



Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Viana do Castelo

Arquitectura e Sistemas de Computadores

2020/2021

Trabalho Prático Laboratorial 1

Iniciação à linguagem C
Compilação com GCC
Análise do código Assembly

Emanuel Lomba
emanuellomba@estg.ipvc.pt

Outubro 2020
(adaptação da *Ficha Prática n° 1* de ASC-ERSC, sil@estg.ipvc.pt, 2019)

1. Objectivos

Escrever um programa simples em linguagem C

Compilar o programa com o compilador GCC

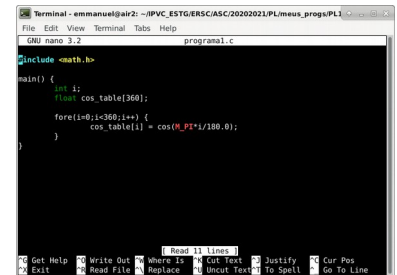
Identificar as diferentes etapas do processo de compilação do GCC

Gerar código *Assembly* com o GCC

2. Material necessário (além do computador)

2.1 Editor de Texto: Para a escrita e/ou consulta dos ficheiros de código fonte (.c ou .asm, entre outros), é necessário um Editor de Texto; e.g., Notepad++, Nano, Vi.

N.B. Não confundir editor de texto com Processador de Texto! O *Writer* da LibreOffice ou o *Word* da Microsoft são processadores de texto. Um processador de texto, além das funções básicas de um editor de texto (escrever, guardar, modificar), permite formatar o texto; e.g. fonte, tamanho desta, cor do texto, formatação das páginas e dos parágrafos, criação de índices, etc.



2.2 GCC: O *GNU Compiler Collection* é um conjunto de ferramentas e bibliotecas para compilar códigos de diversas linguagens, tais como Ada, Fortran, C++, ou C, como é o caso deste trabalho.

O GCC está normalmente disponível nos sistemas Unix/Linux. No entanto, é possível utilizá-lo em sistemas Windows, recorrendo à distribuição Cygwin (entre outras).



GCC: <https://gcc.gnu.org/>

Cygwin: <https://www.cygwin.com/>

No desenvolvimento deste guião, foram usadas as aplicações GNU Nano 3.2 e GCC 8.3.0, sobre Linux Debian 10 (buster).

3. Compilação de um programa com o GCC

O GCC compila um ficheiro (ou mais) de código fonte em linguagem C, em quatro passos, para gerar o ficheiro executável.

a) Pré-processamento (*Preprocessing*)

Neste primeiro passo, o GCC elimina comentários, substitui textos e inclui ficheiros externos; gerando um ficheiro com a extensão '.i' que normalmente não é guardada na pasta de trabalho. As

tarefas de substituição e inclusão) são levadas a cabo, pela interpretação de directivas: linhas começadas pelo carácter # (e.g., `#define` ou `#include`).

b) Compilação (*Compilation*)

Neste passo, o compilador traduz o produto do passo anterior em código Assembly, dando origem a ficheiro com a extensão '.s'.

c) Assemblagem (*Assembling*)

Nesta fase do processo, o GCC converte o código assembly em código de máquina, gerando um código Objecto com a extensão '.o'.

d) Linkagem (*Linking*) - Ligação

Nesta última fase do processo, o GCC estabelece as ligações entre os diversos ficheiros de código Objecto e gera o ficheiro executável. Note-se que os outros ficheiro Objecto podem ter sido criados pelo autor do programa ou, no mínimo, tratar-se-ão de ficheiros Objecto relacionados com inclusões declaradas e tratadas na fase de pré-processamento (e.g. a directiva `#include <stdio.h>` implicará que o ficheiro pré-compilado 'stdio.o' seja ligado ao projecto, nesta fase).

A Figura 1 ilustra os passos seguidos pelo GCC para gerar o ficheiro executável.

