



Licenciatura em Tecnologias e Sistemas de
Informação
2019/2020

Relatório

Ficha de Trabalho Nº3

Jorge Miguel dos Santos Martins a2019100813

3 de Abril de 2020

Índice

1 INTRODUÇÃO.....	3
2 TRABALHO REALIZADO.....	4
2.1 EXERCÍCIO 1.....	4
2.2 EXERCÍCIO 2.....	5
2.3 EXERCÍCIO 3.....	5
2.4 EXERCÍCIO 4.....	7
2.5 EXERCÍCIO 5.....	8
3 CONCLUSÃO.....	10
1 REFERÊNCIAS.....	11
2 ANEXOS.....	12

1 Introdução

A ficha de trabalho número 3 é focada no manuseamento de ponteiros, dando assim conhecimento de como os utilizar e o que realmente esta opção na linguagem difere nas outras. Dando a conhecer as suas vantagens e os seus cuidados no seu uso.

Ponteiros tal como o nome indica é algo que aponta para, neste caso é algo que está apontar para um exato endereço de memória, sendo cada endereço único.

2 Trabalho realizado

2.1 Exercício 1

a)

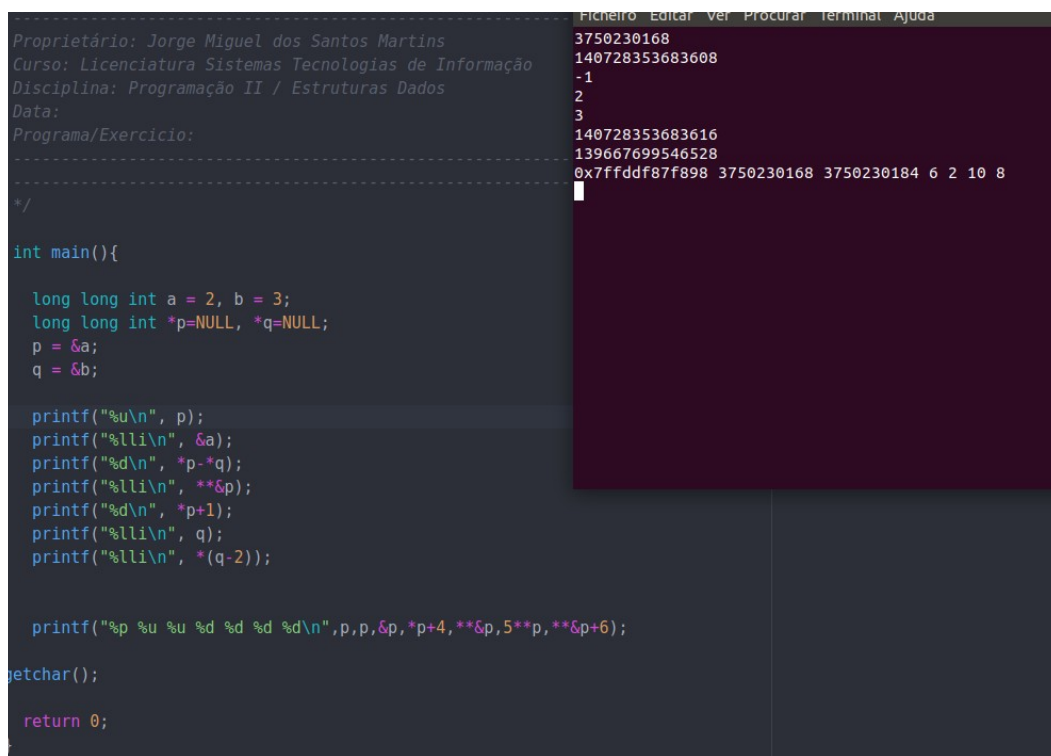
i. $p == \&a \rightarrow \text{True}$. Visto ser apenas o p e não $*p$, este contém o endereço de a , ou seja daria o resultado True.

