

Algoritmo e Estrutura de Dados

Professor: Leonardo Garcia Tampelini

Respostas da Lista de Exercícios – Assunto: Estruturas de Controle

1)

Observações sobre o exercício:

A resolução dos exercícios será dividida em duas etapas: Projeto do programa e Implementação. Embora a resposta do exercício seja apenas o código-fonte do programa estas etapas servirão para ilustrar como um programa deve ser elaborado passo-a-passo.

a) Dados três números inteiros digitados pelo usuário diga qual é o maior e o menor entre eles

Projeto do programa:

Este programa pode ser dividido nas seguintes etapas

- Ler os três números
- Comparar números lidos e descobrir o maior
- Idem para descobrir o menor
- Imprimir o maior e o menor número

Implementação:

a) Ler os três números

Para ler os 3 números, basta declará-los e utilizar a função `scanf` para efetuar a leitura do teclado

```
#include <stdio.h>

int main(void){
    //declaração e leitura dos números
    int a,b,c;
    scanf("%d %d %d" ,&a,&b,&c);

    return 0;
}
```

b) Comparar números lidos e descobrir o maior

O valor do maior poderia ser obtido ordenando-se os três números, porém isto é uma tarefa difícil. Uma maneira mais simples é inicialmente escolher um número como maior (digamos, a variável `a`), armazená-lo em uma variável e depois comparar com os outros dois sucessivamente. Se em alguma dessas comparações o suposto maior número for menor que o comparado, atualizamos o valor da variável maior.

Para implementar esta etapa precisamos então declarar o variável maior, efetuar as comparações necessárias. Caso alguma comparação seja verdadeira o valor da variável maior deverá ser atualizado com o novo maior valor.

```
#include <stdio.h>

int main(void){
    //declaração e leitura dos números
    int a,b,c;
    //declaração da variável maior
    int maior;
    scanf("%d %d %d" ,&a,&b,&c);
    //escolhemos um possível maior valor
    maior = a;
    //comparamos maior com b
    if( maior < b){
        //suposto maior é menor que b então b é o possível maior
        maior = b;
    }
    //o mesmo raciocínio é válido para c
    if(maior < c){
        maior = c;
    }

    return 0;
}
```

c)Descobrir o menor número

O mesmo raciocínio utilizado para descobrir o maior número pode ser utilizado para descobrir o menor número, só que testamos se o suposto menor número é maior que os números restantes... caso a comparação seja verdadeira atualizamos a variável menor.

```
#include <stdio.h>

int main(void){
    int a,b,c;
    //declaração da variável maior e menor
    int maior,menor;
    scanf("%d %d %d" ,&a,&b,&c);
    maior = a;
    if( maior < b){
        maior = b;
    }
    if(maior < c){
        maior = c;
    }
    //supomos que o menor número é a
    menor = a;
    //comparamos com b para verificar se a afirmação anterior é válida
    if(menor > b){
        //b é menor que o suposto menor, logo b é o menor
        menor = b;
    }
    if(menor > c){
        // c é menor que o suposto menor, logo c é o menor
    }
}
```

```

        menor = c;
    }

    return 0;
}

```

d) Imprimir o menor e o maior número

Uma vez que descobrimos ambos os valores e atribuímos os mesmos as suas respectivas variáveis, basta chamar o comando printf para imprimir o valor das variáveis.

```

#include <stdio.h>

int main(void){
    int a,b,c;
    int maior,menor;
    scanf("%d %d %d", &a,&b,&c);
    maior = a;
    if( maior < b){
        maior = b;
    }
    if(maior < c){
        maior = c;
    }
    menor = a;
    if(menor > b){
        menor = b;
    }
    if(menor > c){
        menor = c;
    }
    //imprimimos o valor do maior e menor valor
    printf("O maior valor e %d e o menor %d",maior,menor);

    return 0;
}

```

b) Dados os número n e m (digitados pelo usuário), escreva a tabuada de n de 1 até m. Restrinja a entrada de dados de maneira que caso seja digitado um valor menor que 2 e maior que 9 seja exibida uma mensagem de aviso e não calcule a tabuada.

Projeto de programa:

Calcular uma tabuada de uma operação (por exemplo: multiplicação) de um número n de 1 até m é o equivalente calcular os seguintes valores:

```

n x 1
n x 2
n x 3
...
n x m

```

Então as tarefas a serem implementadas no programa são:

- Ler n e m
- Calcular o valor de cada elemento da tabuada de 1 até m e imprimí- lo
- restringir a entrada de dados de maneira que a tabuada só seja calculada se n e m estiverem entre 2 e 9 (incluindo 2 e 9).

Implementação:

a) Ler n e m

Para implementar esta etapa basta declarar as variáveis e utilizar o scanf para obter os dados do usuário.

```
#include <stdio.h>

int main(void){
    //declaração e leitura de n e m
    int n,m;
    scanf( "%d %d" ,&n,&m);

    return 0;
}
```

b) Calcular o valor de cada elemento da tabuada de 1 até m e imprimí- lo

Não podemos fazer no código o cálculo de cada expressão direto no código pois não temos um m determinado ! Dada esta situação devemos então utilizar uma estrutura de controle de repetição como while, do-while ou for.

