DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Licenciatura em Engenharia Eletrotécnica e de Computadores (LEEC)

Algoritmos e Estruturas de Dados
Aula de Laboratório #01

Objetivos

Neste laboratório pretende-se ilustrar um conjunto de ferramentas cuja utilização será útil no desenvolvimento de código. A abordagem seguida é de exemplificação das capacidades e modo de utilização de cada uma destas ferramentas, suportada para isso num conjunto de exemplos simples. O laboratório complementa a aula prática abordando os mesmos tópicos, nomeadamente o desenvolvimento, compilação, depuração e verificação de código.

O funcionamento da disciplina é totalmente agnóstico ao ambiente de desenvolvimento utilizado pelos alunos embora no laboratório se disponibilizem apenas máquinas com o sistema operativo Linux.

Será ainda apresentado o sistema de submissão e avaliação que será usado na disciplina.

Plano da Aula

Para atingir os objetivos descritos, preconiza-se o seguinte plano de aula:

1 Passos da Compilação

Considere o programa seguinte que se encontra disponível no ficheiro `lab1a.c`.

1.1. Compile este programa com `gcc` usando o comando `gcc -g -c lab1a.c`. Verifique que como resultado da compilação foi criado um novo ficheiro, `lab1a.o`. Veja de que tipo é o ficheiro.
Nota: para ver propriedades de um ficheiro pode, na linha de comandos, usar o comando `file`: `file lab1a.o`.

1.2. Compile de novo o programa com `gcc`, agora usando o comando `gcc -g -o lab1a lab1a.o`. Verifique que, como resultado da compilação, foi criado um novo ficheiro. Veja de que tipo é o ficheiro e compare com o ficheiro criado na alínea anterior.

1.3. Analise os resultados anteriores de forma a identificar os vários passos do processo de compilação e a sua relevância.

1.4. Compile o ficheiro com os seguintes argumentos: `gcc --std=c90 -g -o lab1a lab1a.c`. Verifique que o compilador indica diversos erros e que não foi criado nenhum executável. Identifique esses erros por inspeção do código fonte.

1.5. Sem efetuar nenhuma alteração no código fonte, compile de novo o ficheiro, agora com os seguintes argumentos: `gcc -g -o lab1a lab1a.c`. Verifique que o compilador já não apresenta qualquer erro e que foi criado um executável.

2 Depuração

Considere ainda o programa disponível no ficheiro `lab1a.c`. O programa é suposto receber uma *string* como argumento e calcular a partir dela o número de letras, dígitos e outros caracteres incluídos na mesma *string*.

2.1. Execute o programa com os seguintes argumentos: `./lab1a -/aed-MASTER+92+` e analise o resultado apresentado. Atendendo ao funcionamento descrito para o programa justifique o resultado obtido.

2.2. Identifique eventuais problemas no código disponibilizado que lhe permitam corrigir o problema para obter o resultado esperado com o argumento utilizado. Corrija esses problemas.

2.3. Utilize `valgrind` para correr o programa, executando o comando `valgrind --leak-check=full ./lab1b -/aed-MASTER+92+`. Verifique o resultado.

2.4. Identifique e corrija quaisquer problemas adicionais que o programa tenha de forma a conseguir que o mesmo efetue exatamente o pretendido.

Considere agora o programa disponível no ficheiro `lab1b.c`. O programa é suposto receber um curto texto ou *string* como argumento e calcular a partir dele a frequência com que cada letra do abecedário aparece nesse texto (o programa ignora a diferença entre maiúsculas e minúsculas – veja no código como o faz).

2.5. Execute o programa com os seguintes argumentos: `./lab1b bananinhas` e analise o resultado apresentado.

2.6. Identifique eventuais problemas no código disponibilizado.

2.7. Execute agora o programa com os seguintes argumentos: `./lab1b zaragatoa` e verifique o resultado. Identifique e corrija os problemas.

2.8. Utilize `valgrind` para correr o programa, executando o comando `valgrind --leak-check=full ./lab1b bananinhas`.

2.9. Corrija quaisquer problemas adicionais que o programa tenha de forma a conseguir que o mesmo efetue exatamente o pretendido.

Considere agora o programa disponível no ficheiro `lab1c.c`. Compile o programa com o comando `gcc -g -o lab1b lab1c.c`. Vamos voltar a utilizar o comando `valgrind` para nos auxiliar a detetar problemas relacionados com incorreta utilização de memória.

2.10. Utilize `valgrind` para correr o programa, executando o comando `valgrind --leak-check=full ./lab1c bananinhas`. O programa executa sem queixas ou erros, produzindo o resultado certo. Será que está mesmo tudo certo?

2.11. Utilize de novo o `valgrind` para correr o programa, executando agora o comando `valgrind --leak-check=full ./lab1c zaragatoa`. Identifique e corrija os problemas e justifique porque é que na execução anterior o valgrind não deu qualquer tipo de aviso.

2.12. Justifique por que é que na execução em 2.7 o programa *crashou*, mas agora não o fez sendo o problema semelhante.
Nota: na sua máquina não é impossível que a implementação que tem de valgrind possa não crashar.

Considere agora o programa disponível no ficheiro `lab1d.c`. Compile o programa com o comando `gcc -g -o lab1d lab1d.c`. Este programa recebe como argumentos uma sequência de números inteiros, que armazena numa lista simplesmente ligada e percorre posteriormente a lista para retirar da mesma lista todos os números ímpares que lá constem, mantendo os restantes ligados na lista.

2.13. Execute o programa com a seguinte sequência de números, `./lab1d 6 8 11 23 4 5 9 13 1 4`. Verifique que o resultado obtido é o pretendido.