ii. $*p - *q \rightarrow -1$

iii. $**\&p \rightarrow 2$

iv. $*p+1 \rightarrow 3$

v. $*(q-2) \rightarrow$ mostra(esta apontar) o valor o valor alocado em memória que está a menos dois tamanhos de inteiros em memória em relação ao espaço de memória de q .



```

Proprietário: Jorge Miguel dos Santos Martins
Curso: Licenciatura Sistemas Tecnologias de Informação
Disciplina: Programação II / Estruturas Dados
Data:
Programa/Exercício:
-----
*/

int main(){

    long long int a = 2, b = 3;
    long long int *p=NULL, *q=NULL;
    p = &a;
    q = &b;

    printf("%u\n", p);
    printf("%lli\n", &a);
    printf("%d\n", *p-*q);
    printf("%lli\n", **&p);
    printf("%d\n", *p+1);
    printf("%lli\n", q);
    printf("%lli\n", *(q-2));

    printf("%p %u %u %d %d %d %d\n",p,p,&p,*p+4,**&p,5**p,**&p+6);

    getch();

    return 0;
}

```

```

3750230168
140728353683608
-1
2
3
140728353683616
139667699546528
0x7ffddf87f898 3750230168 3750230184 6 2 10 8

```

Código e seu resultado

Para o exercício b, podemos ver o seu resultado na ultima linha do terminal que aparece na imagem acima. Ao qual o “%p” irá nos dar o valor dentro da variável mas no tipo Hexadecimal, neste caso está a mostrar o endereço de a , pois é o valor que está dentro de p .

Seguidamente podemos ver o endereço de a mas já como um inteiro, no seguinte observamos o endereço mas do ponteiro p .

Para o resultado “ $*p+4$ ” vai ser dado o valor apontado por p mais 4, o resultado é o que se observa 6. Em seguida como temos duas indicações de apontar uma apontando para o endereço de p e outra que em seguida o faz apontar para o valor inserido nesse ponteiro, é então escrito o valor de “ a ” que é 2. Seguidamente como temos um inteiro a multiplicar com o valor apontado por “ p ” então recebemos o resultado de 2×5 , sendo este 10.

Finalmente estando a apontar para o seu endereço de “p” com outra indicação que se aponte para o seu valor interno, sendo este o endereço de “a” é devolvido 2 ao qual soma com 6 dando assim ao resultado observado 8.

2.2 Exercício 2

Neste exercício concluímos que por diferentes sejam os tamanhos dados pelo sistema às variáveis, o tamanho do próprio ponteiro é constante, sendo este de 8 bytes no sistema que eu utilizei que foi Ubuntu.

Ficheiro	Editar	Ver	Procurar	Terminal	Ajuda
Tipo	Tamanho em bytes			Tamanho do Ponteiro e bytes	
Char	1			8	
Inteiro	4			8	
Real	4			8	
Double	8			8	

Tabela de tamanhos

2.3 Exercício 3

Nome Variável					a	b		i		j	
Endereço Físico	5		12		1001	1002		2020		2024	
Valores	1001		1002		5	12		1001		1002	
A I	1001		1002		12	5		1001		1002	
B I	1001		1002		5	12		1002		1001	
C I	1001		1002		5	5		12		1002	
A II	1002		1001		5	12		1001		1002	
B II	1001		1002		12	5		1001		1002	
C II	1001		1001		1002	12		1001		1002	
A III	1001		1002		12	5		1001		1002	
B III	1001		1002		5	12		1002		1001	
C III	1001		1002		5	5		12		1002	