A primeira coisa ser feita é analisar qual das 3 estruturas de controle serão utilizadas.

while é recomendado quando queremos repetir uma operação 0 ou mais vezes e não podemos precisar exatamente quando a repetição termina. Ex: Calculo de logaritmo, conversão de bases

do-while segue o mesmo princípio do while, mas queremos que as operações a serem repetidas sejam executadas ao menos uma vez. Ex: leitura de sequência de numeros até usuário digitar -1, solicitação de senha.

for é mais utilizado quando sabemos exatamente quantas vezes as operações serão repetidas. Ex: fatorial, tabuada

Como podemos precisar que repetiremos m vezes o cálculo dos valores da tabuada, utilizaremos o for.

Uma vez escolhida a estrutura de controle , teremos que ver como adaptá- la ao nosso programa... a cada repetição vemos que o valor do multiplicando vai sendo incrementado e que a repetição para ao chegar ao valor de m. Como o valor do multiplicando **varia** a cada iteração, então devemos declará- lo com uma variável e alterar seu valor a cada rodada. Sabemos também que o multiplicando sempre inicia com valor 1. Com isto podemos implementar o nosso for.

```
#include <stdio.h>
```

```

int main(void){
    int n,m;
    // declaracao do multiplicando e do valor da operação em cada
    etapa
    int mt,valor;
    scanf("%d %d" ,&n,&m);
    /*
    Na estrutura do for temos já a inicialização, teste e atualização
    da variável de controle, no caso a variável "mt"
    */
    for(mt = 1; mt <= m; mt = mt+1){
        //aqui o valor da operação a ser calculada
        valor = n*mt;
        // e é claro nós imprimimos o mesmo
        printf("%d x %d = %d\n",n,mt,valor);
    }
    return 0;
}

```

c) Restringir a entrada de dados de maneira que a tabuada só seja calculada se n e m estiverem entre 2 e 9 (incluindo 2 e 9).

O programa como está não restringe a entrada de dados do usuário. Para tal, teremos que fazer uso de uma estrutura de controle condicional e com ela tomarmos uma decisão de acordo com o valor digitado.

As seguintes decisões precisam ser tomadas pelo programa:

- Se n e m estiverem entre 2 e 9 , ou seja $2 \leq n \leq 9$ e $2 \leq m \leq 9$, o programa deve calcular a tabuada
- Caso contrário, o programa deverá exibir uma mensagem de aviso

Temos então que adaptar essas decisões a uma ou mais estruturas de controle, no caso, apenas um `if` bastará.

Para a construção do IF precisamos determinar a expressão que resulte o valor Verdadeiro caso n e m atendam as condições especificadas acima e Falso caso contrário. A expressão que avalia isso é:

`((n>=2)&&(n <=9)) && ((m >=2) && (m <= 9))`

Esta é uma expressão composta, o primeiro grupamento `((n>=2)&&(n <=9))` possui valor Verdadeiro se n respeitar a definição pedida assim como o segundo possui valor Verdadeiro se m respeita a definição dada. Caso n ou m tenham valor fora do conjunto [2,3,4..9] o && lógico entre os dois grupamentos garante que a expressão terá valor Falso e terá valor verdadeiro apenas se as duas variáveis estiverem dentro da faixa especificada.

Uma vez pronta a expressão podemos ajustar nosso programa para atender a ultima etapa.

```

#include <stdio.h>

int main(void){
    int n,m;
    int mt,valor;
    scanf("%d %d" ,&n,&m);
    //testamos se n e m estão na faixa especificada
    if( ((n>=2) && (n<=9)) && ((m >=2) && (m <=9)) ){
        //n e m estão na faixa especificada !
    }
}

```

```

        for(mt = 1; mt <= m; mt = mt+1){
            valor = n*mt;
            printf("%d x %d = %d\n",n,mt,valor);
        }
    }else{
        // n e m fora da faixa permitida
        printf("Valores fora da faixa permitida !\n");
    }
    return 0;
}

```

c)Dados os números n e m, escreva todos os números entre 1 a 200 que sejam divisíveis por n mas não por m.

Projeto do de programa :

Este programa consiste em listar uma seqüência de números de 1 a 200, só que antes de imprimir um número em particular devemos verificar se o mesmo atende a restrição pedida.

Tarefas a serem implementadas

- Obter o valor de n e m
- Percorrer todos os números de 1 a 200
- Analisar para cada número percorrido se o mesmo pode ser impresso na tela

Implementação:

Ler números é algo bem simples , a partir de agora essa etapa não será mais abordada.

a)Percorrer todos os números de 1 a 200

Percorrer todos os números dentro de um intervalo restrito e bem definido é uma tarefa que pode ser feita facilmente com for ou while. O que precisamos a mais é um contador para regular a quantidade de repetições.

```

#include <stdio.h>

int main(void){
    int n,m;
    //declaração de um contador
    int cont;
    //leitura de n e m;
    scanf("%d %d" ,&n,&m);
    //números são percorridos pelo contador
    for(cont = 1; cont <= 200; cont = cont+1){
        /*
            aqui deve ser tomada a decisão de imprimir um número ou
            não
        */
    }

    return 0;
}

```

b) Analisar para cada número percorrido se o mesmo pode ser impresso na tela

Uma tomada de decisão deve ser feita através de uma estrutura de controle condicional, no caso a mais adequada é o IF. Para poder utilizar o IF devemos ver quantos serão necessário e qual expressão avalia Verdadeiro ou Falso diante da análise pedida.

