Programação de Computadores - Plano de Ensino

profa. Dra. Fabíola Ribeiro

carga horária semanal: 2 aulas (aula quizenal)

1 Objetivos

Tornar o aluno capaz de elaborar soluções de problemas práticos na forma de algoritmos estruturados e aplicá-los através de uma linguagem de programação.

Desenvolver atividddades lógicas usando pensamento racional e abstrato.

Permitir ao aluno a descoberta do potencial da computação em Engenharia.

2 Conteúdo Programático

1. números reais e inteiros, tipos de dados, variáveis, comandos de atribuição, operadores aritméticos, operadores / e #.
2. implementação dos exercícios em laboratório.
3. estruturas de decisão: if, if... else, switch... case.
4. implementação dos exercícios em laboratório.
5. estruturas de repetição: while, for, do... while.
6. implementação dos exercícios em laboratório.

3 Estratégia

Teoria: demonstração do conteúdo de cada módulo com algoritmos estruturados, diagramas de blocos e fluxogramas.

Laboratório: estruturas discutidas na teoria implementadas em linguagem C++. Será usada a ferramentoa DEV C++ (freeware).

4 Avaliação

Mesmo critério do semestre anterior:

$MS = \frac{NP1 + NP2}{2}$
Se $MS > 7 \rightarrow$ aprovado
senão
{ $MF = \frac{MS + EX}{2}$
se $MF > 5 \rightarrow$ aprovado
senão reprovado
}

Presença mínima 75%.

5 Bibliografia

básica:

FORBELLONE, A. L. ERBESPÄCHER, H. F. Lógica de Programação: a construção de algoritmos e estruturas de dados. 3a ed. Prentice Hall Brasil, 2005.

ASCENCIO, A.F.G. CAMPOS, E. A. V. Fundamentos da programação de computadores - Algoritmos, Pascal e C/C++. Prentice Hall, 2007.

MANZANO, J. A. N. G. OLIVEIRA, J. F. de. Algoritmos: lógica para desenvolvimento de programação de computadores. Érica, 2010.

complementar:

MEDINA, M. FERTIG, C. Algoritmos e programação: teoria e prática. Novatec, 2006.

BORATTI, I. OLIVEIRA, A. Introdução à programação de algoritmos. 3a ed. Visual Books, 2007.

SALVETTI, D. D. BARBOSA, L. B. Algoritmos. Pearson, 2001.

CORMEN, T. H. et al. Algoritmos, teoria e prática, 2a ed. Elsevier, 2002.

6 Introdução

6.1 Definição de algoritmo

"Algoritmo é a descrição de uma sequência de passos que deve ser seguida para a realização de uma tarefa."(ASCENCIO, 1999)

Exemplo: troca de uma lâmpada.

- desligar a chave no quadro de luz
- retirar a lâmpada antiga
- colocar a lâmpada nova
- religar a chave no quadro de luz

6.2 C/C++

C: resultado de um processo evolutivo de linguagens de programação.

C++: extensão da linguagem C. C é um subconjunto dentro do C++. Inclui suporte à programação orientada a objetos (gráfica).

O C é sensível a letras maiúsculas e minúsculas: variável *num* $\neq$ *Num*. Os comandos são sempre em letras minúsculas.

Estrutura básica:

#include<nome da biblioteca>
void main()
{
sequência de comandos do programa
}

Bibliotecas: ex: iostream.h e conio.h $\rightarrow$ comandos de entrada e saída de dados

É sempre útil incluir comentários para facilitar o ente3ndimento do programa, o que pode ser feito da seguinte forma:
/* comentários */
ou
// comentários

O final de linha tem sempre ; senão o compilador interpreta que o comando continua na linha seguinte.

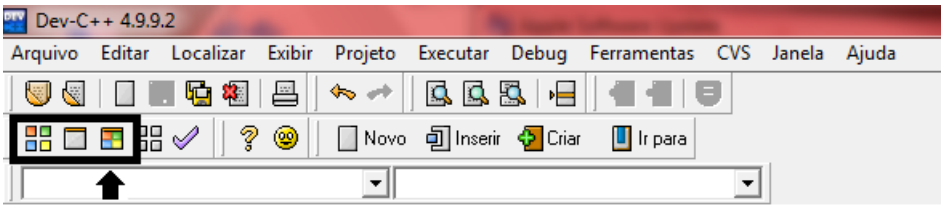
Programação de Computadores (lab) - aula 1

profa. Dra. Fabíola Ribeiro

1 DEV C++

Para transformar um programa qualquer em um arquivo executável, é necessário um compilador. O compilador faz a 'tradução' da linguagem em que programamos para a linguagem da maquina. Neste curso utilizaremos o compilador **DEV C++** da *Bloodshed*, disponível para donwload em http://www.bloodshed.net/dev/devcpp.html. É necessário baixar a versão com GCC (Gnu C Compiler) já que o windows não vem com compilador C instalado.

