Representação de Números

Guião: **G-1**

António Pina

Apresentação

No âmbito desta disciplina **guião** é o nome dado a uma classe de documentos que, ao longo do semestre, irão sendo apresentados aos estudantes para satisfazer os seguintes objectivos: a) apresentar o enunciado dos trabalhos propostos para as sessões laboratoriais; b) fornecer informação útil para a compreensão e desenvolvimento dos trabalhos e c) guiar o estudante na exploração e desenvolvimento das actividades previstas.

Logística

Após a leitura do guião, realizada antes da sessão laboratorial, os estudantes poderão trabalhar individualmente ou em grupo de acordo com os recursos de computação disponíveis.

As salas de aulas possuem postos de trabalho que mediante a identificação através de um *nome*, e *palavra-passe* (fornecidos em tempo útil) disponibilizam aos estudantes um ambiente de desenvolvimento com acesso à Internet assente no sistema de exploração **Linux**.

Contexto
Este documento é parte indissociável da sessão laboratorial prevista para a terceira aula teórica-prática da semana 3 (31-03-2009) .

Objectivos
Os problemas apresentados destinam-se a ser resolvidos durante a sessão laboratorial, com vista a servir de complemento no processo de auto-avaliação, associado à resolução dos exercícios propostos nos **TPC1** e **TPC2**.

Temas
Neste guião são abordados os temas: Formatos de Inteiros e Representação de Objectos em Memória.
Aqueles que possuem menor experiência no uso do ambiente Linux poderão, também, beneficiar do texto introdutório ao Ambiente de Trabalho

Material de Apoio
As fontes dos programas desenvolvidos para esta sessão estão disponíveis em: ligação

---

Formatos de Inteiros

Os computadores contemporâneos armazenam e processam informação representada como sinais com dois valores. Estes dígitos elementares binários, ou *bits*, estão na base da revolução digital.

Enquanto isto os humanos continuam a privilegiar o sistema decimal, baseado na utilização de dez dígitos, para tratar e processar informação numérica.

Um bit é uma unidade de representação demasiado elementar para ser processada de forma eficiente pelos humanos. Assim o agrupamento de bits sob a forma de sistemas de base-8 ou de base-16 tem vindo a ser preferencialmente usado, sempre que se torna necessário interpretar e processar a informação produzida pelos computadores.

Neste contexto, uma tarefa comum é a conversão manual entre os vários sistemas de representação de números. Sempre que a magnitude dos valores é elevada é, naturalmente, mais prático usar uma calculadora ou computador para efectuar os cálculos.

No que segue vamos usar a linguagem de programação para **Perl** para fazer uma série de operações de conversão entre diferentes bases usadas para representar inteiros.

1. Decimal e Hexadecimal

Se pretendermos converter uma lista de números decimal, com sinal, para hexadecimal podemos usar o seguinte programa escrito em linguagem Perl:

```
#!/usr/bin/perl
# Converte uma lista de números em formato decimal para formato hexadecimal

for ($i = 0; $i < @ARGV; $i++) {
printf("%d\t= 0x%x\n", $ARGV[$i], $ARGV[$i]);
}
```

Use o seu editor preferido para escrever aquele programa e criar um ficheiro, em texto simples, para o salvaguardar com o nome ***d2h***. Seguidamente defina o atributo de execução para aquele ficheiro.