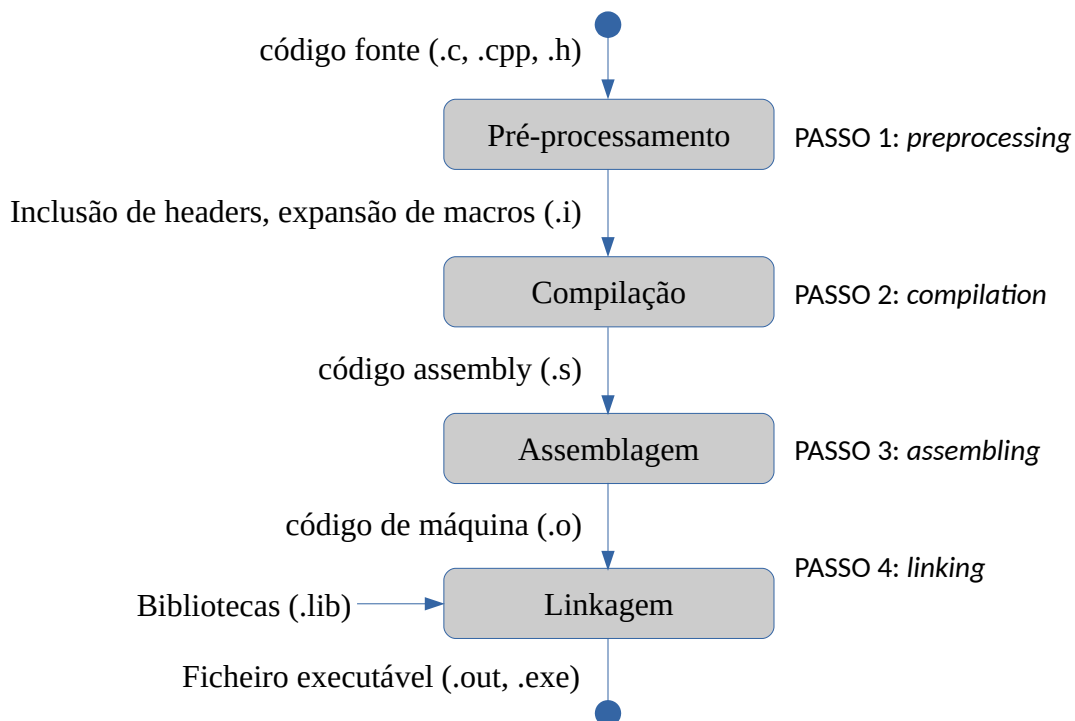


Figura 1 - Processo de “fabricao” do executável

4. Exemplo de compilação de um programa em C

Considere o programa abaixo, escrito em C. Copie este programa para um ficheiro nomeado `hello.c`, recorrendo a um editor de texto (e.g. `nano`).

```
//  
// Ficheiro: hello.c  
// ESTG-IPVC | ERSC-AESDC | SIL 2018  
//  
  
#include <stdio.h>  
#define PI 3.14  
  
int main() {  
    printf(" Hello World! O valor de PI é: %f\n", PI);  
    return 0;  
}
```

O compilador GCC permite efectuar separadamente alguns passos do processo de compilação, tais como: pré-processamento, compilação, “assemblagem” e “linkagem”. Se nenhuma opção for usada, são executados todos os passos, sendo gerado um ficheiro executável se não ocorrer nenhum erro ao longo de todo processo.

Existem diversas opções disponibilizadas pelo GCC, sendo as mais importantes as seguintes:

- ‘-E’, permite terminar o processo de compilação logo após o pré-processamento de diretivas e macros.
- ‘-S’, gera código assembly correspondente ao programa de origem em C, não gera código binário.
- ‘-c’, gera código binário mas não fazer a linkagem.
- ‘-o filename’, coloca o código binário no ficheiro “filename”.

Exemplo 1: Para compilar o programa `hello.c` escreva o seguinte comando no terminal:

```
$ gcc hello.c
```

O ficheiro executável de saída gerado pelo GCC é chamado de ‘a.out’ em sistemas Linux; em sistemas Windows, o ficheiro será nomeado ‘a.exe’. Para executar o programa:

```
$ ./a.out          ou          $ ./a.exe
```

Exemplo 2: Para especificar o nome do ficheiro de saída, utiliza-se a opção ‘-o’:

```
$ gcc hello.c -o hello.out
```

Para executar o programa:

```
$ ./hello.out
```

O processo de compilação com o GCC pode ser executado de forma manual pelo utilizador, através dos passos abaixo. Desta forma, é possível ter-se acesso aos ficheiros intermédios

Etapa 1) *Preprocessing*: gera ficheiro C (*.i) preprocessado:

```
$ gcc -lm hello.c -E -o hello_etapa1.i
```

Etapa 2) *Compilation*: gera código Assembly (*.s):

```
$ gcc hello.c -S -o hello_etapa2.s
```

