

[MAC0211] Laboratório de Programação I

Aula 14

Interfaces (Exemplos)

Automação de Compilação de Programas e Bibliotecas

Alair Pereira do Lago

DCC-IME-USP

16 de abril de 2015

Exemplo de projeto de interface em C

Geração de números pseudo-aleatórios

- ▶ A interface `stdlib.h` possui a função `rand`:

```
int rand(void);
```

que devolve um inteiro pseudo-aleatório maior ou igual a zero e menor ou igual a `RAND_MAX`, constante definida em `stdlib.h` e que equivale ao maior número inteiro armazenável no sistema.

Exemplo de projeto de interface em C

Programa para teste do rand (testarand.c)

```
#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;
}
```

Exemplo de projeto de interface em C

Programa para lançamento de moedas (testamoeda.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;
}
```

Exemplo de projeto de interface em C

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])?

Exemplo de projeto de interface em C

Generalizando o problema: geração de um número inteiro pseudo-aleatório em um dado intervalo

```
/* Esta funcao primeiro obtem um inteiro aleatorio
 * no intervalo [0..RAND_MAX] e depois converte-o
 * em um numero no intervalo [min..max] aplicando
 * os seguintes passos:
 * (1) Gera um numero 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 inicio apropriado.
 */

int InteiroRandomico(int min, int max)
{
    int k;
    double d;

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

Exemplo de projeto de interface em C

Salvando ferramentas em bibliotecas – versão preliminar de `random.h`

```
#ifndef _random_h
#define _random_h

/*
 * Funcao: InteiroRandomico
 * Uso: n = InteiroRandomico(min, max);
 * -----
 * Esta funcao devolve um inteiro aleatorio no intervalo
 * fechado [min .. max].
 */

int InteiroRandomico(int min, int max);

#endif
```

Exemplo de projeto de interface em C

Expandindo a biblioteca: geração de um número real pseudo-aleatório em um dado intervalo

```
/*  
 * Funcao: RealRandomico  
 * -----  
 * A implementacao de RealRandomico eh similar a do  
 * InteiroRandomico, sem o passo da truncagem.  
 */  
  
double RealRandomico(double min, double max)  
{  
    double d;  
  
    d = (double) rand() / ((double) RAND_MAX + 1);  
    return (min + d * (max - min));  
}
```


Exemplo de projeto de interface em C

Atualizando a interface `random.h`

```
#ifndef _random_h
#define _random_h

/* Funcao: InteiroRandomico
 * Uso: n = InteiroRandomico(min, max);
 * -----
 * Esta funcao devolve um inteiro aleatorio no intervalo
 * fechado [min .. max]. */

int InteiroRandomico(int min, int max);

/* Funcao: RealRandomico
 * Uso: d = RealRandomico(min, max);
 * -----
 * Esta funcao devolve um numero real aleatorio no intervalo
 * semi-fechado [min .. max), significando que o resultado eh
 * sempre maior ou igual a min, mas estritamente menor que max */

double RealRandomico(double min, double max);

#endif
```