Um programa semelhante com o nome ***h2d*** que converte números em hexadecimal para decimal com sinal é apresentado a seguir:

```
#!/usr/local/bin/perl
# Converte uma lista de números em formato hexadecimal para formato em decimal

for ($i = 0; $i < @ARGV; $i++) {
$val = hex($ARGV[$i]);
printf("0x%x = %d\n", $val, $val);
}
```

Problema 1:

a) Converta **de** decimal/hexadecimal **para** hexadecimal/decimal a seguinte lista de números:

```
100          = 0x
500          = 0x
-500         = 0x____
751          = 0x
0xfffffe0c   =
1023         = 0x____
0x0ffffe0c   =
-128         = 0x
0xffffff80   =
```

b) Para confirmar as conversões execute na consola do Linux o programa ***d2H*** com os parâmetros abaixo indicados:

```
> ./d2H 100 500 751 1023
```

2. Binária e Octal

O programa **cvBases** cujo código, um pouco mais elaborado, a seguir se apresenta, permite visualizar, nos formatos binário, decimal, hexadecimal e octal, números introduzidos na linha de comando em notação decimal ou em hexadecimal `(0x---)`.

```
#!/usr/bin/perl
#!/usr/local/bin/perl
# Converte uma lista à entrada de números em decimal em hexadecimal para
# os formatos binário, decimal, hexadecimal e octal correspondentes.

for ($i = 0; $i < @ARGV; $i++) {
    if ($ARGV[$i]=~ /^0x/) {$valor=hex($ARGV[$i])} else {$valor = $ARGV[$i];}

printf("Binario: %16b\t Decimal: %d\t Hexadecimal: %4x\t Octal: %8O\n",
$valor,$valor,$valor, $valor);
}
```

Problema 2:

a) Preencha as entradas em falta na tabela que a seguir se apresenta

Decimal	Binary	Octal	Hexadecimal
0	00000000		00
55			
136			
243			
-245			
	01010010		
	10101100		
	11100111		
			A7
			3E
			BC

b) Use o programa **cvBases** atrás apresentado para confirmar os resultados apresentados na alínea anterior, usando uma sintaxe equivalente ao exemplo que a seguir se apresenta:

```
> ./cvBases 100 0x34 0x23AF 1023
```

Representação de Objectos em Memória

No que segue vamos tomar como base a linguagem de programação C e os seus construtores para mostrar a representação de diferentes tipos de objectos na memória de um computador.

1. Tipos de Dados

Os computadores e os compiladores suportam múltiplos formatos de dados a que correspondem diferentes espaços da memória reservada para os conter.
O número exacto de octetos usados para codificar um tipo de dados expresso em linguagem C depende da máquina usada e do compilador.

Tamanhos
Para permitir o acesso e a visualização da representação em octetos (*bytes)*, de diferentes tipos de objectos em memória recorreremos às facilidades de moldagem fornecidas pelo operador ***cast*** . Através do uso daquela facilidade podemos referenciar um mesmo objecto usando tipos de dados diferentes do tipo de dados definido no momento da criação do mesmo.

Problema 1:
a) Preencha os espaços em branco da coluna "`Tamanho em Linux-32-bit`"

Declaração em C	Tamanho em Linux-32-bit
`char`	
`short int`	
`int`	
`long int`	
`char *`	
`float`	
`double`	

b) Use o programa abaixo, escrito em linguagem C, para confirmar os valores apresentados.

```
1   #include <stdio.h>
2   int main(void)
3   {
4
5       int short sval = 12345;
6       int ival = (int) sval;
7       int lval = (long int) ival;
8       float fval = (float) ival;
9       double dval = (double) fval;
```

```
10          int *pval = &ival;
11          char cval = (char)*pval;
12
13          printf ("\n");printf ("\n");
14          printf ("|---------------|----------------|\n");
15          printf ("|Declaracao em C|Tamanho em Linux|\n");
16          printf ("|---------------|----------------|\n");
17          printf ("|           char|----------%1d-----|\n",sizeof(cval));
18          printf ("|      short int|----------%1d-----|\n",sizeof(sval));
19          printf ("|            int|----------%1d-----|\n",sizeof(ival));
20          printf ("|       long-int|----------%1d-----|\n",sizeof(lval));
21          printf ("|---------------|----------------|\n");
22          printf ("|          int *|----------%1d-----|\n",sizeof(pval));
23          printf ("|---------------|----------------|\n");
24          printf ("|          float|----------%1d-----|\n",sizeof(fval));
25          printf ("|         double|----------%1d-----|\n",sizeof(dval));
26          printf ("|---------------|----------------|\n");
27          return(0);
28      }
```