Temos duas situações:

- Se o número for divisível por n e não por m o número é impresso
- Senão nenhuma atitude é tomada

Sabendo disso é fácil ver que apenas um IF é necessário, sem necessitar da construção de um ELSE pois se a condição de teste falhar nenhum comando é executado. Resta determinar a expressão.

Sabemos que quando um número a é divisível por b então $a \% b$ é sempre 0, qualquer valor distinto indica que a não é divisível por b. Dado isto construímos nossa expressão para determinar de um dado número x é divisível por n mas não por m.

$(x \% n == 0) \ \&\& \ (x \% m != 0)$

Note que esse número a ser avaliado no nosso programa é o próprio contador (variável cont), logo substituímos x por cont.

Nosso programa fica então assim:

```
#include <stdio.h>

int main(void){
    int n,m;
    int cont;
    scanf("%d %d", &n,&m);
    for(cont = 1; cont <= 200; cont = cont+1){
        if( (cont%n == 0) && (cont%m != 0) ){
            //o número é divisível por n mas não por m
            printf("%d\n",cont);
        }
    }

    return 0;
}
```

d) Calcule o logaritmo mais próximo de 10 de um dado número n.

Esta questão está um tanto mal formulada em relação ao que eu pretendia pedir. O enunciado correto deveria ser:

“Calcule x igual ao logaritmo mais próximo de 10 de um dado número n onde $x > n$.”

Esclarecido isso vamos resolver a questão.

Projeto de programa:

A grosso modo um logaritmo de um número pode ser calculado contando- se quantas divisões inteiras podemos fazer até que o resultado da divisão seja 0. Logo nosso programa se resume a:

- Obter o número n
- Dividir n por 10 até que o resultado da divisão inteira seja 0
- Contar quantas divisões foram feitas

Note que o que este programa faz no fundo é contar quantos dígitos possui um número positivo.

Implementação:

Vamos construir o básico do programa:

```
#include <stdio.h>

int main(void){
    int n,x;
    scanf("%d" ,&n);

    printf("O logaritmo maior mais proximo e %d\n",x);
    return 0;
}
```

a)Dividir n por 10 até que o resultado da divisão inteira seja 0

Primeiro passo – definir que estrutura de controle utilizar... desta vez não sabemos até quando iremos dividir n então de cara FOR não é o mais recomendado (embora seja perfeitamente possível utilizá- lo). Temos que decidir então se WHILE ou DO- WHILE é mais adequado... Note que pelo menos uma divisão teremos que fazer , mesmo que o número seja 0. Dado isso DO- WHILE facilitará mais a construção de nosso programa já que ele executa uma operação a ser repetida pelo menos uma vez.

Logicamente a operação a ser repetida é de dividir n por 10. A condição de parada é que o resultado da divisão seja 0, logo a operação de divisão deve ser repetida enquanto o resultado da divisão seja diferente de 0 (note que “while” traduzido significa “enquanto”)

```
#include <stdio.h>

int main(void){
    int n,x;
    //declaramos uma variavel que armazena o resultado da divisão
    int r;
    scanf("%d" ,&n);
    //inicialmente r deve começar com o valor de n
    r = n;
    do{
        /*
        dividimos o resultado da ultima divisão por 10
        e armazenamos o resultado
        */
        r = r/10;
    }/*
```



```

        enquanto r não atingir o valor 0 devemos continuar
        dividindo-o
    */
}while(r != 0);

printf("O logaritmo maior mais proximo e %d\n",x);

return 0;
}

```

b) Contar quantas divisões foram feitas

Esta etapa é bem fácil. Precisamos de um contador, que na verdade é o resultado desejado, representado pela variável x. A cada divisão aumentamos o valor do contador em 1.

```

#include <stdio.h>

int main(void){
    int n,x;
    int r;
    scanf("%d" ,&n);
    //inicializamos o contador com 0 (não foi feita divis
    x = 0;
    r = n;
    do{

        r = r/10;
        //a cada divisão incrementamos o contador
        x = x + 1;
    }while(r != 0);
    //Agora sim há sentido em imprimir o valor de x
    printf("O logaritmo maior mais proximo e %d\n",x);

    return 0;
}

```

e) Dado um dígito, escrever o mesmo em código Morse

Projeto do programa:

Este programa é simples. Dado um dígito específico imprimir o seu código Morse associado. A questão é decidir mesmo qual estrutura de controle condicional se adequa mais a situação.

Implementação:

O programa pode ser feito tanto com um grupo de IFs quanto um SWITCH só. Para fins ilustrativos o programa será feito com ambas estruturas.

