Instituto Superior de Engenharia de Lisboa

LEETC – LEIRT

Programação II

2019/20 – 2.º semestre letivo

Primeira Série de Exercícios

1. Armazenamento de inteiros em binário

O programa seguinte é destinado a recolher, de *standard input*, uma lista de inteiros e depositar os respetivos valores, em binário, num ficheiro identificado pelo argumento de linha de comando.

```
#include <stdio.h>

int main( int argc, char *argv[] ){
     int v;
     FILE *f = fopen( argv[1], "wb" );
     while( scanf( "%d", &v ) == 1 )
          fwrite( &v, sizeof v, 1, f );
     fclose( f );
     return 0;
}
```

Identifique os erros de execução que podem ocorrer devido a parâmetros incorretos. Corrija o programa de modo a controlar as situações identificadas. Quando ocorrerem, deve apresentar uma mensagem de erro, através de `stderr` (*standard error*) e produzir um código de insucesso no valor de retorno: 1, se tiver parâmetros errados; 2, se houver falha de acesso ao ficheiro.

Teste o programa e observe o formato dos ficheiros produzidos, usando o utilitário `hd` (*hexadecimal dump*). Verifique nomeadamente a representação dos valores, positivos ou negativos, e a ordem de armazenamento dos *bytes* com a convenção *little endian* usada pela arquitetura Intel-64.

2. Apresentação em binário do conteúdo de ficheiros

Pretende-se desenvolver um utilitário, designado por `hbd` (*hexadecimal and binary dump*), semelhante ao `hd`, mas com a apresentação em hexadecimal e em binário.

Pode receber, na linha de comando, um ou dois argumentos: O primeiro é obrigatório e indica o nome do ficheiro de entrada, cujo conteúdo binário vai exibir através de *standard output*; O segundo é opcional e indica a dimensão dos elementos de dados a observar no ficheiro. As opções do segundo argumento são indicadas na tabela seguinte. Por omissão é considerada a seleção de *word*.

Opção na linha de comando	Significado	Dimensão do elemento (*bit*)
"b", "B" ou "1"	*Byte*	8
"h", "H" ou "2"	*Halfword* (2 *byte*)	16
"w", "W" ou "4"	*Word* (4 *byte*)	32
"d", "D" ou "8"	*Doubleword* (8 *byte*)	64

O programa apresenta em cada linha uma palavra com a dimensão selecionada, exibindo as duas representações: hexadecimal e binário. Nas representações numéricas de cada palavra deve produzir, da esquerda para a direita, os algarismos do maior peso para o menor peso.

Valoriza-se a portabilidade para diferentes plataformas, apoiada no uso do operador `sizeof` e nas definições disponíveis em "`limits.h`"

2.1. Escreva a função

```
void print_bin( unsigned long long value, int size );
```

que apresenta, em *standard output*, a representação na base 2 dos `size` bits de menor peso de `value`.

Na apresentação dos valores em binário, deve separar em grupos de bits, colocando um ponto entre grupos de quatro bits e um espaço entre grupos de oito bits. Por exemplo, um inteiro com o valor 0x123456ab, deve ser exibido como "`0001.0010 0011.0100 0101.0110 1010.1011`".

2.2. Escreva o programa `hbd` especificado, usando a função `print_bin` para apresentar os valores na base 2 e a função `printf` para a base 16. Para testar o programa propõe-se que utilize ficheiros de dados gerados com o programa do exercício 1.

3. Formatação uniforme de nomes

Pretende-se o desenvolvimento de *software* para processamento de nomes, que modifique as palavras contidas numa *string*, convertendo a primeira letra para maiúscula e as restantes para minúscula. Deve ter também a possibilidade de colocar algumas palavras totalmente em minúsculas. Admite-se que os nomes a processar não têm acentos.

Por exemplo, pretende-se converter: "`alberto dos santos  `" para "`Alberto dos Santos`"; "`MANUEL DA COSTA`" para "`Manuel da Costa`"; "`joaquim campos e silva`" para "`Joaquim Campos e Silva`".

3.1. Escreva a função

```
int verify(char *nameWord, char *lowerWords[], int numWords );
```

que, recebendo em `nameWord` uma palavra pertencente a um nome, verifica se ela é idêntica a alguma das *strings* indicadas pelos ponteiros do *array* `lowerWords`. O parâmetro `numWords` é o número de *strings* indicadas por `lowerWords`. Em caso afirmativo retorna 1; se não, 0.

