Paradigmas de Programação II
Departamento de Informática
Universidade do Minho

LESI 1º Ano MCC

2002/2003

Ficha Teórico-Prática Nº 0
Ambiente (Unix) de Programação em C

24 de Fevereiro de 2003

1 Funcionamento das aulas Teórico-Práticas

As aulas teórico-práticas da diciplina Paradigmas de Programação II são baseadas na realização de uma ficha teórico-prática por aula, disponíveis no site da disciplina, em `http://www.di.uminho.pt/~prh/curpp202.html`.

Todos os alunos devem resolver as fichas práticas no computador, fora das aulas. Nas aulas, na primeira hora são comentadas as resoluções (que os alunos devem trazer em disquete) e na segunda hora são discutidas possíveis soluções para os exercícios da ficha seguinte.

2 Ambiente de Programação em C

Para a resolução dos programas apresentados nas fichas de PP II, aconselham-se todos os alunos a criar um ambiente de trabalho no computador que permita editar, compilar e executar os programas C, bem como preparar a documentação associada aos programas desenvolvidos.

2.1 Fórum PPII

(Já foi pedido ao CI a criação de um fórum Web para a disciplina de PPII. Todos os alunos ficam automaticamente inscritos, com o respectivo email da UM. Assim que o fórum estiver disponível, serão dadas mais informações.)

2.2 Comentários em C

Exercício 1 *Diga quais as formas permitidas para misturar comentários com o texto dos programas, na linguagem C.*

2.3 Primeiro programa C

Exercício 2 *Desenvolva um programa C que permita escrever um algarismo por extenso, em alemão, de acordo com a seguinte tabela:*

Alemão	*Português*
null	*zero*
eins	*um*
zwei	*dois*
drei	*três*
vier	*quatro*
fünf	*cinco*
sechs	*seis*
sieben	*sete*
acht	*oito*
neun	*nove*

Resolução

```
#include <stdio.h>

int main() {
        int k;
        printf("Introduza um algarismo: ");
        scanf("%d", &k);
        switch(k) {
                case 0: printf("null\n"); break;
                case 1: printf("eins\n"); break;
                case 2: printf("zwei\n"); break;
                case 3: printf("drei\n"); break;
                case 4: printf("vier\n"); break;
                case 5: printf("funf\n"); break;
                case 6: printf("sechs\n"); break;
                case 7: printf("sieben\n"); break;
                case 8: printf("acht\n"); break;
                case 9: printf("neun\n"); break;
                default: printf("Algarismo desconhecido.\n");
        }
}
```

2.4 Compilação separada em C

Exercício 3 *Diga, sucintamente, o que faz cada uma das seguintes opções do CC:*

1. `cc -E`

   Invoca o pré-processador; o resultado é um novo texto resultante da plicação do pré-processador.

2. `cc -S`

   Compila o programa, resultando um ficheiro ainda em assembly (ou seja, por assemblar). O resultado é inteligível.

3. `cc -c`

   Compila e converte o assembly em código máquina, mas não invoca o link.

4. `cc -o` *ou só* `cc`

   Compila, assembla e linka. Gera um arquivo pronto a executar.

Escreva um novo programa C, com a mesma funcionalidade do programa anterior, mas em que:

1. o palavra alemã é calculada dentro de uma função alemao() que recebe como argumentos o número e a string onde será guardada a palavra;

2. a função é escrita num ficheiro `extenso.c`:

```
#include "extenso.h"

char *alemao(char *res, int algarismo) {
        switch(algarismo) {
                case 0: strcpy(res,"null"); break;
                case 1: strcpy(res,"eins"); break;
                case 2: strcpy(res,"zwei"); break;
                case 3: strcpy(res,"drei"); break;
                case 4: strcpy(res,"vier"); break;
                case 5: strcpy(res,"üfnf"); break;
                case 6: strcpy(res,"sechs"); break;
                case 7: strcpy(res,"sieben"); break;
                case 8: strcpy(res,"acht"); break;
                case 9: strcpy(res,"neun"); break;
                default: strcpy(res,"Algarismo desconhecido.");
        }
        return res;
}
```

3. a declaração da função é escrita num ficheiro `extenso.h`:

```
#include <string.h>
char *alemao(char *res, int algarismo);
```

4. O programa principal, `principal.c`, invoca a função alemao():

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

int main() {
        int k; char palavra[100];
        printf("Introduza um algarismo: ");
        scanf("%d", &k);
        printf("%s\n", alemao(palavra, k));
}
```

Exercício 4 *Escreva os comandos seguintes:*

1. *Comandos para compilar os programas C (sem gerar o executável).*

2. *Comandos para gerar um executável* `alemao.exe`

2.4.1 Makefiles

Analise a seguinte Makefile:

