Universidade de São Paulo
Brasil

Aula 16
Interfaces (Continuação)

MAC0216 - Técnicas de Programação I

Professores: Alfredo, Daniel, Fabio e Kelly

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

Princípios de Interfaces

Princípios de uma Interface

Uma boa interface de biblioteca deve ser:

- **Unificada** – possuir um tema que unifique suas funções
- **Simples** – procurar esconder a complexidade de suas implementações
- **Suficiente** – prover as funcionalidades necessárias para satisfazer as necessidades dos usuários
- **Genérica** – ser suficientemente flexível para atender as necessidades de diferentes tipos de usuários
- **Estável** – manter a estrutura e efeito de suas funções, mesmo quando as implementações são modificada

Para projetar boas interfaces:

- Oculte os detalhes de implementação
  - encapsulamento, abstração, modularização
- Escolha um conjunto ortogonal pequeno de funções
- Não aja pelas costas do usuário
- Faça uma mesma coisa de forma igual em todos os lugares em que ela aparecer

Ocultação dos detalhes de implementação

A implementação por trás de uma interface deve ficar oculta, de modo que ela possa mudar sem afetar os sistemas que a usam

- Evite o uso de variáveis globais
  - Sempre que possível, passe os dados por meio de parâmetros para funções
- Não use dados que estão sempre “visíveis”
  - É difícil manter a consistência dos valores quando usuários podem alterar variáveis de forma indiscriminada
- Classes (de orientação a objetos) são um ótimo mecanismo para esconder informações

Escolha de um conjunto ortogonal pequeno de funções

A interface deve prover tantas funcionalidades quanto o necessário. Funções não devem se sobrepor muito em suas funcionalidades

- Ter muitas funções pode tornar uma biblioteca mais fácil de ser usada, mas mais difícil de ser escrita e mantida
- Interfaces enormes são difíceis de ser “aprendidas” pelos usuários

Escolha de um conjunto ortogonal pequeno de funções

- Não caia na tentação de incluir em sua interface funções que fornecem formas variadas de se fazer a mesma coisa
  - Funções da libc para a escrita de um caracter em um stream – putc, fputc, fprintf, fwrite
- Lembre-se da filosofia do Unix
  - “faça uma só coisa e faça-a bem feita”
- Não adicione itens a uma interface porque é possível fazê-lo e nem adicione-os para corrigir falhas na implementação

This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

de Doug McIlroy, inventor do conceito de pipes do Unix (trecho presente no livro The Art of Unix Programming, de E. S. Raymond)

Não agir pelas costas do usuário

- Uma biblioteca não deve criar variáveis de ambiente ou arquivos secretos, nem mudar dados globais
- Ela deve ser cuidadosa ao mudar dados de seu chamador
- Uma biblioteca não deve requerer outra apenas para a conveniência do projetista da interface ou do implementador
- É desejável que a biblioteca seja auto-contida
  - Quando isso não for possível, é preciso deixar explícito os serviços externos que ela requer

Não agir pelas costas do usuário

- Mau exemplo: função **strtok**, da string.h, que quebra uma string em uma sequência de tokens

  char *strtok(char *s, const char *delim);

- Devolve um ponteiro para o primeiro token em p que não contém um caracter em s
- Na primeira chamada à função, s é a string a ser varrida
- Nas chamadas subsequentes, deve-se usar NULL como valor para indicar que a varredura deve continuar de onde parou na chamada anterior

- **Mantém dados entre chamadas, impossibilitando que chamadas intercaladas sejam feitas**
- **Ela escreve NULLs no meio da string de entrada**

Fazer uma mesma coisa igual em todos os lugares

Consistência e regularidade são importantes

Coisas relacionadas devem ser alcançadas por meios relacionados

- Bom exemplo: funções básicas str... de C
  - Se comportam de forma parecida – dados fluem da direita para a esquerda nos parâmetros e devolvem a string resultante
- Mau exemplo: funções da biblioteca padrão de E/S de C
  - É difícil prever a ordem dos parâmetros das funções – algumas possuem o parâm. FILE* primeiro; outras, por último
  - Algumas pedem a definição de um limite máximo de bytes; outras, não

Gerenciamento de recursos

É um dos problemas mais difíceis de se lidar no projeto de interfaces

- Como gerenciar recursos que são de propriedade da biblioteca ou que são compartilhados pela biblioteca com aqueles que vão chamá-la?
  - Exemplos de recursos: memória, arquivos, estado de variáveis
  - Problemas: inicialização, manutenção do estado, compartilhamento e cópia, e limpeza

Gerenciamento de recursos

A liberação de um recurso deve ser feita na mesma camada em que ele foi alocado

O estado da alocação de um recurso não deve ser alterado através da interface

- Exemplo 1: se uma função da interface recebe como entrada um arquivo aberto, então ela deve deixá-lo aberto quando for encerrada
- Exemplo 2: gerenciamento de memória com garbage collector (coleta automática de “lixo”)

Tratamento de erros

O que fazer na ocorrência de um erro irrecuperável?