Dica: Para criar o executável, pode usar o seu programa de edição favorito para criar um ficheiro, de nome *tamTipos.c*, com aquele conteúdo. Seguidamente crie um ficheiro makefile com o conteúdo apresentado em produção e executar o comando que segue:

```
> make tamTipo
```

Dados em Memória

A partir da combinação das possibilidades oferecidas pelo uso do operador `cast` e da declaração `typedef`, o código que segue pode ser utilizado para estudar a representação de tipos de dados em memória.

```
 1      /* Representacao de tipos de dados em  memoria*/
 2      #include <stdio.h>
 3      extern char octetos[];
 4      typedef unsigned char *byte_pointer;
 5      void mostra_octetos(byte_pointer start, int len)
 6      {
 7          int i;
 8          char *p=octetos;
 9          for (i = 0; i < len; i++)
10            sprintf(p+i*3,"%.2x ", start[i]);
11          p=p+i*3;
12          for (; i<len; i++)
13            sprintf(p++," ");
14          sprintf(p,"|\n");
15
16      }
17      void mostra_int(int x)
18      {
19          mostra_octetos((byte_pointer) &x, sizeof(int));
20      }
21      void mostra_short(short x)
22      {
23          mostra_octetos((byte_pointer) &x, sizeof(short));
24      }
25      void mostra_float(float x)
26      {
27        mostra_octetos((byte_pointer) &x, sizeof(float));
28      }void mostra_double(double x)
29      {
30          mostra_octetos((byte_pointer) &x, sizeof(double));
31      }
```

```
32      void mostra_pointer(void *x)
33      {
34          mostra_octetos((byte_pointer) &x, sizeof(void *));
35      }
```

Note-se, na linha 4, a definição de um novo tipo de dados `byte_pointer` que cria um apontador para um objecto do tipo `unsigned char`.
A função `size_of(TipodeDados)`tem um papel importante no desenvolvimento de código portátil entre diferentes arquitecturas, na medida em que, permite obter valores dependentes da plataforma e do compilador usados em cada caso.

Problema 2:
a) Preencha os espaços em branco da tabela que segue:

Valor	Tipo	Octetos (Hex
12345	`short`	
12345	`int`	
12345	`float`	
12345	`double`	
12345	`int *`	

b) Repita o exercício para valores diferentes de entrada, por exemplo `0x7FFFFFFF`

c) Use o código C que a seguir se transcreve para verificar a correcção das soluções encontradas para as alíneas anteriores.

```
 1      #include <stdio.h>
 2      //variavel usada para conter a representacao textual
 3      char octetos[25];
 4      //Prototipos de varias funcoes para visualizar a representacao
 5      //em memoria de varios tipos de dados
 6      void mostra_int(int x);
 7      void mostra_short(short x);
 8      void mostra_float(float x);
 9      void mostra_double(double x);
10      void mostra_pointer(void *x);
11      int  main(int argc, char *argv[])
12      {int ival;
13       short sval;
14       float fval;
15       double dval;
16       int *pval;
17          if (argc > 1) {
18            //distingue na entrada entre decimal e hexadecimal
19            if (argv[1][0] == '0' && argv[1][1] == 'x')
20            sscanf(argv[1]+2, "%x", &ival);
21            else
22            sscanf(argv[1], "%d", &ival);
23
24          }
25          //Converte entrada nos varios formatos disponiveis
26          sval = (short) ival;
27          fval = (float) ival;
28          dval = (double) ival;
29          pval = &ival;
30          printf ("\n");
31          printf ("|---------------|\n");
32          printf ("|    Valor      |\n");
33          printf ("|---------------|\n");
34          printf ("| %8.4x      |\n", ival);
35          printf ("|---------------|------------------------|\n");
```

```
36          printf ("|Declaracao em C|   Memoria em Linux     |\n");
37          printf ("|---------------|-----------------------|\n");
38          mostra_short(sval);
39          printf ("|          short|%26s",octetos);
40          mostra_int(ival);
41          printf ("|            int|%26s",octetos);
42          mostra_float(fval);
43          printf ("|          float|%26s",octetos);
44          mostra_double(dval);
45          printf ("|         double|%26s",octetos);
46          mostra_pointer(pval);
47          printf ("|        pointer|%26s",octetos);
48          printf ("|---------------|-----------------------|\n\n");
49          return(0);
50      }
```