```
#
# Makefile para o primeiro programa C
# PPII, 2003
#
CFLAGS =
```

```
CC = gcc

alemao.exe: principal.o extenso.o
	$(CC) -o alemao.exe principal.o extenso.o

principal.o: principal.c extenso.h
	$(CC) -c principal.c

extenso.o: extenso.c extenso.h
	$(CC) -c extenso.c

teste:
	echo "9" | ./alemao.exe

limpa:
	rm *.log *.dvi *.aux
	rm *~
	rm *.o
```

Exercício 5 *Considerando a Makefile apresentada, e que a mesma está guardada no arquivo Makefile, responda às seguintes questões:*

1. *Comando para gerar o executável.*
2. *Comando para testar o executável.*

Exercício 6 *Estude o funcionamento do comando make, e diga:*

1. *Como poderá invocar o make, de forma a redefinir a macro CC, de forma a que CC = cc, e não a gcc.*

   make "CC=cc"

2. *Qual a diferença entre os comandos:*

   ```
   limpa:                          limpa:
           rm *.log *.dvi *.aux            -rm *.log *.dvi *.aux
   ```

   Quando a execução de um comando retorna erro, o make pára. Com o prefixo '-', o make não pára se o comando retornar um erro.

3. *Substitua os comandos que determinam a compilação dos ficheiros principal.c e extenso.c por uma destas regras (equivalentes):*

   ```
   .c.o:                           .c.o:
           $(CC) $(CFLAGS) -c $<           $(CC) $(CFLAGS) -c $*.c
   ```

   Diga qual o significado destas regras.

3 LaTeX

Exercício 7 *Acrescente na Makefile os comandos para gerar um documento com o enunciado desta ficha teórico-prática:*

1. *Em Postscript,*
2. *Em PDF.*

Exercício 8 *Diga, sucintamente, o que fazem os seguintes packages:*

1. *listings*

   Para incluir todo o tipo de listagens de programas.

2. *srcltx*

   Para permitir o "inverse search", à custa de acrescentar mais informação no .dvi. Ou seja, é possível a partir do .dvi localizar o ficheiro fonte e a linha de texto associada.

Nota: Veja como configurar um visualizador de DVIs de forma a tirar partido do package srcltx. Por exemplo, em Windows, pode configurar o YAP para visualizar o ficheiro que deu origem ao DVI com o seguinte comando (para utilizadores do WinEdt):

```
"C:\Program Files\WinEdt\winedt.exe" "[Open('%f');SelLine(%l,8)]"
```

ou (para utilizadores do Vi):

```
"C:\Program Files\Vim\vim60\gvim.exe" "%f" +%l
```

Exercício 9 *Defina uma formatação para a linguagem bash, para mostrar, por exemplo, a seguinte script:*

```
#!/bin/bash
for numero in 0 1 2 3 4 5 6 7 8 9; do
        echo $numero | ./a.out
done
```

```
\lstdefinelanguage{bash}{%
keywords={for, if, else, then, fi, echo, case, in,
elif, esac, for, do, done, tar, while, until, shift,
mv, mkdir, ;;, \$\#, \$1, \$2},
morestring=[d]{"},
morecomment=[l]{\#},
sensitive=TRUE,
basicstyle={\small\ttfamily}}
```

Para utilizar, basta fazer:

```
\lstset{language={bash},breaklines}
\lstinputlisting{progs/testa}
```

4 Gnu RCS

O RCS é um sistema de controlo de versões, disponível gratuitamente. Serve para coordenar a criação e edição de ficheiros para um conjunto de indivíduos ou para apenas um utilizador.

Sumariamente, o RCS permite:

- o acesso à última versão de um ficheiro que está a ser trabalhado por várias pessoas
- o acesso às versões anteriores e respectiva recuperação caso necessário
- ver a lista de alterações que foram feitas pelos vários utilizadores
- evita que dois utilizadores façam alterações em paralelo ao programa, evitando que apareçam múltipla últimas versões.

A utilização de controlo de versões é uma prática muito aconselhada na engenharia de software e um requisito fundamental no desenvolvimento de grandes projectos de software.

O RCS consiste em quatro comandos mais frequentemente usados - ci, co, rlog e rcsdiff - e por outros muito pouco utilizados - rcs, ident, rcsclean e rcsmerge.

Exercício 10 *Estude o sistema RCS. Assuma que se criou uma pasta RCS, na pasta onde está o ficheiro alemao.c.*

Diga o que fazem os seguintes comandos:

1. *ci -u alemao.c*
2. *co RCS/*
3. *co -l alemao.c*
4. *ci -l alemao.c*
5. *rlog alemao.c*
6. *rscdiff -u alemao.c*

Exercício 11 *Imagine que eram acrescentadas as seguintes regras na Makefile apresentada.*

```
SRCS=principal.c extenso.c
$(SRCS):
        co $@
```

1. *Qual o propósito desta regra?*
2. *Escreva uma regra mais adequada ao caso apresentado, atendendo a que existem outros ficheiros que não são .c.*