Aula 7 - Mais problemas com inteiros

Já vimos nas aulas anteriores alguns detalhes de operações com inteiros.

a) A divisão é inteira e o resultado é truncado
b) Existe o operador % (resto da divisão)
c) Cuidado com o resto quando o dividendo e/ou o divisor são negativos

Vamos agora a alguns problemas usando valores inteiros.

P21) Dado N ($0 \leq N \leq 9999$ – N tem até quatro dígitos), imprimir cada um dos dígitos de N na ordem unidade, dezena, centena e milhar.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   /* variáveis usadas no programa */
   int N, unidade, dezena, centena, milhar;

   /* leitura de N */
   printf("entre com o valor de N:");
   scanf("%d", &N);

   /* consistência de N */
   if (N < 0) {
      printf("valor de N invalido");
      system("PAUSE");
      return 0;
   }
   if (N > 9999) {
      printf("valor de N invalido");
      system("PAUSE");
      return 0;
   }
   /* calcular os dígitos da unidade, dezena, centena e milhar */
   unidade = N % 10;
   N = N / 10;
   dezena = N % 10;
   N = N / 10;
   centena = N % 10;
   milhar = N / 10;
   printf("unidade:%10d\ndezena:%10d\ncentena:%10d\nmilhar:%10d\n",
unidade, dezena, centena, milhar);
   system("PAUSE");
   return 0;
}
```

Agora, vamos resolver o mesmo problema com $N \geq 0$ qualquer, imprimindo cada um dos dígitos na mesma ordem. Podemos fazer isso, da mesma forma anterior, dividindo N por 10 e pegando o resto da divisão por 10.

P22) Dado $N \geq 0$, imprimir os dígitos de N na ordem unidade, dezena, centena, etc.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   /* variáveis usadas no programa */
   int N, digito;
```

```
   /* leitura de N */
   printf("entre com o valor de N:");
   scanf("%d", &N);

   /* consistência de N */
   /* esperar até que o usuário digite N correto */
   while (N < 0) {
        /* nova leitura de N */
        printf("\nvalor de N invalido\nentre com o valor de N:");
        scanf("%d", &N);
   }
   /* separar os dígitos */
   while (N > 0) {
      digito = N%10;
      printf("\n%d", digito);
      N = N/10;
   }
   system("PAUSE");
   return 0;
}
```

P23) O problema da máquina de sacar dinheiro.
Dado N ≥ 0 (N é um valor em R$) determinar quantas notas de 100, 50, 20, 10, 5 2 e 1 são necessárias para compor N.

P23a) Dados N >0 e d (0<d<9), verificar quantas vezes o dígito d ocorre em N. Basta usar a solução acima para separar cada dígito de N e compará-lo com d.

P24) (Esse só dá para resolver quando falarmos do tipo float ou double). Dado um valor em reais V (com duas casas decimais, os centavos), além das notas de 100, 50, 20, 10, 5 2 e 1, determinar quantas moedas de 0,50 0,25 0,10 0,05 e 0,01 serão necessárias para compor N.
Sugestão: basta multiplicar V por 100 e colocar numa variável inteira.

Cálculo do dígito de redundância

Em casos como, número de contas bancárias, número da identidade, número do CPF, etc., existe o conceito do dígito de redundância para garantir que sempre que este número for digitado, não haja nenhum erro de digitação. Esse dígito de redundância ou de conferência é uma função dos demais de tal forma que se houver algum erro na digitação o programa que recebe esses dados pode acusar e solicitar nova digitação.

Esses códigos não são perfeitos, mas resolvem a maior parte dos casos.

Exemplo: O seu NUSP é constituído por 7 dígitos: d1 d2 d3 d4 d5 d6 d7. Suponha que d7 é o dígito de redundância e que d7 deve ser igual a (d1*9 + d2*7 +d3*5 + d4*3 + d5*2 + d6*1) módulo 10.

P24a) Dado NUSP verifique se segue a regra acima, isto é, tem 7 dígitos e o d7 segue a regra acima.

O CPF possui 9 dígitos mais 2 de redundância. Supondo que os dois últimos dígitos sejam o resultado da seguinte fórmula: (d1*1 + d2*2 + ... + d9*9) módulo 100.

P24b) Dado um CPF verifique se segue a regra acima.