Nota: o programa recebe como parâmetro de entrada um número inteiro com sinal em formato decimal ou um valor em hexadecimal.

Ambiente de trabalho

No que se segue faz-se uma revisão breve da informação necessária à utilização do sistema Linux e de algumas das ferramentas que o acompanham e fornecem-se, também, referências para outras fontes de informação.

Existem múltiplas fontes de informação mais completas acerca dos temas relacionadas com o sistema Unix. Um ponto de acesso na Internet adequada ao acompanhamento da disciplina pode ser alcançado através da ligação a http://heather.cs.ucdavis.edu/~matloff/unix.html.

Os tópicos abordados nesta secção incluem: Interface, Sistema de Ficheiros, Comandos, Pseudónimo, Permissões, Variáveis de Ambiente, Redireccionamento, Encaminhamento, Substituição de Comandos, Programação, Compilação e Produção.

Interface

A interface com o computador hospedeiro faz-se através de um programa de sistema (imediatamente activo após a validação do utilizador) que actua de forma transparente para o utilizador, de forma a: i) ler comandos do teclado e informação de posição do rato; ii) executar as acções correspondentes à execução dos programas e iii) apresentar no ecrã os resultados pretendidos.

O mais popular daqueles programas de sistema em Unix é o *Bourne shell* (sigla **sh)**. Existem actualmente muitas outras variantes modernizadas e enriquecidas daquele *software* de sistema, sendo uma das mais conhecidas a **bash,** presente no ambiente Linux.

Comandos

O utilizador estabelece a interface com o sistema de exploração através da execução de comandos. Existem dois tipos de comandos em Unix: os internos (pré-construídos) e os externos (executáveis em directórios). Os comandos de interface mais habituais em Linux são os seguintes.

Sigla	Parâmetros	Exemplo	Significado
mkdir	<NomeDirectório>	`mkdir ~/descente/pontoA`	Cria directório
cd	<NomeDirectório>	`cd ../acima/aqui`	Altera directório actual
pwd		`pwd`	Identifica o directório actual
ls	[-a,-F…] directório>	`ls –a /`	Mostra todos os ficheiros na raiz
mv	f1, f2	`mv f1 ../f2`	Ficheiro *f1* é renomeado *f2* e movido para o directório ascendente
cp	f1, f2, .., fn /ali/directório	`mv f1 f2 destino`	Copia os ficheiros *f1,f2,..,fn* para o directório *destino*
chmod	modos permissões	`chmod o+rwx fich1`	As permissões de acesso a *fich1* incluem, para todos, a leitura, a execução e a escrita
less	<NomeFicheiro>	`less d1/d2/f1`	Visualiza o conteúdo de *f1*
exit		`exit`	Abandona a sessão corrente
man	<NomeDeComando>	`man ls`	Ajuda para o comando ls

Pseudónimo

Uma possibilidade interessante para a definição de comandos personalizados é a utilização do comando `alias` que permite criar um pseudónimo para um novo comando a partir de comandos anteriormente definidos.

```
alias ls "ls -F"
alias dir “ls –a”
alias more “less”
alias raiz “dir /”
```

Sistema de Ficheiros

Depois de feita a validação do utilizador dá-se o arranque automático de uma instância da *bash* que é imediatamente “posicionada” num directório inicial, atribuído pelo sistema a cada um dos utilizadores. Este é o directório usado, por omissão, pela *bash*, como referência para a leitura de ficheiros de entrada de dados e a escrita de ficheiros de resultados.

