Mestrado em Engenharia de Automação Industrial

Elementos de Programação

Relatório do Trabalho Laboratorial nº1

Docente:

Armando Pinho

Discentes:

Bruno Silva

João Nogueiro

João Lopes

Ex1.

Programa: print_bit.c

Descrição: O utilizador digita dois valores inteiros no teclado: *a*, um número decimal e *b*, o bit posição pretendido. O programa retorna o valor binário (0 ou 1) referente à posição, *b*, da representação binária de *a*. O bit menos significativo corresponde à posição 1.

Execução do programa:

Compilação e Execução

```
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex1$ gcc print_bit.c
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex1$ ./a.out
```

Por exemplo com o a=10 e b=4

```
Insert a number
10
Insert a bit position starting at p=1
4
```

Output:

```
The 4'th leat significant bit is 1
```

Neste exemplo, como foi executado no programa para o número a=10, o quarto bit menos significativo é 1.

Do bit menos significativo 0 (b=1) para o bit mais significativo 3 (b=4).

a = 10 **bits**

3	**2**	**1**	**0**
1	0	1	0

```
printf("The %d'th leat significant bit is %d\n", b , ( (a & (1<<(b-1)) ) >>(b-1)));
```

O programa vai executar deslocamento bit a bit, o valor 1 *shift left b-1* vezes e compara com o valor de *a* utilizado o comparador AND (&). De seguida o programa utiliza shift right para obtermos o resultado final 0 ou 1.

Por exemplo com o a = 10 e b = 4

a=10 1010

1 < < (4-1) & 1000

1000

(10 & (1 < < (4-1)) > > (4-1) 1 (RESULTADO)

Nota: Nos scanfs é necessário que o input seja um endereço de memória e não um valor. Ao contrário de muitas linguagens como o python que passam argumentos “by assignment”, o C por default passa “by value”, ou seja uma cópia do input com o mesmo valor noutro endereço de memória qualquer (incluída no stack frame da função scanf neste caso). Se queremos alterar o valor da variável inicial temos de dar como argumento a posição de memória onde a variável está armazenada (&var).

Ex2.

Programa: print_bits.c

Descrição: O utilizador digita um número inteiro decimal e o programa retorna em formato binário.

Execução do programa:

Compilação e Execução

```
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex2$ gcc print_bits.c
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex2$ ./a.out
```

Por exemplo, a = 10

```
Insert a number
10
```

Output

```
00001010
```

Logica:
A função binary() vai comparar bit a bit, utilizando o operador AND (&):

exemplo para a=10 para a primeira iteração do ciclo for, i = 10000000

```
    1000 0000
&   0000 1010
    0000 0000  (RESULTADO)
```

Pelo que o 8 digito significativo da representação binária de 10 vai ser 0, como é esperado

Utilizando o operador ?:

expressão condicional **?** expressão1 **:** expressão2;

Caso a condição for verdadeira, a expressão 1 será resultado da expressão condicional, caso contrário, resulta na expressão 2.

```
void binary(unsigned value){
    unsigned i;

    for(i=1<<7 ; i>0 ; i=i>>1 )
    {
        (value & i) ? printf("1") : printf("0");
    }
}
```

Este programa, está executável até 8 bits, ou seja 256 combinações, do número 0 a 255. No entanto, é possível ir até números maiores, mudando o tipo da variável i, de unsigned int para unsigned long int por exemplo. Mesmo sem mexendo ao tipo da variável, com 8 bits só estamos a usar ¼ da memória alocada de um int (4 bytes) , pelo que poderíamos ir até números maiores. (max i<<31). Aumentar este valor tem a contrapartida de fazer o programa mais lento, pelo que optamos por um valor arbitrário intermédio.

Ex3.

Programa: bits_to_int.c

Descrição: O utilizador digita um número no formato binário e o programa retorna o correspondente valor decimal.

Execução do programa:

Compilação

```
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex3$ gcc bits_to_int.c
```

Execução

Ao executar o programa o utilizador terá que colocar logo a representação binário do número que pretende saber, da seguinte forma:

./a.out valor_binario

Por exemplo com 00001010 (10 em decimal)

```
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex3$ ./a.out 00001010
```

Output

```
Your number in decimal is 10
```

Logica:

Primeiramente o programa verifica se o utilizador colocou os inputs necessários. Caso não sejam contabilizados dois intputs o programa é interrumpido. Nos exercícios anteriores como se usou o scanf invés dos argumentos da shell, os argumentos escritos a mais são apenas ignorados.

```
if(argc!=2){
    printf("ERROR - too many or no inputs\n");
    return 1;
}
```

De seguida verifica se o argumento argv[1], que corresponde ao valor binário, é apenas composto por 0's e 1's, caso não seja, o programa é interrumpido.

```
int i;
for(i=0; argv[1][i]!='\0' ; i++) {

    if( (argv[1][i]-'0') != 1 && (argv[1][i]-'0') != 0){
        printf("Input must consist of 1's or 0's\n");
        return 1;
    }
```

Na conversão, o ciclo *for* percorre as *i* posições do *input* binário, começando no bit menos significativo para o mais significativo, isto é, da direita para a esquerda, S_length-i.
Primeiro o programa retira o valor *mp* que apenas pode tomar o valor (0 ou 1), *char* convertido para *int*, devido à subtração do '0'. Quando fazemos operações aritméticas com chars, como subtrações, acontece uma 'integer promotion' pelo que o resultado de *mp* é um integer.

O valor de *mp* é multipicado por *exp* e somado à variável *decimal*.
O *exp* nada mais é que o equivalente a termos o calculo da potência com base 2, *exp* começa com o valor 1 e vai tomando valores de 1, 2, 4, 6, 8,...

```
int S_length = i;
int exp=1;
int decimal=0;
int mp;

for(i=0 ; i<S_length ; i++){

    mp = argv[1][S_length-1-i] - '0';
    decimal= decimal + mp * exp;
    exp = exp*2;
}
```

Ex4.

