Universidade de Brasília – UnB
Instituto de Ciências Exatas – IE
Departamento de Ciência da Computação – CIC
Algoritmos e Programação de Computadores (113476) - Profa. Carla Denise Castanho

– Laboratório 0 –

Introdução ao Ambiente de Programação

1. Entrando no ambiente

Nas atividades a seguir, vamos aprender os conceitos básicos do sistema operacional Linux, e como criar, compilar e executar nosso primeiro programa C.

1.1 Após ligar o computador e aguardar os procedimentos de inicialização, uma tela de login surgirá, onde você deve informar seu usuário e senha, obtenha esta informação junto ao pessoal do LINF. Dê **ENTER**.

1.2 Uma vez logado na máquina, abra um *terminal*, por meio de uma das duas opções:

1.2.1 Acesse: **Aplicativos > Sistema > Konsole**

1.2.2 ou pressione as teclas **ALT+F2** e digite `konsole`, pressionando **ENTER** em seguida.

1.3 Um terminal é uma aplicação que nos permite dar comandos ao sistema operacional. Por exemplo, para trocar sua senha de usuário Linux, digite no terminal o seguinte comando:

```
> passwd
```

o sistema vai solicitar que você entre com uma nova senha e, em seguida confirme.

2. Trabalhando com diretórios e arquivos

O sistema operacional salva todos os seus dados em **arquivos**, que são organizados em **diretórios**, sempre formando uma hierarquia. Quando você utiliza um terminal, há sempre um **diretório corrente**, que é o diretório base onde será executado seu comando. Quando você loga no sistema, o diretório corrente é sempre o **home** do usuário: cada usuário possui um diretório `/home/<nome de login>`, a partir do qual ele pode construir sua árvore de diretórios pessoal e armazenar os seus arquivos.

2.1 A qualquer instante, de qualquer outro ponto ou diretório do sistema, você pode voltar ao seu diretório home digitando o comando:

```
> cd
```

sem argumentos, ou ainda

```
> cd ~
```

pois o símbolo ~ significa seu diretório *home*.

2.2 Para verificar os arquivos existentes em um diretório, utilize o comando:

```
> ls
```

digite o comando ls no seu diretório *home* e verifique o que ele retorna.

2.3 O comando `cd` (*change directory*) altera o diretório corrente. No Linux, o **diretório raiz** é `/` e todos os caminhos são separados por `/`. Digite no terminal (sempre dando **ENTER** ao final de cada comando):

```
> cd /
> ls
```

o sistema lista todos os subdiretórios do diretório raiz:

- `/bin` - utilitários principais;
- `/dev` - dispositivos, tais como: partições de disco, unidade de cdrom, fax-modem, etc;
- `/etc` - arquivos de configuração da máquina;
- `/lib` - bibliotecas e funções utilizadas por outros aplicativos;
- `/tmp` - arquivos temporários;
- `/home` - onde ficam localizados os diretórios pessoais dos usuários cadastrados;
- `/usr/bin` - outros utilitários;
- `/usr/lib` - bibliotecas de função;
- `/usr/spool` - spool de e-mail e impressora.

Um **path**, ou **caminho**, é um descritor da localização de um objeto (arquivo ou diretório), por exemplo, `/home/carla/APC/trabalho1.c`. Note que letras maiúsculas e minúsculas fazem diferença: "Apc", "apc" e "APC" são nomes de pastas diferentes!

Quando descrevemos um caminho iniciando com `/`, ou seja, a partir do diretório raiz, este é um **caminho absoluto**, caso contrário, o caminho será tomado a partir do diretório corrente, nesse caso, é um **caminho relativo**. Para se referir ao diretório corrente, pode-se utilizar '`.`', quando queremos nos referir ao **diretório pai**, utilizamos '`..`'.

2.4 Abra um terminal e digite:

```
> cd /usr/lib
```

Observe que utilizamos um caminho absoluto. Agora digite:

```
> cd ..
> ls
```

Note que subimos para o diretório pai, *i.e*, `/usr`. Agora digite:

```
> cd ./lib
> ls
```

Voltamos a `/usr/lib`, nesse caso, poderíamos ter feito simplesmente:

```
> cd lib
```

pois, quando um caminho não se inicia com `/`, o sistema sempre assume que é um caminho relativo.