Exemplo de projeto de interface em C

Construindo um programa cliente (testadado.c)

```
#include <stdio.h>
#include "random.h"

#define NTentativas 10

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

int main()
{
    int i;

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

Exemplo de projeto de interface em C

Expansão da biblioteca `random`

- ▶ Inclusão da função `Randomize`
- ▶ Inclusão da função `SorteRandomica`
- ▶ Para ver a implementação final da biblioteca, consultar arquivos `random.h` e `random.c` na página do curso

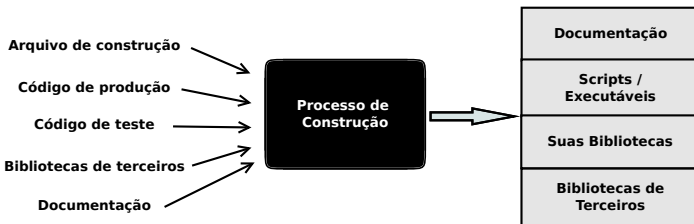
Escrita de software × Construção de software

- ▶ **Escrita de software**: é um processo que envolve arte, ciência e engenharia. Uma sessão de escrita é uma experiência única e irreprodutível
- ▶ **Construção (*built*) de software**: é como fazer linguça – o código fonte é moído em pedaços, para ser consumido pelo computador (e geralmente não estamos interessados nos detalhes envolvidos nesse processo). É um processo repetitivo, passível de automação

Automação de construção

(também conhecida como “automação da compilação”)

- ▶ A **automação de construção** automatiza o processo de construção de um software
- ▶ O **arquivo de construção** (*build file*) contém o passo-a-passo de tudo que tem que ser executado para o construção de um programa. Ele lista os itens envolvidos nesses passos (como os códigos fontes, arquivos de configuração, bibliotecas, etc.)
- ▶ Esse arquivo pode ser criado manualmente pelo desenvolvedor ou pode ser gerado de forma automática por algumas ferramentas



Processo de construção

Ferramentas para a automação de compilação

- ▶ **GNU Make** – bastante usada para código em C, C++, Latex. Mas pode ser usada com qualquer outra linguagem cujo compilador possa ser executado com um comando *shell*
- ▶ **Apache Ant** – usada principalmente para código Java (mas também funciona para C, C++, ...)
- ▶ **Apache Maven** – usada em projetos em Java (entretanto, é mais do que um gerenciador de compilação; ela auxilia no gerenciamento do projeto, publicando informações do projeto e possibilitando o compartilhamento de jars entre projetos)
- ▶ **MSBuild (Microsoft)** – funciona conjuntamente com o Visual Studio (Visual Basic, Visual C++, C#, ...)

GNU Make

- ▶ É um utilitário que determina automaticamente quais pedaços de um grande programa precisam ser recompilados e dispara os comandos que os recompilam
- ▶ Pode ser usado com qualquer linguagem de programação cujo compilador possa ser executado com um comando *shell*
- ▶ Não se limita à construção de programas. Pode ser usado para descrever qualquer tarefa em que alguns arquivos precisam ser atualizados automaticamente a partir de outros sempre que houver alterações nesses outros

Makefile

- ▶ para usar o *Make*, é preciso escrever um arquivo chamado *makefile*, que descreve os relacionamentos entre os arquivos do seu programa e indica comandos para atualizar cada arquivo. **Exemplo:** em um programa, o arquivo executável é construído a partir de arquivos objetos que, por sua vez, são gerados a partir de código-fonte. O *makefile* pode indicar como os fontes são compilados e ligados para gerar o executável.
- ▶ Uma vez que um *makefile* correto exista, basta executar o comando `make` para que todas as recompilações necessárias para a atualização do programa sejam executadas
- ▶ O *Make* usa as informações contidas no *makefile* e os horários da última modificação dos arquivos para decidir quando um arquivo precisa ser atualizado. O próprio *makefile* indica como a atualização deve ser feita

Uma introdução a *Makefiles*

Regras

- ▶ Um *makefile* é composto por regras do tipo:

```
alvo ... : pré-requisitos ...  
receita  
...  
...
```

- ▶ **alvo**: geralmente é o nome de um arquivo que é gerado por um programa (ex.: arquivos executáveis ou objeto). Mas pode ser também o nome de uma ação a ser executada
- ▶ **pré-requisito**: é um arquivo que é usado como entrada na criação do *alvo*. Um alvo pode depender de vários arquivos
- ▶ **receita**: ação que o *make* executará. Pode possuir um ou mais comandos (na mesma linha ou um em cada linha). **Importante**: é preciso colocar um caractere de tab no início de cada linha da receita

Uma introdução a *Makefiles*

Regras

- ▶ Em um regra, uma receita geralmente serve para criar o arquivo alvo quando há alteração em algum dos arquivos pré-requisitos.
- ▶ Mas uma regra não é obrigada a ter pré-requisitos (veremos um exemplo disso a seguir)
- ▶ Um makefile pode conter outras coisas além das regras. Mas um makefile simples só precisa delas para ser feito

Um *makefile* simples (do editor de texto)

```
edit : main.o kbd.o command.o display.o \  
      insert.o search.o files.o utils.o  
gcc -o edit main.o kbd.o command.o display.o \  
      insert.o search.o files.o utils.o  
  