O utilizador pode, a todo o momento, alterar a definição da posição de leitura e de escrita dos ficheiros usados pela *bash* e pelos programas executados, através de comandos específicos. Os comandos usam uma notação própria para a identificação dos elementos presentes no sistema de ficheiros do sistema hospedeiro.

O símbolo “**~**” é usado como um substituto para a identificação da posição do directório inicial de arranque que em Unix é um nodo de um sistema de ficheiros hierárquico, descrito como uma árvore invertida em que o topo da árvore, designado por raiz, é representado pelo símbolo “***/***” isolado.

Qualquer ponto da árvore pode ser alcançado pela definição de um caminho construído tomando como referência um ponto inicial (a raiz (*/*), o directório corrente (**.**), o director ascendente (**..**), ou o directório inicial (**~**)), sufixado por uma lista de nomes (designativa de sub-directórios descendentes) separados pelo símbolo “***/***”.

Permissões

As permissões para acesso aos ficheiros (*quem*?) são determinadas por uma combinação de propriedades definidas através dos caracteres [u,g,o]. Para o criador (u), para o grupo de que faz parte (g) e para todos os outros (o).

Para definir os modos de acesso (*o quê?)* usam-se os caracteres [r,w,x]. Para leitura (r), para escrita (w) e para execução (x); no caso de um directório, possuir a permissão (x) isso quer significar que pode ser atravessado, isto é, ser incluído da definição de um caminho.

São habitualmente usados os comandos `chmod` e `chgrp` para definir as permissões de acesso a ficheiros.

Um utilizador para poder executar um programa contido num ficheiro, tem de possuir permissões compatíveis com os atributos do mesmo, no sistema de ficheiros. Assim, para uma ou mais das classes de utilizadores a quem é garantido o acesso tem de estar, obrigatoriamente, definidos os atributos de **leitura** e de **execução**.

Variáveis de Ambiente

A execução dos programas em Unix faz uso de definições de ambiente construídas a partir dos valores atribuídos a **variáveis** de sistema, criadas automaticamente no momento de arranque. Através de comandos específicos, o utilizador pode criar novas variáveis, ou visualizar e alterar os valores já atribuídos. Exemplos de variáveis pré-definidas e respectivos valores iniciais são:

```
LOGNAME="amp"
HOME="/home/amp",
PATH="/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/amp/bin"
SHELL="/bin/bash"
LANG="pt_PT.UTF-8"
```

Tomando como referência a *bash*, para visualizar o valor de uma determinada variável usa-se um comando apropriado e o nome da variável prefixado com o símbolo **$** como em:

```
echo $PATH
```

Uma nova variável pode ser criada localmente no ambiente actual, ou ser definida globalmente como se pode ver nas seguintes definições:

```
PATH=$PATH:/usr/novoBin
export MINHA=”/usr/outro/Caminho”
PATH=$PATH:$MINHA
```

Redireccionamento

Habitualmente os programas enviam resultados para o ecrã (*stdout*) para poderem serem examinados. Fazendo uso de uma facilidade conhecida como redireccionamento é possível preservar aqueles resultados sob a forma de um ficheiro. Assim, qualquer comando pode usar o símbolo “**>**” seguido do nome de um ficheiro para redireccionar toda os resultados para o ficheiro correspondente, como é o caso em:

```
dir > listaDedirectórios

pwd > directórioCorrente
```

Encaminhamento

Uma forma de redireccionamento mais poderosa é a que permite o encaminhamento dos resultados de saída de dados (*stdout*) produzidos por um determinado programa (comando) para a entrada de (*stdin*) de outro comando e assim sucessivamente, através do uso do símbolo “**|**”.

