Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

SISTEMAS OPERATIVOS I

Ficha 6

Abril de 2006

Nuno Malheiro
Maria João Viamonte
Berta Batista
Luis Lino Ferreira

Sugestões e participações de erros para:
ntm@dei.isep.ipp.pt

1 A compilação simples de um programa em C

1.1 Coloque o código fonte se encontra a seguir, num ficheiro chamado “ola.c”. Compile-o utilizando o gcc, com o comando que lhe segue. Execute o resultado usando a linha final.

```
#include <stdio.h>

int main (void)
{
  printf ("Ola Mundo!\n");
  return 0;
}
```

```
$ gcc -Wall ola.c -o ola
```

```
$ ./ola
```

1.2 Coloque o código fonte se encontra a seguir, num ficheiro chamado “mau.c”. Encontre os erros do programa seguinte, compilando-o com e sem os avisos. Corrija o programa de acordo com os avisos. Recompile-o e execute-o.

```
#include <stdio.h>

int main (void)
{
  printf ("Dois mais dois é %f\n", 4);
  return 0;
}
```

```
$ gcc mau.c -o mau
$ ./mau
```

```
$ gcc -Wall mau.c -o mau
```

1.3 Compilação do exemplo “Olá Mundo” usando múltiplos ficheiros fonte

Crie um ficheiro chamado “ola.h” com o código:

```
void ola (const char * nome);
```

Crie um ficheiro chamado “ola_fn.c” com o código:

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

void ola (const char * nome)
{
  printf ("Ola, %s!\n", nome);
}
```

Crie um ficheiro chamado “main.c” com o código:

```
#include "ola.h"

int main (void)
{
  ola ("Mundo");
  return 0;
}
```

Compile o código, usando a linha de comando:

```
$ gcc -Wall main.c ola_fn.c -o novo_ola
```

Verifique o funcionamento:

```
$ ./novo_ola
Ola Mundo!
```

1.4 Compilação independente do exemplo “Olá Mundo”

Crie o ficheiro objecto “main.o” usando a linha de comando:

```
$ gcc -Wall -c main.c
```

Crie o ficheiro objecto “ola_fn.o” usando a linha de comando:

```
$ gcc -Wall -c ola_fn.c
```

Crie um novo executável usando a linha de comando:

```
$ gcc main.o ola_fn.o -o ola_ind
```

Execute-o, verificando que o resultado é o mesmo:

```
$ ./ola_ind
```

1.5 Alteração de ficheiros e recompilação do exemplo “Olá Mundo”

Edite o ficheiro “main.c” , modificando a instrução de impressão para:

```
. . .
  ola ("a Todos");  /* alterado de "Mundo" */
. . .
```

Recompile o ficheiro “main.c” usando o commando:

```
$ gcc -Wall -c main.c
```

Religue o novo ficheiro objecto usando o comando:

```
$ gcc main.o ola_fn.o -o ola_relig
```

Verifique o resultado do executável usando a linha de comando:

```
$ ./ola_relig
```

2 Exemplos mais avançados de compilação de um programa em C

2.1 Um makefile simples para o exemplo “Olá Mundo”

Edite novamente o ficheiro “main.c” , modificando a instrução de impressão para:

```
. . .
  ola ("Mundo");
. . .
```

Crie um makefile para o exemplo “Olá mundo” com o conteúdo:

```
CC=gcc
CFLAGS=-Wall
FICHEIROS = main.o ola_fn.o
EXECUTAVEL = main
$(EXECUTAVEL): $(FICHEIROS)

clean:
      rm -f $(EXECUTAVEL) $(FICHEIROS)
```

Crie o executável usando o comando make:

```
$ make
```

Verifique que todos os ficheiros foram compilados e note que o comando make usa a opção –Wall por omissão. Execute o resultado usando a linha de comando:

```
$ ./main
```

Edite novamente o ficheiro “main.c”, modificando a instrução de impressão para:

```
. . .
  ola ("a Todos");
. . .
```

Crie o executável usando o comando make:

```
$ make
```

Verifique que só foi compilado o “main.c”, tendo sido efectuada uma religação no final. Execute o resultado usando a linha de comando:

```
$ ./main
```

2.2 Exemplo de ligação a biblioteca estática

Crie um ficheiro chamado “curs.c” com o código:

```
#include <curses.h>

int main(int argc, char *argv[])
{
   initscr(); clear();
   move(5,32); insstr("Cadeira de SO1");
   move(7,18); insstr("Biblioteca de tratamento de terminal - CURSES");
   refresh();  getchar(); endwin();
   return 0;
}
```

Tente compila-lo através do comando seguinte e identifique o erro:

```
$ gcc -Wall curs.c -o curs
```

Recompile o programa através da linha de comando seguinte e comente as diferenças:

```
$ gcc -Wall curs.c /usr/lib/libcurses.a -o curs
```

Execute o programa anterior usando a linha de comando:

