Introdução aos Algoritmos e Estruturas de Dados (https://fenix.tecnico.ulisboa.pt/disciplinas/IAED7645111326/2016-2017/2-semestre)

Exercícios facultativos, folha 01

Com esta folha de exercícios, procuramos oferecer um breve exemplo sobre os passos intermédios do processo de compilação, assim como mostrar como se compila programas com código espalhado por vários ficheiros. No exercício 1 em baixo comece por fazer download do ficheiro sugerido. Nesse pacote encontrará vários ficheiros com versões alternativas para o cálculo do factorial de um número. Dependendo de como faz a ligação no acto de compilação poderá usar uma outra versão na criação do executável. No exercício 2 oferecemos um percurso sobre os passos intermédios do processo de compilação. Experimente!

1. **Compilação e ligação de múltiplos ficheiros (facultativo)**:
    1. Faça o download do ficheiro `fact.tgz (https://dspace.ist.utl.pt/bitstream/2295/1117671/1/fact.tgz)` para o seu directório de trabalho.
    2. Descomprima o ficheiro com o comando:
       `$ tar xvfz fact.tgz`
       Foi criado um directório denominado `fact` com o conteúdo do arquivo. Entre dentro desse directório .
    3. Visualize os ficheiros `main.c`, `fact.h` , `iter.c` e `recurs.c` com um editor de texto, e observe:
        - declaração da função main: argumentos e tipo de retorno
        - valores de retorno
        - função do programa
        - includes e variáveis globais
    4. Compile os módulos fonte do programa iterativo (`main.c` e `iter.c`) e faça a ligação do código objecto.
       `$ gcc -ansi -pedantic -Wall -o iter main.c iter.c`
    5. Execute o programa `iter` e verifique que este imprime o factorial de 5.
    6. Repita os passos 4 e 5 utilizando a versão recursiva.

2. **Análise de passos intermédios do processo de compilação (facultativo)**:

   É naturalmente interessante perceber o que o seu compilador está a fazer. Para tal sugerimos que tente analisar os passos intermédios do processo de compilação.

    1. *Pré-processamento*: fase que antecede a compilação e que executa as directivas iniciadas por #. Por exemplo, no ficheiro `main.c` serão processadas as directivas `include` e `define` .
       `$ gcc -E main.c`
       O resultado do pré-processamento é enviado para o terminal. (O pré processador pode ser invocado separadamente com o comando `cpp`)
    2. *Compilação*: fase que gera código final em assembly. O assembly ainda tem um formato textual, pode ser lido e modificado por um vulgar editor de texto, mas o código gerado já depende do processador, arquitectura e sistema operativo.
       `$ gcc -S iter.c`
       Verifique o código assembly gerado no ficheiro `iter.s` . Compare as variantes do ficheiro `iter.s` quando utiliza o optimizador (adicionar a opção `-O`) e a informação para o debugger (adicionar a opção `-g`).
    3. *Montagem ou assemblagem*: fase que produz os códigos binários, ficheiros objecto (".o", nada tem a ver com linguagens orientada para objectos), que serão processados pelo CPU. Esta fase é independente da linguagem de alto nível utilizada: C, Pascal, Fortran, etc.
       `$ gcc -c main.c`
       Verifique o tipo de ficheiro gerado, com o comando:
       `$ file main.o`
       e as dimensões das secções, com o comando:
       `$ size main.o`
       e a tabela de símbolos, com o comando:
       `$ nm main.o`
    4. *Ligação ou Linkagem*: fase que produz o ficheiro executável final através da interligação dos vários ficheiros objectos ou de bibliotecas (conjuntos de ficheiros objecto).
       `$ gcc -o factorial main.o iter.s`
       Verifique o tipo de ficheiro gerado e as dimensões das secções. Use o comando `ldd` para verificar as dependências das bibliotecas dinâmicas. Tente gerar um ficheiro executável com as variantes iterativa e recursiva, simultaneamente,
       `$ gcc main.c iter.c recurs.c`
       e verifique que existem duas realizações de factorial com o mesmo nome. Por outro lado, se tentar criar um ficheiro executável apenas com o ficheiro `main.c` ,
       `$ gcc main.c`
       falta um ficheiro ou biblioteca que forneça uma realização de factorial.
    5. O processo completo (*executável estático vs dinâmico*): o comando `gcc` permite, como já pode observar, controlar todo o processo de compilação para a linguagem C. Contudo, pode verificar a execução das diversas fases através da opção `-v` .
       `$ gcc -v -static main.c iter.c`
       Neste exemplo, geramos um executável estático, isto é, não depende na execução da existência das bibliotecas dinâmicas. Como contrapartida, o executável final fica muito maior, pois inclui no próprio ficheiro uma cópia de todas as funções utilizadas, como por exemplo o `printf` . Verifique o tipo, dimensões do ficheiro (`ls -l`), dimensões das secções e dependências do ficheiro `a.out` gerado, face ao executável dinâmico gerado na alínea anterior. Retire a informação simbólica, com o comando `strip` , e verifique que o executável ficou mais pequeno e que já não é possível saber a posição dos símbolos (comando `nm`).
    6. Faça o download do ficheiro fact-makefile.zip (https://dspace.ist.utl.pt/bitstream/2295/1117672/1/fact-makefile.zip). Inspeccione o exemplo de ficheiro "*Makefile*" com um editor de texto e crie os dois executáveis através do comando *make*.