Possibilidades de tratamento:

- Mostrar uma mensagem contendo detalhes sobre o erro ocorrido e sair do programa
- Assinalar o erro e dar uma chance ao chamador de se recuperar
- Registrar a mensagem de erro em um arquivo de log
  - Bom para os casos em que a biblioteca pode estar sendo executada em um ambiente em que a mensagem interferiria nos dados mostrados pelo chamador

Tratamento de erros

Detectar erros num nível baixo; lidar com eles num nível alto

- O chamador é quem deve determinar a forma como o erro deve ser tratado
- As rotinas da biblioteca devem, em casos de erro, falhar de forma “graciosa”
  - Não abortar o código
  - Retornar detalhes suficientes sobre o erro, para que o chamador possa fazer um tratamento apropriado
- Ex.: a função getchar devolve um valor (o EOF) que não é caractere quando o fim do arquivo é encontrado ou em caso de erro

Tratamento de exceções

Usar exceções somente nas situações excepcionais

- Algumas linguagens possuem o conceito de *exceções* para capturar situações não usuais e se recuperar delas
- Elas permitem que um fluxo de controle alternativo seja executado quando algo errado aconteça
- Use exceções com parcimônia:
  - Não use-as para tratar valores de retorno esperados (como o EOF na leitura de um arquivo)
  - Como elas distorcem o fluxo de controle, podem conduzir a construções confusas e mais susceptíveis a erros

Interfaces com usuários

- Erros devem ser detectados e reportados
    - Recuperação deve ser tentada quando cabível
    - Mensagens devem ser tão informativas quanto possível (indicando a causa do erro)
- O texto das mensagens de erro, do prompt e das caixas de diálogo devem expressar o formato dos dados de entrada válidos
    - Objetivo: conduzir o usuário ao uso correto do programa
- Princípios de estilo que contribuem para a criação de interfaces (textuais ou gráficas) fáceis de se usar:
    - simplicidade, clareza, regularidade, familiaridade, restrição

Exemplo de Projeto de Interface em C

Geração de números pseudo-aleatórios

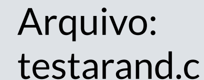
- A interface stdlib.h possui a função

  int rand(void);

  que devolve um inteiro pseudo-aleatório maior ou igual a zero e menor ou igual a RAND_MAX

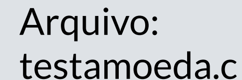
- RAND_MAX é uma constante definida em stdlib.h e que equivale ao maior número inteiro armazenável no sistema

Programa para teste da função rand




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

#define NTentativas 10

int main()
{
   int i, r;

   printf("Neste computador, RAND_MAX = %d.\n", RAND_MAX);
   printf("Eis os resultados de %d chamadas a rand:\n", NTentativas);
   for (i = 0; i < NTentativas; i++) {
       r = rand();
       printf("%10d\n", r);
   }
   return 0;
}
```

Programa para lançamento de moedas




```c
#include <stdio.h>
#include <stdlib.h>

#define NTentativas 10

int main()
{
    int i;
    for (i = 0; i < NTentativas; i++) {
        if (rand() <= RAND_MAX / 2)
            printf("Cara\n");
        else
            printf("Coroa\n");
    }
    return 0;
}
```

Função para o lançamento de um dado

```
int LancaDado()
{
   int r = rand();
   if (r < RAND_MAX / 6)
       return (1);
   else if (r < RAND_MAX / 6 * 2)
       return (2);
   else if (r < RAND_MAX / 6 * 3)
       return (3);
   else if (r < RAND_MAX / 6 * 4)
       return (4);
   else if (r < RAND_MAX / 6 * 5)
       return (5);
   else
       return (6);
}
```

E se fosse o sorteio de uma carta de baralho (número inteiro em [1..52])?

Generalizando o problema

Geração de um número inteiro pseudo-aleatório em um dado intervalo

