Funções e códigos

Estruturas de dados

Para algumas das funções e códigos foram utilizadas Pilhas, do mesmo jeito poderiam ser utilizadas Filas, mas optei pelas pilhas pois foram suficientes para os exercícios. Como ainda não tenho conhecimento das estruturas Listas, acabei usando de pilhas para os exercicíos que pediam listas também. Seguem as estruturas e funções utilizadas para mexer com as pilhas:

Quando o exercício pedia utilização de números inteiros, a diferença aparecia na struct elemento que possuia um int dado ao invés de um char caracter e a função colocaDado recebia um inteiro além da pilha. Existe também uma variância de nomes das funções mas no geral foram usadas funções para colocar dado na pilha, retirar dado na pilha e imprimir a pilha, além das estruturas elemento, pilha e função comecaPilha que sem elas não seria possível usar tal estrutura.

```c
typedef struct elemento{
    char caracter;
    struct elemento *proximo;
} Elemento;

typedef struct pilha{
    Elemento *topo;
} Pilha;

Pilha *comecaPilha(){
    Pilha *p;
    p = (Pilha*)malloc(sizeof(Pilha));
    p->topo = NULL;
    return p;
}
```

```c
void colocaDado(char caracter, Pilha *p) {

    Elemento *novo = (Elemento*)malloc(sizeof(Elemento));

    novo->caracter = caracter;

    novo->proximo = p->topo;

    p->topo = novo;


}
```

```c
int retiraDado(Pilha *p) {

    if(p->topo != NULL){

        Elemento *aux = p->topo;

        p->topo = p->topo->proximo;

        free(aux);
        aux = NULL;
    }
    return 0;
}
```

```c
void imprimePilha(Pilha *p) {

    Elemento *aux = p->topo;

    while(aux != NULL){
        printf("%i ", aux->dado);
        aux = aux->proximo;

    }
    printf("\n");


}
```

Muito simples 1

Função distancia( );

Essa função recebe as coordenadas de 2 pontos (x1, y1, x2, y2, nessa ordem) e calcula a distância entre eles através da fórmula da distância usando as funções sqrt( ) e pow( ) da biblioteca math.h.

```c
void distancia(float x1, float y1, float x2, float y2){

    float distancia;

    distancia = sqrt(pow(x2-x1, 2) + pow(y2-y1, 2));

    printf("%.5f", distancia);

}
```

Muito simples 2

Função converte_temperatura( );

Essa função recebe uma temperatura em fahrenheit e a transforma em celsius.

```c
void converte_temperatura(float f){

    float c;

    c = (f - 32)/1.8;

    printf("%.2f", c);
}
```

Simples 1

Esse programa tem como objetivo somar o conteúdo de duas posições X e Y de um vetor de 10 posições. Primeiro ele recebe do teclado o vetor de 10 posições e depois as posições X e Y que querem ser somadas. O programa retorna a soma de vetor[x-1] + vetor[y-1].

```c
int main (void){

    int vetor[10];
    int i;
    int x, y;
    int soma = 0;

    for (i=0; i<10 ;i++){

        scanf("%i", &vetor[i]);
    }

    printf("Digite x: ");
    scanf("%i", &x);
    printf("Digite y: ");
    scanf("%i", &y);

    soma = vetor[x-1]+vetor[y-1];

    printf("Soma = %i", soma);

    return 0;
}
```

Simples 2

Função verificaPerfeito( );

Entende-se por número perfeito aquele cujo a soma dos divisores (tirando o próprio número) é igual ao número. Essa função recebe um número inteiro e começa um contador do 1 até a metade do número (maior divisor que não é ele mesmo), com passo 1, se o resto da divisão do número pelo contador i é igual a zero, o i é somado a variavel soma inicializada como 0. Ao final se a soma é igual o número a função retorna 1, se não é, retorna 0.

```
void verificaPerfeito(int numero){

    int soma = 0;
    int divisores[100];
    int i;

    for(i=1;i<=(numero/2);i++){

        if(numero%i==0){
            soma+=i;
        }
    }

    if (soma == numero){
        printf("1");
    }
    else{
        printf("0");
    }
}
```

Simples 3

Função encontra_primo( );

Essa função depende de estruturas de dados e funções de estruturas de dados explicadas anteriormente nessa documentação.

Essa função recebe uma pilha p1 de inteiros e, enquanto ela não estiver vazia, inicia um iterador i que começa em 1 e vai até o valor do dado no topo da pilha com passo igual a 1. Se o resto da divisão desse dado por i for igual a zero é somado uma unidade a variavel contador inicializada como 0. Se o contador for igual a 2 (apneas dois divisores, 1 e ele mesmo) esse dado é retirado da p1 e colocado em putra pilha chamada p2, senão, ele é apenas retirado da p1 e o contador volta a ser 0. Depois os dados voltam para a p1.

```
int encontra_primo(Pilha *p1){

    Pilha *p2 = inicializaPilha();

    int a, b, i;
    int contador = 0;

    while(p1->topo != NULL){

        for(i=1; i<=p1->topo->dado; i++){
            if(p1->topo->dado%i==0){
                contador++;
            }
        }
        if(contador==2){
            a = p1->topo->dado;
            retiraDado(p1);
            colocaDado(a, p2);
        }
        else{
            retiraDado(p1);
        }
        contador=0;
    }

    while(p2->topo != NULL){

        b = p2->topo->dado;
        retiraDado(p2);
        colocaDado(b, p1);
    }
    return 0;
}
```

Simples 4

Função encontra_letra( );