Como neste programa temos uma grande quantidade de casos particulares em torno de um mesmo valor, SWITCH é o mais recomendado. Utilizando-o o programa adquire este aspecto:

```
#include <stdio.h>

int main(void){
    int n;
    //declaramos uma variavel que armazena o dígito
    scanf("%d" ,&n);

    switch(n){
        case 0: printf("_ _ _ _");
                break;
        case 1: printf(". _ _ _");
                break;
        case 2: printf(". . _ _");
                break;
        case 3: printf(". . . _");
                break;
        case 4: printf(". . . .");
                break;
        case 5: printf(". . . .");
                break;
        case 6: printf("_ . . .");
                break;
        case 7: printf("_ _ . .");
                break;
        case 8: printf("_ _ _ .");
                break;
        case 9: printf("_ _ _ _ .");
                break;
        default:
            //caso em que n possui mais de um dígito!
            printf("A entrada deve ser composta de 1 dígito !\n");
            break;
    };
    return 0;
}
```

Repare que o scanf de um número não impede que digitemos mais de um dígito nem números negativos... neste caso devemos informar o usuário da situação.

O programa também pode ser feito com IFs e ELSEs encadeados... é mais trabalhoso fazer o mesmo assim, mas a funcionalidade será a mesma. Confira na resposta abaixo:

```
#include <stdio.h>

int main(void){
    int n;
    //declaramos uma variavel que armazena o dígito
    scanf("%d" ,&n);

    if( n == 0){
        printf("_ _ _ _");
    }else if( n == 1){
```

```

        printf(". _ _ _ _");
    }else if( n == 2){
        printf(". . _ _ _");
    }else if( n == 3){
        printf(". . . _ _");
    }else if( n == 4){
        printf(". . . . _");
    }else if( n == 5){
        printf(". . . . .");
    }else if( n == 6){
        printf("_ . . . .");
    }else if( n == 7){
        printf("_ _ . . .");
    }else if( n == 8){
        printf("_ _ _ . .");
    }else if( n == 9){
        printf("_ _ _ _ .");
    }

    }else{
        //caso em que n possui mais de um dígito!
        printf("A entrada deve ser composta de 1 digito !\n");
    }
    return 0;
}

```

f)Dado um número inteiro, escrever o mesmo em código Morse. O código deve ser fornecido na mesma ordem que os caracteres digitados.

Projeto do programa:

Este realmente não é um programa trivial. Temos que extrair dígito a dígito do número, do mais significativo para o menos significativo, e para cada um deles imprimir o código Morse correspondente.

Tarefas do programa

- Obter o número a ser impresso
- Extrair dígito a dígito do número, do mais significativo ao menos significativo
- Imprimir o código correspondente para cada dígito extraído

Destas etapas, a mais difícil de longe é a segunda.

Para facilitar o entendimento e fazer um código mais limpo e fácil de entender utilizaremos funções e procedimentos na construção do programa.

Implementação:

Vamos fazer a parte básica do programa:

```

#include <stdio.h>

int main(void){
    int n;
    scanf("%d", &n);
}

```

```
    return 0;
}
```

a) Extrair dígito a dígito do número, do mais significativo ao menos significativo

Vamos analisar um número qualquer, digamos o número 145.

É fácil ver que se queremos extrair o primeiro dígito por meio de uma conta basta fazer a divisão inteira de 145 por 100. Logo:

$$145/100 = 1$$

Porém queremos também obter o que restou do número (no caso 45) , pois o próximo dígito é necessário para que este seja impresso também. Para fazermos isso basta utilizar o operador %. Observe:

$$145\%100 = 45$$

Vamos repetir o mesmo tratamento para o número 45, mas trabalhando com 10 em vez de 100:

$$45/10 = 4 \text{ (dígito mais significativo)}$$

$$45\%10 = 5 \text{ (dígitos restantes)}$$

Repetindo o caso para 5 temos:

$$5/1 = 5$$

$$5\%1 = 0$$

Ou seja dado um número n qualquer precisamos:

- Descobrir qual é a potência p de 10 mais próxima de n tal que $p \leq n$
- Dividir n por p e assim obter o primeiro dígito
- Calcular o resto da divisão de n por p para obter os dígitos restantes
- repetir o procedimento para os dígitos restantes até que não reste mais nenhum

Note que a potência p de 10 mais próxima de um número n tal que $p \leq n$ é exatamente a quantidade de dígitos de n menos 1 . E que no exercício da letra d foi feito um programa que faz esta conta. Em vez de copiarmos diretamente o código-fonte e inserir no meio do main, vamos fazer isto de maneira mais prática... através de uma **função**.

Uma função é um subprograma que dado um certo conjunto de parâmetros de entrada, o mesmo retorna um valor de acordo com o conjunto de parâmetros. Você pode imaginar isto como se fosse uma função matemática.