```
$ ./curs
```

2.3 Exemplo de Ligação a bibliotecas partilhadas

Usando o exemplo de 2.2, crie um novo executável usando o comando:

```
$ gcc -Wall curs.c -lcurses -o curs_partilhado
```

Compare o ficheiro executável “curs” criado em 2.2 com o ficheiro executável “curs_partilhado” quer em termos de tamanho quer de execução.

2.4 Exemplo sobre directórios de pesquisa

Crie um ficheiro chamado “lista.c” com o código:

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

void  myfunc(char *i, char **udata)
{
   printf("%s ; %s : %s\n", udata[0], udata[1], i);
}

int main(void)
{
  GList *list = NULL;
  char *args[] = {"Parametro 1", "Parametro 2"};

  list = g_list_append(list, "Elemento 1");
  list = g_list_append(list, "Elemento 2");

  g_list_foreach(list, (GFunc) myfunc, args);
  return 0;
}
```

Tente compila-lo usando a linha de comando seguinte e verifique os erros:

```
$ gcc -Wall lista.c -lglib-2.0
```

Veja quais os directórios necessários aos ficheiros de inclusão através do comando:

```
$ pkg-config --cflags glib-2.0
```

Compile novamente programa mas com a linha de comando:

```
$ gcc -Wall lista.c $(pkg-config --cflags glib-2.0) -lglib-2.0
```

Execute o programa anterior usando a linha de comando:

```
$ ./a.out
```

Atribua a variável de ambiente C_INCLUDE_PATH o valor:

```
$ C_INCLUDE_PATH=/usr/include/glib-2.0:/usr/lib/glib-2.0/include
$ export C_INCLUDE_PATH
```

Volte a tentar compila-lo usando a linha de comando seguinte e discuta o funcionamento:

```
$ gcc -Wall lista.c -lglib-2.0
```

2.5 Utilização do Pré-processador

Crie um ficheiro chamado “dteste.c” com o conteúdo:

```
#include <stdio.h>

int main (void)
{
#ifdef TESTE
  printf ("Modo de teste\n");
#endif
  printf ("Execução...\n");
  return 0;
}
```

Compile-o e execute-o usando a linha de comando:

```
$ gcc -Wall -DTESTE dteste.c
$ ./a.out
```

Volte a compilar e executar usando a linha de comando:

```
$ gcc -Wall dteste.c
$ ./a.out
```

2.6 Macros com valores

Crie um ficheiro chamado “dtestevalor.c” com o conteúdo:

```
#include <stdio.h>

int main (void)
{
  printf ("Valor de NUM e %d\n", NUM);
  return 0;
}
```

Compile-o e execute-o usando a linha de comando:

```
$ gcc -Wall -DNUM=100 dtestevalor.c
$ ./a.out
```

Volte a compilar e executar usando a linha de comando:

```
$ gcc -Wall -DNUM="2+2" dtestevalor.c
$ ./a.out
```

Compile e execute novamente usando a linha de comando:

```
$ gcc -Wall -DNUM dtestevalor.c
$ ./a.out
```

Verifique os pré-processamentos anteriores usando:

```
$ gcc -Wall -DNUM=100 -E dtestevalor.c
```

```
$ gcc -Wall -DNUM="2+2" -E dtestevalor.c
```

```
$ gcc -Wall -DNUM -E dtestevalor.c
```

2.7 Macros com valores

Acrescente ao início do ficheiro de 2.6 a linha

```
#define NUM 4*3
```

Verifique o pré-processamento usando:

```
$ gcc -Wall -E dtestevalor.c
```

Compile e execute novamente o programa com a linha de comando:

```
$ gcc -Wall -E dtestevalor.c
$ ./a.out
```

2.8 Passos de compilação

Usando o programa de 2.6, efectue a geração de todos ficheiros correspondentes a todos os passos de compilação usando:

```
$ gcc -Wall -save-temps dtestevalor.c
```

Descreva todos os ficheiros criados e respectivas extensões.

3 Compilação para Optimização

Crie um ficheiro chamado “teste_opt.c” com o conteúdo:

```
#include <stdio.h>

double potencian (double d, unsigned n)
{
  double x = 1.0;
  unsigned j;

  for (j = 1; j <= n; j++)    x *= d;

  return x;
}
```

```
int main (void)
{
  double soma = 0.0;
  unsigned i;

  for (i = 1; i <= 100000000; i++)   soma += potencian (i, i % 5);

  printf ("soma = %g\n", soma);
  return 0;
}
```

Experimente a sua compilação e execução com as várias alternativas de compilação, verificando para cada uma qual o tempo de execução do programa e qual o ganho de optimização:

```
$ gcc -Wall -O0 teste_opt.c -lm
$ time ./a.out
```

```
$ gcc -Wall -O1 teste_opt.c -lm
$ time ./a.out
```

```
$ gcc -Wall -O2 teste_opt.c -lm
$ time ./a.out
```

```
$ gcc -Wall -O3 teste_opt.c -lm
$ time ./a.out
```