```
/* Esta função primeiro obtém um inteiro aleatório
 * no intervalo [0..RAND_MAX] e depois converte-o
 * em um número no intervalo [min..max] aplicando
 * os seguintes passos:
 * (1) Gera um número real entre 0 e 1.
 * (2) Escala-o para o tamanho apropriado de intervalo.
 * (3) Trunca o valor para um inteiro.
 * (4) Traduz para o ponto de início apropriado.
*/
int InteiroAleatorio(int min, int max)
{
   int k;
   double d;
   d = (double) rand() / ((double) RAND_MAX + 1);
   k = (int) (d * (max - min + 1));
   return (min + k);
}
```

Salvando ferramentas em bibliotecas

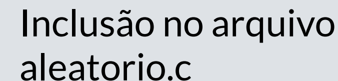
Versão preliminar da interface

```
#ifndef _aleatorio_h
#define _aleatorio_h

/*
* Função: InteiroAleatorio
* Uso: n = InteiroAleatorio(min, max);
* ------------------------------------
* Esta função devolve um inteiro aleatório no intervalo
* fechado [min .. max].
*/
int InteiroAleatorio(int min, int max);

#endif
```

Expandindo a biblioteca

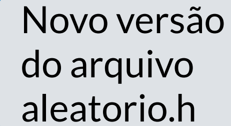
Geração de um número real pseudo-aleatório em um dado intervalo

```
/*
* Função: RealAleatorio
* --------------------
* A implementação de RealAleatorio é similar a do
* InteiroAleatorio, mas sem o passo da truncagem.
*/
double RealAleatorio(double min, double max)
{
   double d;

   d = (double) rand() / ((double) RAND_MAX + 1);
   return (min + d * (max - min));
}
```



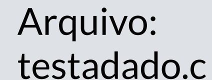
Atualizando a interface

```
#ifndef _aleatorio_h
#define _aleatorio_h

/* Função: InteiroAleatorio
 * Uso: n = InteiroAleatorio(min, max);
 * ------------------------------------
 * Esta função devolve um inteiro aleatório no intervalo
 * fechado [min .. max]. */
int InteiroAleatorio(int min, int max);

/* Função: RealAleatorio
 * Uso: d = RealAleatorio(min, max);
 * ---------------------------------
 * Esta função devolve um número real aleatório no intervalo
 * semi-fechado [min .. max), significando que o resultado é
 * sempre maior ou igual a min, mas estritamente menor que max. */
double RealAleatorio(double min, double max);

#endif
```



Construindo um programa cliente

```c
#include <stdio.h>
#include "aleatorio.h"
#define NTentativas 10

int LancaDado(void)
{
   return (InteiroAleatorio(1, 6));
}

int main()
{
   int i;
   for (i = 0; i < NTentativas; i++)
       printf("%d\n", LancaDado());
   return 0;
}
```



Outras expansões da biblioteca

```
/** Função: Aleatorize
* Uso: Aleatorize();
* --------------------
* Esta função indica a semente para o rand de forma que a
* sequência aleatória seja imprevisível. Durante a fase de
* depuração, é melhor não chamar esta função. Assim, o
* comportamento do programa será reprodutível.*/
void Aleatorize(void);

/** Função: SorteAleatoria
* Uso: if (SorteAleatoria(p)) . . .
* ---------------------------------
* A função SorteAleatoria devolve true com a probabilidade
* indicada por p, que deve ser um número real entre 0
* (significando nunca) e 1 (significando sempre). Por
* exemplo, a chamada SorteAleatoria(.30) devolve true
* 30% das vezes.*/
bool SorteAleatoria(double p);
```

Inclusão no arquivo aleatorio.h

A implementação dessas novas funções devem ser incluídas no arquivo aleatorio.c

Versão final da biblioteca *aleatorio*

Para ver a implementação final da biblioteca do exemplo, consultar arquivos aleatorio.h e aleatorio.c disponibilizados com o material da aula

Referências Bibliográficas

- Capítulo 4 (Interfaces), do livro:

  B.W. Kernighan e R. Pike, ***A Prática da Programação***, Campus, 1999.

- Capítulos 7 (Libraries and Interfaces) e 8 (Designing Interfaces) do livro:

  Roberts, ***The Art and Science of C***, Addison-Wesley, 1995.

Template de slides do site SlidesCarnival