Quando colocamos uma função no meio de uma expressão o subprograma

Ex.:

A função sqrt tem como parâmetro de entrada um número real R e é avaliada em uma expressão como a raiz quadrada de R. Veja só a avaliação desta expressão:

$$(4 + \text{sqrt}(81.0))/2.0 \implies \text{sqrt}(81.0) \text{ é avaliada como } 9.0$$

$$(4 + 9.0)/2.0$$

$$13.0/2.0$$

$$7.5$$

Relembrando a declaração de uma função:

```
tipo NomeDaFunção (Listagem de Parâmetros de Entrada) {  
  
    comandos  
  
    return ValorDaFunção ;  
}
```

Ex:

a função abaixo calcula a média aritmética de 3 números reais passados como parâmetro de entrada (declarados como a,b e c), O valor retornado pela função é do tipo float (real, ou ponto flutuante).

```
float media3(float a, float b, float c){  
    return (a + b + c)/3.0 ;  
}
```

Portanto, vamos criar uma função que dado um número inteiro retorna a quantidade de dígitos que ele possui reaproveitando o código do exercício anterior.

```
int QtdDigitos(int n){  
    int x;  
    //inicializamos o contador com 0  
    x = 0;  
    do{  
        n = n/10;  
        //a cada divisão incrementamos o contador  
        x = x + 1;  
    }while(n != 0);  
  
    return x;  
}
```

Quando colocamos QtdDigitos(i) dentro de uma expressão , o subprograma será executado com a variável n inicializada com o valor de i e após seu término será avaliado com o valor de x do tipo inteiro.

Note que essas variáveis n e x declaradas dentro dele só existem dentro do bloco da função (trecho de código entre chaves) e são completamente independentes de quaisquer variáveis declaradas dentro de outros blocos de função ou do main. Inclusive podemos declarar a variável n e x dentro do bloco do main sem quaisquer restrições , isto é o n declarado dentro de uma função não é o mesmo n declarado dentro do bloco do main.

Voltando ao programa...

Já sabemos qual é o expoente da potência de 10, mas precisamos é do valor de 10 elevado a este expoente. Vamos então construir uma função que dado um número positivo e retorna 10^e .

```
int potencia10(int e){  
    int r,cont;
```

```

    r = 1;
    for(cont = 1; cont <= e; cont++){
        // enquanto contador não for igual a e multiplicamos r por 10
        r = r*10;
    }
    return r;
}

```

Agora resta apenas decidir qual é a estrutura de repetição que vamos utilizar. Sabemos que se um número tem t dígitos então vamos repetir a operação t vezes. Podemos utilizar o FOR ou WHILE. Desta vez vamos utilizar o WHILE. O programa então ficará desta maneira

```

#include <stdio.h>

int QtdDigitos(int n){
    int x;
    x = 0;
    do{
        n = n/10;
        x = x + 1;
    }while(n != 0);

    return x;
}

int potencia10(int e){
    int r, cont;
    r = 1;
    for(cont = 1; cont <= e; cont++){
        r = r*10;
    }
    return r;
}

int main(void){
    int n,p,t,d;
    scanf("%d", &n);
    //calculamos o total de dígitos do numero
    t = QtdDigitos(n);
    //enquanto houverem dígitos a serem extraídos repetimos operação
    while(t > 0){
        //calculamos a potencia p
        p = potencia10(t - 1);
        //extraímos o dígito mais significativo
        d = n/p;
        //recuperamos o restante dos dígitos
        n = n%p;
        //diminuímos um dígito do total de dígitos
        t = t - 1;
    }

    return 0;
}

```

b)Imprimir o código correspondente para cada dígito extraído

Esta é a etapa final do programa. Note que já fizemos também um programa que dado um dígito imprime o seu código Morse. Podemos transformar este programa em um subprograma também e adicionarmos ao nosso código.

A questão é que desta vez não faremos uma função, e sim um **procedimento**. Um procedimento é um subprograma que dado seus parâmetros de entrada, executa seu código sem avaliar nenhum valor, ou seja não é possível utilizá-lo no meio de uma expressão e sim como um comando. Um bom exemplo disso é o comando printf, o mesmo não retorna valor algum e seu código imprime na tela uma lista de parâmetros dados.

A declaração de um procedimento é similar a de uma função, só que o tipo do mesmo é sempre **void** (que quer dizer “vazio”), o restante é exatamente igual a uma função.

Ex:

este procedimento imprime n asteriscos na tela dado um n qualquer;

```
void imprimeAsteriscos(int n){
    int contador;
    for(contador = 1; contador <= n; contador++){
        printf("*");
    }
}
```

Da mesma maneira vamos fazer isto com o programa que imprime o código Morse de um dígito.

```
void ImprimeMorse(int n){
    switch(n){
        case 0: printf("_ _ _ _ _");
                break;
        case 1: printf(". _ _ _ _");
                break;
        case 2: printf(". . _ _ _");
                break;
        case 3: printf(". . . _ _");
                break;
        case 4: printf(". . . . _");
                break;
        case 5: printf(". . . . .");
                break;
        case 6: printf("_ . . . .");
                break;
        case 7: printf("_ _ . . .");
                break;
        case 8: printf("_ _ _ . .");
                break;
        case 9: printf("_ _ _ _ .");
                break;
        default:
            //caso em que n possui mais de um dígito!
            printf("A entrada deve ser composta de 1 digito !\n");
            break;
    }
}
```

```
};  
}
```