2.5 Se você quiser descobrir o diretório corrente, utilize o comando `pwd`, sem parâmetros, que mostra na tela seu caminho absoluto:

```
> pwd
```

2.6 Você pode executar:

```
> ls -l
```

em qualquer diretório para listar seus arquivos e subdiretórios com detalhes, tais como tamanho, usuário criador/dono, permissões de acesso, etc...

2.7 Digite:

```
> cd
```

para retornar ao seu diretório *home*.

2.8 Para criar um novo diretório, utilize o comando `mkdir`. Digite:

```
> mkdir trabalhos
> cd trabalhos
```

para criar um novo diretório chamado "trabalhos" e entrar nele.

2.9 Para remover um diretório (vazio), utilize o comando `rmdir`. Ainda dentro de "trabalhos", faça:

```
> mkdir xyz
> rmdir xyz
```

o primeiro comando cria um novo diretório vazio "xyz" e o segundo o apaga, logo em seguida.

2.10 Observe que só é possível remover diretórios **vazios**, caso queria remover um diretório e todo seu conteúdo (subdiretórios e arquivos), faça:

```
> rm --recursive --force <nome do diretório>
```

ou então:

```
> rm -rf <nome do diretório>
```

que é uma versão abreviada do primeiro comando.

Atenção! Tome extremo cuidado com esse comando, **o sistema operacional Linux sempre assume que você sabe o que está fazendo**, ele não vai pedir confirmação, vai executar todos os comandos como você digitar.

2.11 O comando `cp` é utilizado para copiar arquivos ou diretórios. Para copiar um arquivo, use:

```
> cp <nome do arquivo original> <nome destino da cópia>
```

já para copiar diretórios, use:

```
> cp -rf <nome do diretório original> <nome destino da cópia>
```

No seu diretório *home*, faça:

```
> cp -rf trabalhos trabalhos-copia
> ls
```

Observe que uma nova pasta "trabalhos-copia" foi criada, uma cópia de "trabalhos". Apague esta pasta:

```
> rmdir trabalhos-copia
```

2.12 Para mover (ou simplesmente renomear) um arquivo ou diretório, utilize o comando `mv`, da seguinte forma:

```
> mv <nome atual> <novo nome e/ou novo caminho>
```

Por exemplo, no seu diretório *home*, faça:

```
> mv trabalhos roteiros
> ls
```

Observe que a pasta “trabalhos” foi renomeada para “roteiros”.

Retorne-a para seu nome original:

```
> mv roteiros trabalhos
```

2.13 O comando `rm` é utilizado também para remover arquivos. Faça:

```
> rm <nome do arquivo>
```

para remover arquivos, ou:

```
> rm -rf <nome do diretório>
```

para remover diretórios. **Novamente: muito cuidado com este comando!!**

3. Alguns comandos úteis de terminal

3.1 Para limpar a tela de seu terminal, faça:

```
> clear
```

Esse comando é muito útil para deixar a tela do terminal mais legível após um número grande de operações.

3.2 Às vezes, é útil digitar dois comandos ao mesmo tempo no terminal. Um exemplo interessante seria mostrar o conteúdo de uma pasta logo após de limpar a tela com `clear`. Você pode tentar:

```
> clear
> ls
```

Ou, mais simples:

```
> clear && ls
```

o `&&` instrui o terminal a executar o comando `ls` assim que terminar de executar o comando `clear`.

3.3 Para sair e fechar um terminal, utilize o comando `exit`:

```
> exit
```

4. Criando, compilando e executando programas em Linguagem C

Os passos principais para criar e testar um programa em Linguagem C são os seguintes:

1. Abrir um editor de texto.
2. Escrever o programa.
3. Salvar o arquivo em um pasta conhecida, com a extensão ".c".
4. Abrir um terminal.
5. No terminal, mudar para o diretório em que está o arquivo com o código fonte.
6. Compilar o código fonte.
   a. Resolver eventuais erros de compilação.
7. Executar o programa.
   a. Resolver eventuais erros de execução.

Vamos ver cada um destes passos a seguir:

4.1 Abra o editor de texto padrão do KDE (interface gráfica do Linux), o Kate: pressione as teclas **ALT+F2** e digite `kate`, pressionando **ENTER** em seguida.