Programa: bits.c

Descrição: Este programa, junta as funcionalidades dos dois ultimos programas.

Execução do programa:

Compilação

```
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex4$ gcc bits.c
```

Execução

Ao executar o programa o utilizador terá escolher o modo de conversão que pretende:
b-d" → conversão de binário para decimal ou "d-b" → conversão de decimal para binário.

./a.out d-b OU b-d

```
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex4$ ./a.out b-d
Please type a binary number to convert
```

```
joaonogueiro@jn:~/Documents/EP/LabWork1Eprog/Ex4$ ./a.out d-b
Please type a decimal number to convert
```

Logica:

Grande parte da lógica de programação está discrita no Ex3 e Ex2. O que difere neste programa é que ambas as funções estão encapsuladas em duas funções isoladas. Isto permite o código ser modular e cada função ter uma porção do stack dedicada a si (stack frame) , volátil, pelo que quando acaba a execução limpa todas as variáveis locais, sendo mais eficiente na gestão de memória.

O uso do strcmp é necessário, uma vez que as strings são char* 'in disguise', um array especial que acaba com o caractér '\0'. Genericamente, quando fazemos refererência direta a um pointer, estamos a referenciar a posição de memória que nele está guardada. Assim ao fazer str1=str2, apenas comparamos os endereços de memória dos dois pointers, que não é o que queremos aqui.

Ex5/6.

Aqui vão ser explicados os dois ao mesmo tempo, porque complementam-se um ao outro.

Programa: test_bit_stream.c

Descrição: O programa lê um ficheiro especificado pelo utilizador e faz a codificação ou descodificação binária-ascii.

Compilação

```
gcc test_bit_stream.c de_encoder_fc.c
```

Aqui o test_bit_stream.c é o main file, o de_encoder_fc.c é o módulo onde tem todas as funções necessárias para dar decode e encode, e é onde estão incluídas as funções do Ex.5.

Execução

```
clear && ./a.out
```

```
*[devBruno][~/Documents/EProg/LabWork1Eprog/BitStream/V2]$ ./a.out decode
Please type the path of the binary file to decode
```

ou

```
*[devBruno][~/Documents/EProg/LabWork1Eprog/BitStream/V2]$ ./a.out encode
Please type the path of the text file to encode
```

Logica:

Na test_bit_stream.c apenas está presente o main, onde são chamadas as diferentes funções, definidas no módulo de_encode_fc.c.

Para incluir a biblioteca feita por nós, é necessário fazer *#include "de_encode_fc.h"*, o header file do módulo, onde apenas estão definidos os *typedef's* das estruturas e os prototype's das funções. As aspas aqui fazem o compilador procurar no diretório atual em vez do default onde se encontram as bibliotecas nativas como o *stdio.h.*

As include guards no início do ficheiro .h :

#ifndef BIT_STREAM_H

#define BIT_STREAM_H

….

#endif

São importantes para fazer o módulo idempotente, isto é, tem efeito a primeira vez que é executado, mas ignorado nas vezes seguintes. Isto previne que caso o módulo incluia bibliotecas que outros módulos ou até o main incluam, não sejam nunca declaradas mais que uma vez, que é inválido em C.

Voltando ao funcionamento do programa, analisando por exemplo a função de decoding.

Esta função pega num ficheiro binário, que quando visualizado apenas mostra caractéres aleatórios que codificam certas sequências binárias, alguns até non printable characters e escreve essas sequências binárias em caracteres ‘1’ e ‘0’ de modo a serem legíveis.

Por exemplo, se no ficheiro binário estiver um ‘a’, corresponde ao código ascii 0d97 que em binário seria 0b1100001.

A função decoder chama continuamente a função readbit para saber se deverá escrever um ‘0’ ou ‘1’ no ficheiro out.txt, enquanto para a função writeBit e readBit funcionarem bem, existe uma estrutura de dados “BitSream” que armazena o index do bit selecionado no ficheiro, um buffer para armazenar o byte atual, e um ficheiro para o qual deve ler ou escrever.

A função readBit guarda um byte numa posição de memória e avança a posição do ponteiro do ficheiro. Deve calcular no bit de memória onde o cursor está, devolvendo verdadeiro se o bit for 1 e falso se for 0, e em seguida incrementar o cursor. De acordo com o retorno da função,na função decoder o programa deve escrever o valor equivalente a '1' ou '0' (ASCII) de modo a escrever os carateres em modo texto. Quando o cursor chega à posição 0, a função decoder vai buscar mais um caracter ao ficheiro e dá reset à posição do cursor do byte.

Olhando agora para o modo de encoding.

Este converte um ficheiro texto para binário, apenas aceitando ficheiros de texto com ‘0’s e ‘1’s.

A função encoder vai buscar o carater ao ficheiro de texto, e depois consoante o valor, a função writeBit é chamada para escrever 1 ou 0 no bit selecionado.

A função writeBit guarda um bit no seu buffer e avança o cursor e o ponteiro do ficheiro, quando o cursor ultrapassa 8 escreve o byte no ficheiro, correspondendo a um qualquer caracter e esvazia o buffer. É usado no encoding.

O argumento 0 ou 1 da função deve ser obtido pelo caracter representado no texto.

Todas estas funções usam como argumentos pointers para BitStream, de modo a poderem ser acedidos em vários locais ao longo do programa. Deste modo quando são feitas alterações às BitStreams dentro das funções, altera as BitStreams defenidas no stack frame das “funções pai” que chamaram diferentes funções.