Se chamarmos `ImprimeMorse(k)` em qualquer trecho do nosso programa, será impresso o código morse do dígito `k`. Uma vez com esse procedimento pronto basta chamar `ImprimeMorse` dentro da repetição principal do programa para cada dígito obtido.

```
#include <stdio.h>  
  
void ImprimeMorse(int n){  
    switch(n){  
        case 0: printf("_ _ _ _ _");  
                break;  
        case 1: printf(". _ _ _ _");  
                break;  
        case 2: printf(". . _ _ _");  
                break;  
        case 3: printf(". . . _ _");  
                break;  
        case 4: printf(". . . . _");  
                break;  
        case 5: printf(". . . . .");  
                break;  
        case 6: printf("_ . . . .");  
                break;  
        case 7: printf("_ _ . . .");  
                break;  
        case 8: printf("_ _ _ . .");  
                break;  
        case 9: printf("_ _ _ _ .");  
                break;  
        default:  
            //caso em que n possui mais de um dígito!  
            printf("A entrada deve ser composta de 1 digito !\n");  
            break;  
    };  
}  
  
int QtdDigitos(int n){  
    int x;  
    x = 0;  
    do{  
        n = n/10;  
        x = x + 1;  
    }while(n != 0);  
  
    return x;  
}  
  
int potencia10(int e){  
    int r, cont;  
    r = 1;  
    for(cont = 1; cont <= e; cont++){  
        r = r*10;  
    }  
    return r;  
}
```



```

}

int main(void){
    int n,p,t,d;
    scanf("%d",&n);
    t = QtdDigitos(n);
    while(t > 0){
        p = potencia10(t - 1);
        d = n/p;
        n = n%p;
        t = t - 1;
        //imprimimos o dígito d extraído em Morse
        ImprimeMorse(d);
        /*
        imprimimos um espaço em branco
        o para evitar que os códigos
        fiquem grudados
        */
        printf(" ");
    }

    return 0;
}

```

Fazer este programa sem utilizar procedimentos e funções é no mínimo , bem mais trabalhoso.

g) Dado um número n, escreva um “X” formado por asteriscos. Permita que apenas números ímpares sejam utilizados como entrada.

Projeto do Programa:

Em vez de pensarmos no “X” como uma figura só , podemos pensar que o programa traça duas diagonais simultaneamente. Desenhar uma diagonal é um problema bem mais simples.

O programa pode ser dividido nas seguintes etapas:

- ler o número
- traçar diagonal principal
- traçar diagonal secundária
- restringir entrada de números pares

Implementação:

Estrutura básica do programa

```

int main(void){
    int n;
    scanf("%d",&n);

    return 0;
}

```

a) Traçar a diagonal principal

Dado um número n, traçar uma diagonal na tela com caracteres é escrever n vezes uma linha composta por n caracteres onde n-1 caracteres brancos e um asterisco. Observe:

Para n = 3

*		
	*	
		*

Para n = 3 escrevemos 3 linhas de 3 caracteres onde o asterisco é posicionado de acordo com a posição da linha. Se for a primeira linha o asterisco entra na primeira posição, se for a segunda, na segunda posição... a regra se repete para todas as linhas. Os caracteres restantes são brancos.

Vendo isso vamos primeiro nos preocupar primeiro em desenhar uma linha. Dada a posição l de uma linha desenhá-la de acordo com a regra acima... isso pode ser feito com uma estrutura de repetição. No caso a mais indicada é o FOR.

```
for(cont = 1; cont <= n; cont ++){
    if(cont == 1){
        printf("*");
    }
    else{
        printf(" ");
    }
}
printf("\n");
```

O que trecho de programa acima faz é escrever uma linha de n caracteres, se a posição do caracter escrito (representado pela variável cont) for igual a posição l da linha escreve-se um asterisco, se não for imprime-se um espaço em branco. De qualquer maneira sempre se escrevem n caracteres.

Resolvemos o problema de escrever uma linha, mas queremos escrever n linhas. Para isso podemos fazer uso de mais uma repetição. Da mesma maneira utilizar FOR é o mais recomendado. Nesta repetição iremos variar o valor de l de 1 até n, colocando o for que desenha a linha dentro dele.

Nosso programa fica assim

```
int main(void){
    int n,l,cont;
    scanf("%d",&n);
    for(l = 1; l <=n;l++){
        //variarmos l de 1 a n
        for(cont = 1; cont <=n; cont ++){
            //escrevemos n caracteres, se for o l-esimo caracter
            //então é um asterisco
            if(cont == l){
                printf("*");
            }
            else{
```

```

        //se for qualquer outro é espaço em branco
        printf(" ");
    }
}
//pulamos a linha após escrever os n caracteres
printf("\n");
}
return 0;
}

```

b) Traçar a diagonal secundária

O problema é praticamente o mesmo de traçar a diagonal principal, só a posição do asterisco cai em uma posição diferente. Se a linha for a primeira o asterisco deverá ser desenhado na n-esima posição, na segunda linha na n-esima posição menos 1 até que na n-esima linha o asterisco deverá ser desenhado na 1ª posição. Seguindo essa lógica, em uma linha de posição l o asterisco deverá ser desenhado na posição $n - l + 1$.

Não iremos fazer outro grupo de FOR para imprimir a diagonal secundária pois em vez de obtermos o X teríamos duas diagonais uma embaixo da outra. Colocar dois agrupamentos de for vai nos dar isso:

*		
	*	
		*
		*
	*	
*		

Queremos isso:

*		*
	*	
*		*

Podemos alterar a expressão do IF da repetição que desenha a linha de maneira que se a posição a ser desenhada for do caracter da diagonal principal **OU** do caracter da diagonal secundaria então imprime-se um asterisco.