2.14. Utilize agora `valgrind` para correr o programa com os mesmos argumento. Execute então o comando `valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes ./lab1d 6 8 11 23 4 5 9 13 1 4`. Verifique que o resultado é o mesmo, mas que o valgrind se queixa de memória não libertada. Será que ao apagar da lista os números ímpares, o programa liberta a respetiva memória? Analise a função `remove_next()` e justifique. Como poderia corrigir o problema?

2.15. Corrija o problema e volte a correr o programa com os mesmos argumentos para verificar que toda a memória é libertada.

2.16. Corra agora o programa com os seguintes argumentos `./lab1d 1 6 8 11 23 4 5`. Verá que o programa agora `crasha`.

2.17. Para procurar resolver o problema execute o programa no *debugger*, fazendo `gdb lab1d` e depois já na linha de comando do `gdb` faça `run 1 6 8 11 23 4 5`. Execute o comando `where` e verá que o programa está na função `remove_next` que foi chamada a partir da função `remove_odd_numbers` que por sua vez foi chamada da `main`. O *stack trace* do gdb não apenas dá esta informação como diz em que linhas foram feitas as chamadas respetivas às funções. Se inspecionar o valor da variável `previous_node`, que é o argumento de entrada da função, verá que ele tem o valor 0x0, ou seja NULL. Qualquer tentativa de dereferenciar um ponteiro de valor NULL dará erro (ou seja fazer `*previous_node` ou `previous_node->next` darão sempre erro). Porquê?

2.18. O que é estranho neste caso é a função `remove_node`, cuja função é remover um elemento da lista, ter sido chamado com um argumento NULL. Para perceber melhor isso execute na linha do `gdb` o comando `up` analise o código dessa função (executando por exemplo um ou mais comandos `list`). Analisando o código desta função, `remove_odd_numbers()`, pense no que sucede se o primeiro elemento da lista for um número ímpar, que pretendemos remover. Veja o que sucede nesse caso por inspeção do código.
Nota: poderá ser interessante ver os valores de `head`, `head->next`, `previous`, etc.

2.19. Faça as alterações necessárias no código para corrigir o problema e verifique que o mesmo deixou de existir e o programa tem o funcionamento esperado com qualquer tipo de argumentos (tente duas ou três situações, inclusivamente com todos os números pares ou todos ímpares).

3 Sistema de Submissão

Nesta parte da aula far-se-á uma introdução ao Mooshak, um excelente sistema de avaliação automática de programas, desenvolvido na Faculdade de Ciências da Universidade do Porto. O Mooshak foi inicialmente desenvolvido a pensar em competições de programação (como as Olimpíadas da Informática), mas atualmente também é usado para apoio a aulas em muitas universidades do nosso país.

Nesta disciplina será usado parcialmente para avaliação dos projetos dos alunos nas duas fases de submissão. Mais detalhes sobre estes concursos serão dados posteriormente, aqui faremos apenas uma introdução ao sistema, à sua utilização e à interpretação da informação por ele fornecida.

Para usar o sistema você precisa de uma password que lhe será atribuída. e que será única para cada grupo de projeto. As instruções para obter a password serão indicadas posteriormente mas ela ser-lhe-á entregue por email, no endereço que consta do Fénix. É por isso extremamente importante que o endereço de email que cada aluno tem indicado no Fénix seja atual e seja regularmente verificado. Se perder a password poderá obter uma nova da mesma forma. Não perca a password nem a revele a ninguém pois isso seria equivalente a divulgar publicamente a sua solução do trabalho final, o que seria considerado fraude.

A estrutura de funcionamento do Mooshak tem por base o conceito de concurso, no qual existem problemas. Para cada concurso e cada problema, cada candidato/grupo submeterá soluções que procurem responder aos requisitos desse problema. Para esta aula, que se pretende apenas de ilustração, utilizaremos o concurso do ano passado e será o docente que submeterá várias soluções que serão analisadas.

Esta parte da aula será fundamentalmente expositória. O docente executará os passos indicados de seguida, mostrará os resultados e analisará as diversas situações.

3.1. No seu *browser* vá à página de submissão do concurso pretendido. Para ilustração utilizaremos `http://amarguinha.tecnico.ulisboa.pt/~aed`. Escolha o concurso apropriado e entre com o seu username e password.
Nota: Nesta disciplina utilizaremos sempre este servidor, mas serão dadas instruções específicas para cada concurso/utilização.

3.2. Ao entrar no sistema poderá escolher o problema para o qual quer submeter, *clickando* no botão correspondente, poderá ver instruções básicas, no botão *View*, poderá colocar questões aos docentes (organizadores do concurso), no botão *Ask*, poderá ter ajuda no *Help*, e no final poderá sair com *Logout*. Poderá ainda escolher ver as submissões feitas, o *ranking* das submissões, e as *Questions*.

Ou poderá, como será feito, submeter uma solução. O formato da solução será definido especificamente para cada concurso. No caso deste que estamos a usar, a solução consiste num ficheiro *ZIP* (ou rar ou tar comprimido) contendo o código fonte, que depois será desempacotado, compilado e o seu executável será utilizado para correr os testes propostos, resultando num dado score.

3.3. Serão feitas diversas submissões, de códigos diferentes, de forma a ilustrar o interface do Mooshak, discutir algumas das mensagens providenciadas pelo sistema, mencionar alguns erros mais básicos mas mesmo assim frequentes, e genericamente apresentar aos alunos o sistema.

3.4. Será feita a análise de alguns elementos tanto do ponto de vista de quem submete como do avaliador, o *judge* do concurso.