Etapa 3) *Assemble*: gera código Objecto (*.o)

```
$ gcc -c hello_etapa2.s -o hello_etapa3.o
```

Etapa 4) *Linking*: invocar o linker do GCC para gerar o ficheiro binário executável (*.out, *.exe):

```
$ gcc hello_etapa3.o -o hello.out
```

5. Resolução de exercícios

Considere os seguintes programas escritos em C, na resolução dos exercícios.

```
// Programa #1

#include <math.h>
main(){
    int i;
    float cos_tabela[360];
    for(i=0;i<360;i++) {
        cos_tabela[i] = cos(M_PI*i/180.0);
    }
}
```

```
// Programa #2

#include <stdio.h>
#include <stdlib.h>
int main() {
    int c, n;
    printf("Dez numeros aleatorios no intervalo [1,100]\n");
    for(c = 1; c <= 10; c++) {
        n = rand() % 100 + 1;
        printf("%d\n", n);
    }
    return 0;
}
```

```
// Programa #3

#include <stdio.h>
#include <math.h>
#define MIN 0
#define MAX 360
void main(void){
    int i,j;
    char *s;
#ifdef DEBUG
    printf("A calcular a tabela de COS ...\n");
#endif
    for(i=MIN;i<MAX;i++) {
        printf("cos(%d*PI/180.0) = %f\n", i, cos(M_PI*i/180.0));
    }
#ifdef DEBUG
    printf("Terminado.\n");
#endif
}
```

5.1 Para cada programa, utilize o GCC para verificar quais são as alterações realizadas no passo de pré-processamento do código.

Exemplo (substitua 'xpto' pelo nome do seu ficheiro; e.g. programa1.c):

```
$ gcc -lm xpto.c -E -o xpto_step1.i
```

NOTA: Em algumas distribuições do GCC a inclusão da *flag* -lm é necessária quando se recorre à biblioteca 'math.h'.

Repita o procedimento efectuado, mas desta vez utilize o GCC com o argumento "-D DEBUG".

Exemplo (substitua 'xpto' pelo nome do seu ficheiro; e.g. programa1.c):

```
$ gcc xpto.c -E -D DEBUG -o xpto_step1.i
```

5.2 Para cada programa, gere o respectivo código Assembly. Caso obtenha erros ou avisos, identifique o ponto da compilação em que ocorreram e altere o programa e/ou as opções da linha de comando do GCC para que a compilação seja bem sucedida.

Exemplo (substitua 'xpto' pelo nome do seu ficheiro; e.g. programa1.c):

```
$ gcc xpto_step1.i -S -o xpto_step2.s
```

5.3 Compare o código Assembly gerado com o código fonte em C. O Assembly obtido é para que arquitectura? Tente identificar as estruturas principais do programa: ciclos, saltos, saltos condicionais e chamadas de funções.

5.4 Para o Programa #2, gere o código Assembly com as opções adicionais de optimização de código do GCC: “-O1” e “-O2”. Compare o código Assembly gerado e observe as diferenças. Compare também as versões do código optimizadas com e sem optimização.

Exemplo:

```
$ gcc programa2_step1.i -O1 -S -o programa2_step2_01.s
```

Leitura adicional: M. T. Jones, “Optimization in GCC”, Linux Journal, January 2005, [weblink](#).

5.5 Para cada programa, compile o código Assembly de forma a obter o respectivo código Objecto.

Exemplo (substitua ‘xpto’ pelo nome do seu ficheiro; e.g. programa1.c):

```
$ gcc -c xpto_step2.s -o xpto_step3.o
```

Use o comando nm para verificar quais os símbolos não definidos, i.e., cujo código não está incluído, no ficheiro objecto e provém de bibliotecas adicionais.

Exemplo (substitua ‘xpto’ pelo nome do seu ficheiro; e.g. programa1.c):

```
$ nm xpto_step3.o
```

Informação adicional: Linux / Unix Command: nm, [weblink](#).

5.6 Compile o código obtido na alínea anterior para obter um ficheiro binário executável. Caso obtenha erros ou avisos do *linker*, identifique os símbolos problemáticos e altere as opções da linha de comando do GCC para que o processo de compilação seja bem sucedido (sugestão: faça uso das opções, -L e/ou -l).

Exemplo (substitua ‘xpto’ pelo nome do seu ficheiro; e.g. programa1.c):

```
$ gcc xpto_step3.o -o xpto_step4.out
```

FIM.