Alterando-se apenas a expressão do IF fazemos nosso programa desenharmos um "X".

```

int main(void){
    int n,l,cont;
    scanf("%d",&n);
    for(l = 1; l <=n;l++){
        //variarmos l de 1 a n
        for(cont = 1; cont <=n; cont++){
            /*escrevemos n caracteres,
            Se a posição for igual a l então é diagonal principal
            se for igual a (n - l + 1) é d. secundaria
            */
            if((cont == l ) || (cont == (n - l + 1 ))){
                printf("**");
            }
        }
    }
}

```

```

    }
    else{
        //se for qualquer outro é espaço em branco
        printf(" ");
    }
}
//pulamos a linha após escrever os n caracteres
printf("\n");
}
return 0;
}

```

c)Restringir a entrada de números pares

Basta fazer um if com a expressão $(n \% 2 == 0)$, pois já sabemos que o resto da divisão por 2 de números pares é sempre 0.

```

int main(void){
    int n,l,cont;
    scanf("%d",&n);
    if(n %2 == 0){
        printf("Digite apenas numeros impares!");
    }else{
        for(l = 1; l <=n;l++){
            //variarmos l de 1 a n
            for(cont = 1; cont <=n; cont ++){
                /*escrevemos n caracteres,
                Se a posição for igual a l então é diagonal principal
                se for igual a (n -l + 1) é d. secundaria
                */
                if((cont == l ) || (cont == (n -l +1 ))){
                    printf("*");
                }
                else{
                    //se for qualquer outro é espaço em branco
                    printf(" ");
                }
            }
            //pulamos a linha após escrever os n caracteres
            printf("\n");
        }
    }
    return 0;
}

```

h)Escreva um programa um número imaginado pelo usuário entre 0 e n. Para cada valor sugerido pelo programa como sendo o valor imaginado pelo usuário, o usuário deve responder se o valor sugerido pelo programa é igual, menor ou maior que o valor imaginado. A execução do programa deve terminar assim que o programa “adivinhar” o valor imaginado pelo usuário.

Projeto do Programa:

O problema maior deste programa é : como o programa vai chutar um valor ? E o que fazer quando este valor não for igual ?

Este programa pode ser feito de várias maneiras, uma delas é a partir das respostas do usuário definir qual é um valor máximo e mínimo possível e escolher sempre o valor que está exatamente no meio deles. Se o valor chutado for maior que o número procurado então o novo valor máximo possível é o valor chutado, o mesmo princípio vale para o caso do valor chutado ser menor.

Veja:

N = 1000;
(usuário pensou no número 128)

Pergunta do Sistema	Valor mínimo	Valor máximo	Resposta do Usuário
O valor é 500 ?	0	1000	500 é maior
O valor é 250 ?	0	500	250 é maior
O valor é 125 ?	0	250	125 é menor
O valor é 187 ?	125	250	187 é maior
O valor é 156 ?	125	187	156 é maior
O valor é 140 ?	125	156	140 é maior
O valor é 132 ?	125	140	132 é maior
O valor é 128 ?	125	132	128 é igual

O valor do chute é sempre $(\text{Valor Mínimo} + \text{Valor Máximo})/2$ considerando a divisão inteira. Caso utilizemos números inteiros o programa sempre acerta o número em uma quantidade limitada de tentativas... o mesmo não vale para números reais, mesmo fazendo a divisão real para alguns números a quantidade mínima de chutes é infinita ! Tente reproduzir este método para achar o número 18 com números reais... Logo o programa será obviamente considerando que os números envolvidos são sempre inteiros

Etapas do programa

- Determinar valor máximo do chute inicial (n)
- Determinar um chute inicial
- Obter respostas do usuário até acertar
- Determinar um novo chute baseado no que o usuário disse caso programa erre.

Implementação:

a) Determinar valor máximo do chute

Este passo é simples, basta declarar n e ler a entrada do usuário.

```
#include <stdio.h>

int main(void){
    int n;
    scanf("%d",&n);

    return 0;
}
```

b) Determinar um chute inicial

Como determinamos antes, o chute feito pelo programa é sempre no meio dos valores mínimo e máximo, logo:

```
#include <stdio.h>

int main(void){
    int n, chute;
    int maximo, minimo;

    scanf("%d", &n);
    maximo = n;
    chute = (maximo + minimo)/2;

    return 0;
}
```

c) Obter a resposta do usuário do usuário até acertar

Ler uma resposta do tipo "128 é maior" ou "200 é menor" é desnecessariamente complicado. Vamos definir que o usuário simplesmente digita >, < ou = para dar suas respostas. Como estes são caracteres temos que utilizar uma variável do tipo **char** para efetuar sua leitura.