Essa função recebe um caracter e uma string e inicia um iterador i de 0 até menor que o tamanho da string (utilizando a função strlen() da biblioteca string.h) com passo 1, se a posição i da string for igual o caracter especificado na chamada da função, é somado um a uma variável contador inicializada como 0. Depois de percorrer a string, a função retorna o valor da variável contador, ou seja, quantas vezes o caracter apareceu na frase.

```c
void encontra_letra(char letra, const char* string){

    int i;
    int contador = 0;

    for(i=0;i<strlen(string);i++){
        if(string[i]==letra){
            contador++;
        }
    }

    printf("%i", contador);
}
```

Intermediário 1

Função angulo( );

Essa função recebe as coordenadas de três pontos, com elas são calculadas as coordenadas de dois vetores que tem o segundo ponto como ponto comum, a norma deles e o produto interno, após isso é possível calcular qual seria o cosseno do ângulo entre esses vetores, esse valor é colocado na função acos( ) da biblioteca math.h que retorna o valor do angulo em radianos, para expressar em graus é feita a multiplicação desse resultado por (180/pi).

```c
void angulo(float x1, float y1, float x2, float y2, float x3, float y3){

    float xvetor1, yvetor1, xvetor2, yvetor2;
    double modBA, modBC;
    double multVet;
    double x;
    double angulo;
    float pi = 3.1415;


    xvetor1 = x1 - x2;
    yvetor1 = y1 - y2;
    modBA = sqrt(pow(xvetor1,2)+pow(yvetor1,2));

    xvetor2 = x3 - x2;
    yvetor2 = y3 - y2;
    modBC = sqrt(pow(xvetor2,2)+pow(yvetor2,2));

    multVet = (xvetor1*xvetor2)+(yvetor1*yvetor2);

    x = (multVet/(modBA*modBC));

    angulo = acos(x)*180/pi;

    printf("%.1f graus", angulo);

}
```

Difícil 1

Esse código depende de estrtruturas de dados e funções de estruturas de dados explicadas anteriormente nessa documentação.

O objetivo desse código é contar quantos “diamantes” (<>) são formados nas n linhas informadas pelo usuário. Iniciamos uma matriz sem definir tamanho, lemos um n correspondente a quantidade de linhas que serão “escavadas” para procurar diamantes e alocamos espaço de n linhas do tipo char para a matriz inicial, onde cada linha pode ter até 1000 caracteres. Um iterador é inicializado como 0 até n com passo 1 e é lido do teclado a linha da matriz, outro iterador j é inicializado como 0 até 1000 com passo 1, se matriz[i][j] for igual a ‘<’ esse valor é colocado em uma pilha, mas caso esse elemento seja ‘>’ e a pilha não esteja vazia, um dado é retirado da pilha e é somado um a uma variavel diamante inicializada como 0, ao final da linha é impresso o valor da variavel diamante e esta volta a ser 0 para ler uma próxima linha, caso haja.

```c
int main(void){

    int i, j;
    int n;
    Pilha *p = comecaPilha();
    int diamante = 0;

    char **matriz;

    scanf("%i", &n);

    matriz = (char**) malloc(n * sizeof (char*));

    for (int i = 0; i <n; ++i){
        matriz[i] = (char*) malloc ( 1000 * sizeof (char));
    }


    for (i=0; i<n;i++){
        setbuf(stdin, NULL);
        fgets(matriz[i], 1000, stdin);
        for (j=0;j<1000;j++){

            if(matriz[i][j]=='<'){

                colocaDado(matriz[i][j], p);
            }
            else if(p->topo!=NULL && matriz[i][j]=='>'){

                retiraDado(p);
                diamante++;
            }
        }
        printf("%i\n", diamante);
        diamante = 0;
    }

    return 0;
}
```

Intermediário 2

Função tiraRepetido( );

Essa função depende de estruturas de dados e funções de estruturas de dados explicadas anteriormente nessa documentação.

Essa função recebe uma pilha p1, enquanto a mesma não for nula, cria um elemento auxiliar que tem o valor do topo da pilha, enquanto o auxiliar não for nulo, copia os dados da p1 para uma pilha p2, depois, enquanto a p2 não é nula, se o dado do topo da p1 for igual ao dado do topo da p2, é somado um a uma variavel contador inicializada como zero e retira esse dado da p2, se não for igual, apenas retira o dado da p2. Quando p2 for nula, se o contador for igual a 1, ou seja, o número só apareceu uma vez, esse dado é retirado da p1 e colocado em uma pilha p3, se o contador for maior que 1, quer dizer que o dado foi repetido e ele apenas é retirado da p1. O contador volta a ser zero para verificar o próximo dado da p1. Quando a p1 estiver nula, os dados da p3, voltam para a p1 e a pilha é impressa na tela sem os números repetidos.

```c
void tiraRepetido(Pilha *p1){

    int contador = 0;
    int a, b, c;
    Pilha *p2 = inicializaPilha();
    Pilha *p3 = inicializaPilha();

    while(p1->topo!=NULL){

        Elemento *aux = p1->topo;

        while(aux!=NULL){
            push(p2, aux->dado);
            aux=aux->proximo;
        }
        while(p2->topo!=NULL){
            if(p1->topo->dado==p2->topo->dado){
                contador++;
                pop(p2);
            }
            else{
                pop(p2);
            }
        }
        if(contador==1){
            a = p1->topo->dado;
            pop(p1);
            push(p3, a);

        }

        else{
            pop(p1);

        }
        contador=0;
    }
    while(p3->topo!=NULL){

        b = p3->topo->dado;
        pop(p3);
        push(p1,b);

    }
    imprimePilha(p1);

}
```