Vejamos como poderíamos saber qual o número de ficheiros de um determinado directório de arranque usando o comando **wc** que permite contar o número de linhas e caracteres existentes num determinado ficheiro.

```
alias contaLinhas “wc –l”
dir  ~/esteDirectório | contaLinhas
```

Substituição

O resultado produzido por um determinado programa pode ser usado como entrada de outro programa usando um mecanismo de substituição disponível através da utilização de um par de plicas “` `” que delimitam o comando que se pretende substituir. Por exemplo para listar todos os ficheiros do directório inicial pode usar-se:

```
dir `pwd`
```

Programação

Linguagens como o C e o C++ oferecem aos programadores níveis muito profundo de programação que visam obter reduzidos tempos de execução para os executáveis programas. Contudo, em muitos domínios de aplicação o tempo de execução não é o principal factor a considerar.

A escrita de programas usando linguagens interpretadas de alto-nível é muitas vezes mais conveniente quando se pretende:

- Prototipificação rápida
- Evitar a declaração de tipos de variáveis
- Usar construtores de alto-nível
- Evitar a compilação

Uma das linguagens de programação interpretadas mais popular nos nossos dias é, provavelmente, o **Perl**. Para além da sua evidente capacidade expressiva e elegância, a semelhança com o C faz com que possa ser usada como uma linguagem de especificação rápida, cujos programas podem facilmente ser traduzidos para C.

Compilação

Para executar um programa escrito em linguagens como o C é necessário que o código fonte desenvolvido seja previamente compilado para produzir um ficheiro contendo o código máquina apropriado para correr em cada sistema de computação concreto.
Em Linux a compilação é, normalmente, levada a cabo através de um comando gcc emitido da consola que obedece ao formato geral que segue.

```
> gcc -Wall –O2 –o progExecutavel funcao-1.c funcao-2 objFuncao-3.o
```

A opção -***Wall*** acciona a geração automática de mensagens de diagnóstico que descrevem a existência de imprecisões ou potenciais fontes de erro exitentes em cada um dos módulos, a opção -***O2*** indica ao compilador que deve usar o nível 2 de optimização do código.

O programa final executável, resulta da ligação de vários módulos que podem estar em estágios diferentes de processamentos. Notar que em linguagem C, um e só um dos módulos pode conter na sua definição uma função com o nome main. A opção ***–o*** refere o nome do ficheiro resultante da execução do comando. Opcionalmente pode ser usada a opção -**S** para indicar que se pretende produzir código em linguagem de montagem (extensão .s)

Produção

A utilização de directivas de compilação é o modo preferido quando a complexidade de um programa constituídos por múltiplos ficheiros de código fonte é gerida através de ferramentas como é o caso do `make`.
Neste caso é preferível reunir aquelas directivas num ficheiro (habitualmente designado por **makefile**) com um conteúdo semelhante ao que segue.

```
CC = gcc
```

```
CFLAGS = -Wall -O2 -I .

PROGS =repDados\
      tamTipos\

all: $(PROGS)

repDados:  mostra_octetos.c repDados.c
      $(CC) $(CFLAGS) mostra_octetos.c repDados.c -o repDados

tamTipos: tamTipos.c
      $(CC) $(CFLAGS) tamTipos.c -o tamTipos

clean:
      rm -f $(PROGS) *.o *~core
```

O ficheiro `makefile` contém directivas capazes de, de uma forma fácil e coerente, garantir que todas as alterações aos fragmentos de código que compõem os programas são devidamente consideradas durante a geração dos executáveis:

Se considerarmos a existência, num determinado directório, de um ficheiro ***makefile*** com o conteúdo acima, a evocação do programa ***make***, evocado na linha seguinte, teria como resultado a criação (actualização) dos executáveis deduzíveis na directiva all que por seu lado, automaticamente, expande ***$(Progs)*** na lista de nomes especificadas através da variável ***Progs***:

```
>make
```