P25) Dados N, M > 0 calcular o Mínimo Múltiplo Comum entre N e M.
Sugestão: usando a definição, o MMC deve ser múltiplo de N e M. Portanto basta verificar qual o menor múltiplo de N (1.N, 2.N, 3.N, etc.) que também seja múltiplo de M.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   /* variáveis usadas no programa */
   int N, M, mmc;

   /* leitura de N */
   printf("entre com o valor de N:");
   scanf("%d", &N);

   /* consistência de N */
   /* esperar até que o usuário digite N correto */
   while (N <= 0) {
        /* nova leitura de N */
        printf("\nvalor de N invalido\nentre com o valor de N:");
        scanf("%d", &N);
   }

   /* leitura de M */
   printf("entre com o valor de M:");
   scanf("%d", &M);

   /* consistência de M */
   /* esperar até que o usuário digite M correto */
   while (M <= 0) {
        /* nova leitura de M */
        printf("\nvalor de M invalido\nentre com o valor de M:");
        scanf("%d", &M);
   }

   /* testar os múltiplos de N */
   mmc = N;
   while (mmc % M != 0) mmc = mmc + N;
   /* mostra o resultado */
   printf("\nMMC entre %d e %d:%d",N, M, mmc);
   system("PAUSE");
   return 0;
}
```

Nesta solução, se N for menor que M faremos algumas comparações desnecessárias.
Para evitar isso podemos escolher o maior e o menor entre N e M e testar somente os múltiplos do maior.

P25a) Dados N, M > 0 calcular o Mínimo Múltiplo Comum entre N e M, com a adaptação acima, ou seja, testar apenas com os múltiplos do maior.

Outra solução mais clássica para este problema é usar a decomposição de N e M em seus fatores primos. Ocorre que para isso seria necessário saber quais são os primos menores que N e M.

P27) Dados N e M > 0 calcular o Máximo Divisor Comum entre N e M.
Vamos usar o algoritmo de Euclides. Exemplo – mdc (30,18):

	1	1	2	
30	18	12	6	
12	6	0		

Divide-se continuamente o primeiro pelo segundo, em seguida este pelo resto da divisão, até que o resto fique zero. O mdc é o último divisor.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   /* variáveis usadas no programa */
   int N, M, dividendo, divisor, resto;

   /* leitura de N */
   printf("entre com o valor de N:");
   scanf("%d", &N);

   /* consistência de N */
   /* esperar até que o usuário digite N correto */
   while (N <= 0) {
        /* nova leitura de N */
        printf("\nvalor de N invalido\nentre com o valor de N:");
        scanf("%d", &N);
   }

   /* leitura de M */
   printf("entre com o valor de M:");
   scanf("%d", &M);

   /* consistência de M */
   /* esperar até que o usuário digite M correto */
   while (M <= 0) {
        /* nova leitura de M */
        printf("\nvalor de M invalido\nentre com o valor de M:");
        scanf("%d", &M);
   }

   /* inicia o processo de divisão sucessiva */
   dividendo = N;
   divisor = M;
   resto = dividendo % divisor;
   /* divisões sucessivas */
   while (resto != 0) {
      dividendo = divisor;
      divisor = resto;
      resto = dividendo % divisor;
   }

   /* mostra o resultado */
   printf("\nMDC entre %d e %d:%d\n", N, M, divisor);
   system("PAUSE");
   return 0;
}
```

Neste caso, será que N tem que ser menor que M?
O que acontece se N for maior que M?

MDC outra solução

Outra solução para este problema é usar a definição de MDC.
Seja N o maior e M o menor. Se não for, basta trocá-los.

O primeiro candidato a MDC é M, pois M pode ser divisor de N.
Os demais candidatos são M-1, M-2, M-3, etc.
Portanto basta ir testando até encontrar um divisor de N. No pior caso que ocorre quando N e M são primos entre si, vamos chegar a 1como MDC.
Este algoritmo é claramente pior que o algoritmo de Euclides.

P27a) Dados N e M > 0 calcular o Máximo Divisor Comum entre N e M, usando o algoritmo acima.

Este problema é melhor resolvido com condições compostas mas dá para resolvê-lo com o que sabemos até agora.

P27b) Dados N >0 verificar se N é palíndrome. Um número é palíndrome quando é o mesmo lido da direita para a esquerda ou da esquerda para a direita. Ou seja, o primeiro algarismo é igual ao último, o segundo igual ao penúltimo e assim por diante. Exemplos:
55, 1331, 2002, 14741, 4658564.