3.2. Escreva a função

```
void upper_first(char *nameWord, char *lowerWords[], int numWords );
```

que, recebendo em `nameWord` uma palavra pertencente a um nome, converte-a para a forma padrão: a primeira letra em maiúscula e as restantes para minúscula; exceção – se a palavra existir nas *strings* indicadas por `lowerWords`, deve ficar toda em minúsculas. Deve utilizar a função anterior.

Por exemplo, se na invocação de `upper_first` o array `lowerWords` indicar as palavras `"de"`, `"da"`, `"do"`, `"dos"` e `"e"`, a tabela seguinte indica alguns resultados do processamento:

Palavra original indicado por nome	Palavra após o processamento
"alberto"	"Alberto"
"dos"	"dos"
"MANUEL"	"Manuel"
"DA"	"da"

3.3. Escreva a função

```
void unifyName(char *name, char *lowerWords[], int numWords );
```

que, recebendo em `name` um nome formado por várias palavras, separadas, antecedidas ou sucedidas por um ou vários espaços ou *Tabs*, reproduz a *string* com as palavras convertidas para a forma padrão e separadas por um espaço, sem espaços no início nem no fim. Para isso, a função identifica as palavras, usando a função `strtok`, converte-as para a forma padrão, usando a função `upper_first`, e reconstrói a sequência. O parâmetro `lowerWords` aponta para *strings* com as palavras que, se existirem na *string* `name`, devem ficar totalmente em minúsculas.

Por exemplo, se na invocação de `unifyName` o array `lowerWords` indicar as palavras `"de"`, `"da"`, `"do"`, `"dos"` e `"e"`, a tabela seguinte indica alguns resultados do processamento:

Nome original indicado por `name`	Nome após o processamento
`"  alberto  dos santos  "`	`"Alberto dos Santos"`
`" MANUEL DA  COSTA "`	`"Manuel da Costa"`

Para utilizar a função `strtok` da biblioteca normalizada deve ler a respetiva documentação. Recomenda-se também que estude as restantes funções declaradas no *header file* "string.h" e utilize as que considerar convenientes.

3.4. Escreva um programa de teste que apresente, em *standard output*, o resultado do processamento de cada linha recebida de *standard input*.

Propõe-se que leia de *standard input* com `fgets`, admitindo que cada linha não tem mais do que 255 caracteres. Pode utilizar as funções de biblioteca que considerar convenientes, tanto nas funções como no programa de teste.

4. Ordenação de nomes

Pretende-se o desenvolvimento de um programa que receba um conjunto de nomes, através de *standard input*, e os reproduza através de *standard output*, uniformizados e ordenados alfabeticamente. Deve reutilizar as funções desenvolvidas para o exercício anterior e desenvolver o código necessário para o objetivo especificado.

Propõe-se a seguinte estratégia no funcionamento do programa:

- Leitura dos nomes, normalização do formato e armazenamento num *array*;
- Ordenação do *array* preenchido com os nomes;
- Apresentação dos conteúdos por ordem.

Admite-se, por simplificação, que há um número limitado de nomes e que cada um tem um número limitado de carateres, pelo que pode ser criado um *array* com dimensões fixas para armazenar os nomes.

Considere a definição do *array*,

```
char names[MAX_ NAME_COUNT][MAX_NAME_SIZE];
```

Para ajudar a definir as dimensões pode avaliar o maior ficheiro de nomes que pretende ensaiar, podendo aplicar o utilitário *word-count* com o comando "`wc -L` *filename*".

4.1. Escreva a função

```
int fillName(char *name, FILE *stream );
```

que lê, a partir da *stream* indicada, um nome, o qual uniformiza e armazena na *string* indicada apor `name`.

Para ler, deve usar a função `fgets` da biblioteca normalizada; para uniformizar o nome, deve utilizar as funções do exercício anterior. Retorna: 1, em caso de sucesso; 0, se ocorreu *end-of-file* na leitura.

4.2. Escreva a função

```
void sortNames( char name[][MAX_NAME_SIZE], int count );
```

que ordena o *array* name, por ordem alfabética crescente. O parâmetro count é o número de elementos preenchidos no *array*.

Para ordenar o *array*, deve utilizar a função `qsort` da biblioteca normalizada. É necessário escrever uma função de comparação que identifica a ordem relativa entre duas *strings*. Esta função é passada por parâmetro à função `qsort`.

4.3. Escreva o programa de aplicação, implementando a estratégia proposta com recurso às funções anteriores.

ISEL, 2020.03.30