A IV	1002	1001	5	12	1001	1002
Endereço Físico, mudam para onde estão apontar.	5(y) Apon ta	12(x) Apon ta	1001	1002	2020	2024
B IV	1001	1002	5	12	1001	1002
Endereço Físico, (muda para onde estão apontar.)	5	12	1001	1002 (x) Fica apont ar para aqui	2020	2024
C IV	1001	1001	5	12	1001	1002
A V	1001	1002	12	5	1001	1002
Endereço Físico, (muda para onde estão apontar.)	5	12	1001 (y) fica apont ar para aqui)	1002 (x) Fica apont ar para aqui	2020	2024
B V	1001	1002	5	12	1001	1002
Endereço Físico, (muda para onde estão apontar.)	5	12(x) fica a apont ar para aqui)	1001	1002	2020	2024
C V	5	12	5	5	1001	1002

Na resolução deste exercício optei pela realização de uma tabela ao qual em cada junção que o exercício nos impõe, sendo estas 3 versões da função troca e as 5 possibilidades de chamada de cada uma dessas funções, o que pensei foi simplesmente demonstrar o que é alterado em cada situação, marcando assim a vermelho as mudanças. As vezes que optei por mostra mais uma linha com o endereço é devido ao facto de algum ponteiro ter ficado simplesmente a apontar nesse endereço.

2.4 Exercício 4

Neste exercício foi implementado o uso de ponteiros para a manipulação de matrizes. Inicialmente foi pedido para iniciar uma matriz com -1, ao qual através de um ciclo for passando por todos os blocos de memória a cada posição da matriz inserimos o seu valor.

```
void inicializer(int tamanho, float *p){ //alinea a)

    for(int i=0; i<(tamanho*tamanho); ++i){
        *p=-1;
        p++;
    }
}

void inicializerCresc(int tamanho, float *p){ //alinea b))

    for(int i=0; i<(tamanho*tamanho); ++i){
        *p=(i+1);
        p++;
    }
}
```

Iniciadores de Matrizes

Aqui vemos os dois primeiros pontos do exercício. Os pontos a referenciar é que em C o espaço para as matrizes mesmo nós as visualizando como se fossem tabelas bidimensionais os espaços em memória que elas ocupam são sempre unilaterais, o que significa que “tamanho*tamanho” indica o número de blocos de memória que o ponteiro vai ter de percorrer. Primeiramente dando apenas o valor de -1 e no segundo dei o valor de i+1 fazendo assim de 1 a 400.

Para os outros dois pontos deste exercício que consistem na inserção de um valor numa linha e coluna à escolha do utilizador e a soma de duas matrizes, podemos observar o código da imagem abaixo.

Desde já referir que a localização da linha e coluna que o utilizador deseja para modificar o número é pedido no main e a matriz que decidi mudar o valor foi na matriz iniciada por -1.

```
void giveValue(int tamanho, float *p, int i, int j){ //alinea c)

    float val;

    printf("Insira o valor a inserir: "); scanf("%f", &val); printf("\n");
    p=p+(i*tamanho+j); //multiplica a linha desejada pelo total de colunas da matriz e em seguida soma-se a coluna desejada. Em c
    *p=val;
    flush_in();
}

void sum(int tamanho, float *p1, float *p2){

    for(int i=0; i<(tamanho*tamanho); ++i){
        *p1= *p1 + *p2;
        p1++;
        p2++;
    }
}
```

Seguindo o que já foi referido anteriormente em relação ao modo como a linguagem C aloca em memória as matrizes mesmo sendo bidimensionais, então para que o ponteiro seja devidamente posto a apontar no endereço certo é necessário ter alguma atenção. Usando um exemplo para explicar, imaginemos que o utilizador deseja mudar um valor que se encontra na linha 1 coluna 1, para que o apontador seja levado ao local correto sem usar (&matriz[linha][coluna]) para dar o endereço, mas sim que haja um manuseamento da memória, temos então de usar um pequeno calculo. Se o que queremos se encontra na primeira linha e primeira coluna e por exemplo é uma matriz 5x5 então de uma forma unilateral este endereço encontra-se na sexta posição, então será linha que o utilizador deseja vezes o tamanho de linhas que a matriz tem mais a coluna que o utilizador deseja. Neste caso seria $1*5+1$ (linha 1 vezes 5 linhas existentes mais 1 coluna) o que dá o resultado de 6.