Outro detalhe é que precisamos repetir esta obtenção de respostas até que o usuário diga que o valor chutado é igual ao que ele pensou, logo precisamos de uma estrutura de repetição. Como no mínimo uma pergunta é feita então escolheremos DO-WHILE.

```
#include <stdio.h>

int main(void){
    int n, chute;
    int maximo, minimo;
    char resp;
    char auxiliar;

    scanf("%d", &n);
    maximo = n;
    minimo = 0;
    //eliminamos o ENTER depois da leitura de N
    scanf("%c", &auxiliar);

    do{
        chute = (maximo + minimo)/2;
        printf("O valor e %d ?", chute);
        scanf("%c", &resp);
        scanf("%c", &auxiliar);
        //enquanto resposta for diferente de = temos que continuar chutando
    }while(resp != '=');

    return 0;
}
```

Se você leu o código deve estar se perguntando por que motivo obscuro tem um scanf de uma variável auxiliar que nem foi citada... o motivo disto é que o usuário ao digitar sua resposta vai escrever invariavelmente o caractere + ENTER. Acontece que ENTER também é um caractere ! Se deixarmos apenas um scanf o ENTER irá ficar na memória do teclado e na próxima rodada do DO-WHILE o scanf irá ler o ENTER provocando o seguinte resultado:

Texto em verde: Saída do programa

Texto em vermelho: Entrada digitada pelo usuário

```
O valor é 500?>ENTER
O valor é 500?O valor é 500?
```

OBS: O valor chutado não mudou pois não alteramos ainda os valores de mínimo e máximo dentro da repetição !

Portanto este scanf extra permite que nos livremos do inconveniente do caractere ENTER. O scanf após a leitura de n também possui a mesma função.

c) Determinar um novo chute baseado no que o usuário disse caso programa erre.

O programa errou seu chute quando o usuário digitou ">" ou "<", neste caso o que devemos fazer é ajustar os valores do máximo ou do mínimo de acordo com a resposta dada. A expressão que avalia o valor do chute não muda.

```
#include <stdio.h>

int main(void){
    int n,chute;
    int maximo, minimo;
    char resp;
    char auxiliar;

    scanf("%d",&n);
    maximo = n;
    minimo = 0;
    scanf("%c",&auxiliar);

    do{
        chute = (maximo + minimo)/2;
        printf("O valor e %d ?",chute);
        scanf("%c",&resp);
        scanf("%c",&auxiliar);
        if(resp == '>'){
            maximo = chute;
        } else if (resp == '<'){
            minimo = chute;
        }
    }while(resp != '=');

    return 0;
}
```

2)

Observações sobre o exercício:

Nenhuma

a)

Erro:

- O teste do for está incorreto. Num fatorial o loop deve rodar n vezes, afinal a fórmula de fatorial é $n! = 1*2*3.. n$. Com a condição $i != n$ serão feitas n- 1 repetições. Outro problema grave é que se for inserido um número menor ou igual a 0 programa entrará em loop infinito, já que i nunca chegara a um valor menor que 1.
- a variável i não foi declarada
- o operação repetida $r = r*1$ sempre resulta 1, não importa quantas vezes seja repetida, e este valor obviamente não é fatorial de n.

Correção:

- Basta trocar o teste $i!=n$ para $i <= n$. Assim nosso loop estará executando corretamente. Isso não resolve o problema do fatorial de números negativos, mas como esta função só está definida para R^+ então não é o caso de se preocupar.
- Declarar um i resolve o problema
- trocar $r = r*1$ por $r = r*i$, desta maneira, como i varia de 1 a n a cada repetição, teremos o fatorial calculado corretamente

b)

Erro:

- Quando um número negativo é lido o mesmo está entrando no cálculo da média.
- O cálculo da média está sendo feito incorretamente. A expressão $total = (total + lido) / contador$ está incorreta em 2 pontos :
1) $V1/1 + V2/2 + V3/3 + ... + Vn/n$ não é a média aritmética.
2) contador é uma variável inteira, logo a divisão efetuada será inteira e portanto incorreta para o cálculo que desejamos.

Correção:

- Antes de somar o valor lido ao total acumulado e incrementar o contador verificar primeiro se ele é um número negativo primeiro. Isto pode ser feito com um IF (lido > 0)
- Substituir a expressão dentro da repetição por $total = total+lido$ e no final do programa em vez de $media = total$ fazer $media = total/(float) contador$;

```
do{
    scanf("%lf",&lido);
    if(lido >= 0){
        contador++;
        total = total + lido;
    }
}while (lido >= 0);
media = total/(float)contador;
```

Note que nossa resolução criou outro problema que antes não ocorria ! Se o primeiro número lido for negativo faremos uma divisão por 0 ao calcular a média. Isso pode ser tratado com outro IF. Veja a solução abaixo:


```

do{
    scanf("%lf",lido);
    if(lido >= 0){
        contador++;
        total = total + lido;
    }
}while (lido >= 0);
if(contador > 0){
    media = total/(float)contador;
}else{
    media = 0;
}

```

c)

Erro:

- R não foi inicializado
- resto não foi declarado
- O trecho de atualização do FOR (no caso resto++) é incorreto, provocando alterações indevidas no resultado final e um possível loop infinito. Este erro muitas vezes ocorre quando se escreve um FOR sem prestar atenção no programa... e foi justamente isso o que ocorreu...

Correção:

- Inicializar R com 0
- declarar resto
- Uma vez que resto é a variável de controle a atualização dele que está dentro das operações repetidas pode ser passado para a estrutura do FOR

```

R = 0;
for( resto = N; resto != 0; resto = resto/M){
    R = R + ((resto%M)*pot);
    pot = pot*10;
}

```