4.2 Digite o programa C a seguir:

```
#include <stdio.h>


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

4.3 Salve seu arquivo na pasta `/home/<seu usuário>/trabalhos` com o nome `ola.c`: pressione as teclas **CTRL+S** e, quando abrir um diálogo solicitando o local onde o arquivo deve ser salvo, navegue até a pasta "trabalhos" e coloque o nome "ola.c".

4.4 Abra um terminal e vá até o diretório "trabalhos". Utilize o comando `ls` para visualizar o conteúdo do diretório e garantir que seu arquivo está no local correto.

4.5 Digite o seguinte comando no terminal:

```
> gcc -ansi -Wall -o ola ola.c
```

Vamos entender o que este comando faz:

- **`gcc`** é o comando que invoca o compilador C, neste caso, estamos invocando a *GNU Compiler Collection*, que é uma suite de compiladores livre, disponível em distribuições Linux;
- **`-ansi`** é uma opção passada ao GCC para que só aceite código C em conformidade com o padrão ANSI C (este é o padrão que utilizaremos nesta disciplina, e seu uso aumenta a portabilidade do código – é possível compilar o mesmo código em várias plataformas diferentes);

- **`-Wall`** é outra opção passada ao GCC, habilita todas as *warnings*, isto é, construções que não impedem a compilação do programa, mas tipicamente estão erradas;
- **`-o`** `<nome do programa>` é a opção que indica ao GCC qual o nome do nosso arquivo de saída, isto é, nosso programa executável. No nosso caso, utilizamos a opção `-o ola` porque queremos que seja gerado um arquivo executavel chamado "ola". Caso você não especifique esta opção, o GCC criará por padrão um programa chamado `a.out`;
- **`ola.c`** – o último parâmetro que você deve passar ao GCC é o nome do seu arquivo de código fonte, observe que podemos indicar apenas o nome do arquivo, já que estamos no mesmo diretório em que ele se encontra.

4.6 Após invocar o GCC, desde que não haja nenhum erro na compilação do programa, deve existir um novo arquivo no diretório "trabalhos", chamado "ola" (sem nenhuma extensão). Confira se existe mesmo:

```
> ls
```

4.7 Agora execute seu programa:

```
> ./ola
```

Nesse caso, utilizar `./` é obrigatório, porque seu programa está no diretório local, não nos locais padrão de busca de comandos do Linux, lembre-se deste detalhe!

Quando você começar a escrever programas mais complexos, vão começar a surgir erros de compilação e *warnings*. Lembre-se de resolver todos os problemas com seu código, mesmo que seja uma *warning*: você perderá pontos em seus trabalhos se não o fizer!

E lembre-se também: **se um programa compilou sem erros, isso não necessariamente significa que está correto, ele pode ter erros de lógica!**

A seguir, alguns erros comuns de compilação que você vai encontrar no começo e suas possíveis causas:

- `warning: implicit declaration of function 'For'` – a linguagem C é case sensitive (diferencia maiúsculas e minúsculas), e o comando `for` só existe em minúsculas; isso também vale para outros comandos, como `while`, `case`, `switch`, `if`, etc...
- `error: 'i' undeclared (first use in this function)` – faltou declarar a variável "i" no escopo daquela função em que é utilizada.
- `error: expected ';' before '}' token` – observar a falta de ponto-e-vírgula no lugar indicado.
- `error: expected declaration or statement at end of input` – provavelmente erro por falta de chaves (note que a falta de chaves fechadas corretamente pode gerar diversas mensagens de erro diferentes).

Sugestão: configure seu editor de textos para enumerar as linhas, pois o compilador diz a linha que contém o erro.

Quando você for executar seu programa, podem surgir novos erros, agora de *runtime* (em tempo de execução), por exemplo:

- `floating point exception` – normalmente, divisão por zero;

- `segmentation fault` – você acessou uma área de memória indevidamente, provalmente uso incorreto de ponteiros (pode ser a falta do ”&” em um `scanf`) ou acesso a valores fora dos limites de um vetor;

ATENÇÃO: Ao corrigir seu código fonte, lembre-se sempre de salvar o arquivo e recompilá-lo antes de executar novamente.

Observação importante: ao incluir a biblioteca `<math.h>` no seu programa, para dispor de funções como `sqrt()` e `pow()`, por exemplo, é necessário usar a opção **`-lm`** na linha de comando do GCC, para que ele inclua a biblioteca de matemática ao gerar seu programa:

```
> gcc -ansi -Wall -lm -o <nome programa> <arquivo de entrada>
```