*
- setprecision(*int*) - seta a precisão para *int*
- setbase(*int*) - sets a saída para hex se *int é* 16, oct se *int é* 8, dec se *int é* 0 ou 10
- setiosflags(*flags*) - seta *flags* on
- resetiosflags(*flags*) - seta *flags* off

```
cout << resetiosflags(ios::floatfield) <<
   setiosflags(ios::fixed | ios::showpoint) <<
   setw(7) << setprecision(2) << setfill('_') <<
   34.267 << endl; // outputs __34.27
```

Input/Output com arquivo

- É feito com as mesmas operações (inserção e extração)
- Simplesmente abra um objeto de saída ou entrada e utilize-o como se fosse cin ou cout
- Para usar
  - include <fstream.h>
  - Crie um objeto de entrada do tipo *ifstream*
  - Ou objeto de saída do tipo *ofstream*

Abrindo arquivos

- Utilize a função *open* ou inclua o nome do arquivo quando declarar a variável

```
ifstream inobj1;
inobj1.open("in1.dat")
ifstream inobj2("in2.dat");
```

- Verifique com a condição abaixo para ver se o arquivo foi aberto com sucesso

```
if (!inobj1)
  cout << "Unable to open file in1.dat" << endl;
```

Abrindo arquivos de saída

A rotina padrão abre o arquivo como “w” em fopen de C

- Arquivo existente será deletado

Flags para outras opções na hora de abrir o arquivo

ios::out – abre como conexão de saída (deve incluir)

ios::append – anexa a um arquivo existente

ios::nocreate – o arquivo deve existir ou da erro

ios::noreplace – o arquivo não deve existir, ou erro

Exemplo

```
ofstream out(“outf”,ios::out | ios::append);
// out é um anexo de outf
```

Fechando arquivos

Utilize *close()* nos objetos para fechar a conexão com os arquivos

```
ifstream in(“in.dat”);
…
in.close();
```

Exemplo com arquivo

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

void main() {
  char infname[101];
  char outfname[101];
  char buffer[101];

  cout << ”File to copy from: ";
  cin >> infname;
  ifstream in(infname);
  if (!in) {
    cout << "Unable to open " << infname << endl;
    exit(0);
  }
```

Exemplo com arquivo

```
  cout << "File to copy to: ";
  cin >> outfname;
  ofstream out(outfname,ios::out |
  ios::noreplace);
  if (!out) {
    cout << "Unable to open " << outfname << " --
  already exists!" << endl;
    exit(0);
 }
  in.getline(buffer,100);
  while (!in.eof()) {
    out << buffer << endl;
    in.getline(buffer,100);
  }
  in.close();
  out.close();
}
```