Para o ultimo ponto deste exercício, no main envio os endereços das duas matrizes ao qual na função recebe-se como ponteiro, tendo assim dois ponteiros um para cada matriz, basta que o valor em uma das matrizes seja a soma dos dois valores apontados por cada ponteiro.

2.5 Exercício 5

Para o primeiro ponto deste exercício o qual nos é pedido para realizar uma função que realize o mesmo que a função strlen da biblioteca da string, usei um ponteiro que corre a string e uma variável de nome count que a cada incremento do ponteiro esta também incrementa, assim dando o tamanho da string. Como usei o scanf([]) este serve para ler strings e com a opção de se poder ignorar espaços, o que faz com que o scanf continue a ler mesmo que o utilizador use mais que uma palavra, assim desta forma não necessito de me preocupar com o '\n' que é um standard da função fgets.

Para a realização do segundo ponto deste exercício para a copia de uma string, peço ao utilizador que indique em que índice quer que se comece a copiar e quantos caracteres quer copiar.

Antes de iniciar o ciclo para fazer o ciclo percorrer a distancia na string e copiar o que vai apontando, "ajeito" o local para onde o ponteiro está a apontar, dando indicação para que este comece no índice que o utilizador indicou. No decorrer do ciclo inicio com a variável "i" com o valor inicial de 0 e a condição é que enquanto este for menor ao número de caracteres que o utilizador indicou a copiar, o ponteiro "*pa" que aponta para uma string auxiliar que irá receber os dados a copiar, irá receber os valores

apontados por “*p” que está a apontar desde o índice dado pelo utilizador e que irá percorrer a string até o fim do ciclo assim o ditar.
Notar que esta função devolve a string que recebeu os caracteres copiados para o main, assim estes sendo escritos no main.

```
char * copyString(char *src, int i, int len){
    char * auxstr=malloc(len+1), *pa=NULL;

    pa=&auxstr[0];
    src=src+i; // apontar para o índice de começo da gravação.

    for(int a=0; a<len; ++a){
        if(*src=='\0'){
            break;
        }
        else{
            *pa = *src;
            src++;
            pa++; // antes do ciclo acabar estes são sempre incrementados
        }
    }
    *pa='\0'; // estou a sinalizar o final da string com '\0' visto que ele estava lá
    return auxstr;
}
```

Função de copia que envia uma string

3 Conclusão

Para este relatório seguindo do que foi feito no anterior, escolhi apenas o que achei mais importante relatar no meu código, deixando também o meu código comentado onde achei essencial explicar as minhas decisões.

Desta ficha importante referenciar o exercício número 3 que nos faz realmente entender como os ponteiros funcionam e o porquê de existir toda uma forma correta de enviar os valores para que os ponteiros a recebam e o programa realize exatamente o que queremos, pois simplesmente ter um resultado que funciona não quer dizer que fez exatamente o que desejávamos. Com este exercício deu a possibilidade de conseguir pensar melhor como resolver os exercícios posteriormente propostos.

1 Referências

Listar consultas usadas

- https://en.cppreference.com/w/c/io/fread?fbclid=IwAR31yVh6M2HEG41x5F6z6edn9mSnnWtioDsOFZb4HHGRJS-b_nItU6SNAWA
- https://www.gnu.org/software/libc/manual/pdf/libc.pdf?fbclid=IwAR3i7GPGYKXIJN5ysbnU1xBT_Dq5TaQay60CBEfnrzNsS5M7J1XAP7cDgcE
- <https://www.quora.com/What-is-p-in-the-printf-function-in-C-programming-language>

2 Anexos

Lista de ficheiros de código:

ex1.c → Não é pedido o código, mas como usei como exemplo neste relatório envio, é apenas um teste ao que a professora pediu para fazer

ex2.c

ex4.c

ex5.c