main.o : main.c defs.h  
      gcc -c main.c  
kbd.o : kbd.c defs.h command.h  
      gcc -c kbd.c  
command.o : command.c defs.h command.h  
      gcc -c command.c  
display.o : display.c defs.h buffer.h  
      gcc -c display.c  
insert.o : insert.c defs.h buffer.h  
      gcc -c insert.c  
search.o : search.c defs.h buffer.h  
      gcc -c search.c  
files.o : files.c defs.h buffer.h command.h  
      gcc -c files.c  
utils.o : utils.c defs.h  
      gcc -c utils.c  
clean :  
      rm edit main.o kbd.o command.o display.o insert.o search.o files.o utils.o
```

Um *makefile* simples – funcionamento

- ▶ O caractere ‘\’ é usado para dividir uma linha longa
- ▶ Para gerar o executável chamado `edit`, basta executar `make`
- ▶ Para apagar o arquivo executável e todos os arquivos objeto do diretório, executar `make clean`

Observações:

- ▶ Por padrão, o *Make* começa com o primeiro alvo do *makefile*. Ele é chamado de meta padrão (default goal). Metas são alvos que o *Make* se esforça para atualizar por último.
- ▶ Uma outra regra do *makefile* só é processada de forma automática no `make` se seu alvo aparece como pré-requisito para a meta ou se é pré-requisito para alguma outra regra da qual a meta depende

Makefile – funcionamento

- ▶ Quando o alvo é um arquivo, ele precisa ser recompilado ou religado se um dos seus pré-requisitos mudam
- ▶ Um pré-requisito que é automaticamente gerado (ou seja, que é o alvo de uma regra também), deve ser atualizado antes de ser usado
- ▶ Por exemplo: `edit` depende de cada um dos 8 arquivos objeto. O objeto `main.o` depende do fonte `main.c` e o cabeçalho `def.h`
- ▶ O alvo `clean` não é um arquivo; é apenas o nome de uma ação. Ela não é pré-requisito para nenhuma outra regra e também não tem pré-requisitos. Por essa razão, a regra do `clean` só será executada pelo `make` quando isso for solicitado explicitamente.
- ▶ Alvos que são apenas ações (e não arquivos) são chamados de *alvos falsos* (*phony targets*)

O uso de variáveis em *makefiles*

- ▶ *Variáveis* nos ajudam a evitar replicações no *makefile* e, consequentemente, diminuem a possibilidade de introdução de erros durante a manutenção

- ▶ Exemplo: a sequência de arquivos

```
main.o kbd.o command.o display.o insert.o search.o  
files.o utils.o
```

aparece mais de uma vez no *makefile* do editor de texto

- ▶ É uma prática frequente definir em um *makefile* uma variável de nome *OBJECTS*, *objs*, *OBJS*, *obj*, ou *OBJ* contendo a lista de todos os nomes dos arquivos objeto:

```
objects = main.o kbd.o command.o display.o insert.o \  
          search.o files.o utils.o
```

Um *makefile* simples (agora com variáveis)

```
objects = main.o kbd.o command.o display.o insert.o search.o files.o utils.o

edit : $(objects)
    gcc -o edit $(objects)

main.o : main.c defs.h
    gcc -c main.c
kbd.o : kbd.c defs.h command.h
    gcc -c kbd.c
command.o : command.c defs.h command.h
    gcc -c command.c
display.o : display.c defs.h buffer.h
    gcc -c display.c
insert.o : insert.c defs.h buffer.h
    gcc -c insert.c
search.o : search.c defs.h buffer.h
    gcc -c search.c
files.o : files.c defs.h buffer.h command.h
    gcc -c files.c
utils.o : utils.c defs.h
    gcc -c utils.c

clean :
    rm edit $(objects)
```

Bibliografia e materiais recomendados

- ▶ Capítulo 4 – *Interfaces*, do livro *A Prática da Programação*, de B.W. Kernighan e R. Pike
(disponível no Peca, na página do curso)
- ▶ Capítulos 7 (*Libraries and Interfaces*) e 8 (*Designing Interfaces*) do livro *The Art and Science of C*, de E.S. Roberts
- ▶ Manual do GNU Make
<http://www.gnu.org/software/make/manual/>
- ▶ Projeto Apache Ant – <http://ant.apache.org/>
- ▶ Projeto Apache Maven – <http://maven.apache.org/>
- ▶ Livro: *Pragmatic Project Automation How to Build, Deploy, and Monitor Java Applications*, de Mike Clark
- ▶ Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
<http://www.ime.usp.br/~kon/MAC211>

Materiais recomendados sobre Latex

- ▶ Página do projeto: <http://www.latex-project.org/>
- ▶ Instalação no Linux: pacote `texlive-latex-base`
- ▶ Editores para Latex:
 - ▶ *Texmaker* (pacote `texmaker`), *TeXstudio* (pacote `texstudio`), *Kile* (pacote `kile`)
 - ▶ Tabela comparativa: http://en.wikipedia.org/wiki/Comparison_of_TeX_editors
- ▶ *A Not so short introduction to Latex*:
<http://mirrors.ctan.org/info/lshort/english/lshort.pdf>
Versão em português de Portugal:
<http://mirrors.ctan.org/info/lshort/portuguese/pt-lshort.pdf>
- ▶ *Wiki Guide*: <http://en.wikibooks.org/wiki/LaTeX>
- ▶ Um guia bem curto para iniciantes:
<https://www.cs.princeton.edu/courses/archive/spring10/cos433/Latex/latex-guide.pdf>

Cenas dos próximos capítulos...

Na próxima aula:

- ▶ GNU Make (continuação)
- ▶ Analisadores léxicos