Para compilar um programa escolhemos o primeiro ícone a esqueda marcado na figura abaixo. Para executar um programa já compilado usamos o segundo ícone. Para executar as duas tarefas em sequência, a compilação e a execução, usamos o terceiro ícone da seleção.



2 Começando um programa em C/C++

Os programas em C tem a extensão *.C* enquanto os programas em C++ têm a extensão *.CPP* (de C-*plus-plus*).

A estrutura básica de um programa em C/C++ é:

#include<nome da biblioteca>
void main()
{
sequência de comandos do programa
}

Com *#include<nome da biblioteca>* listamos as bibliotecas necessárias para o programa, que contém os comandos que utilizaremos. Todas as inclusões de bibliotecas e de constantes são feitas logo

no início do programa.

A linha *void main()* ou simplesmente *main()* marca o início do programa em si, que deve estar contido entre as chaves {} que delimitam onde ele começa e onde ele termina.

É importante lembrar que as linhas do programa sempre terminam com **;**. É sempre útil incluir comentários dizendo o que é feito em cada etapa do programa. Os comentários podem ser feitos de duas formas:
// comentário
ou
/* comentário */

2.1 Meu primeiro programa em C/C++

Vamos fazer um programa simples, que dá como saída uma frase.

```
/* Hello World */
/* 06ago2011   */

#include<iostream> // incluo as bibliotecas de entrada e saida
#include<conio.h>
using namespace std; // uso namespace padrao
main()
{
      // escrevo a frase desejada
      cout << "\nmeu primeiro programa em C++\n";
      // espero por um enter para fechar o programa
      getch();
}
```

No início do programa incluímos as bibliotecas para entrada e saída de dados com *#include <iostream>* e *#include <conio.h>*. Com o *main()* iniciamos o nosso programa, que fica entre as chaves que seguem. Aqui temos um exemplo de como fazer uma saída de dados em C++, usando o comando *cout*, que recebe a frase meu primeiro programa em C, esta frase é passada sempre entre aspas para não ser confundida com variáveis. O caractere $\backslash n$ colocado junto com a frase faz com que seja pulada uma linha.
A linha *using namespace std* é incluida para que o programa não interprete o comando *cout* como uma variável e retorne a mensagem de erro dizendo que esta variável *cout* não está definida.
Usamos no final do programa o comando *getch()* para que o programa espere que uma tecla seja pressionada antes de fechar a janela de execução, senão não teríamos tempo de ler a mensagem de saída na tela.

O comando *cout* não é a unica forma de gerar uma saída na tela. temos também o comando *printf*. Se utilizarmos o comando *printf* não precisaremos mais da biblioteca *conio.h*. A linha que direciona a frase para saída na tela, usando o comando *printf* fica:

printf("\nMeu primeiro programa em C++\n");

Exercício: Faça um programa que escreva na tela do computador seu nome e seu RA.

2.2 Agora usando números

Vamos escrever um programa que, dado dois números, retorna a soma destes números.

```
/* Calcula a soma de dois numeros */
/* 06ago2011                      */

#include <iostream>
#include <conio.h>
using namespace std; //espaco de nomes padrao
main()
{
      float n1, n2, soma;
      // leio os dois numeros
      cout << "Entre com dois numeros separados por enter\n";
      cin >> n1;
      cin >> n2;
      // calculo a soma
      soma = n1 + n2;
      // imprimo o resultado
      cout << "\nsoma = " << soma;
      // espero por um enter para fechar a janela
      getch();
}
```

Na linha *float n1, n2, soma;* definimos as variáveis que usaremos no programa. Para definir variáveis em C/C++ precisamos primeiro indicar seu tipo (aqui é *float*), seguido do nome da variável, ou variáveis separadas por virgula. Na tabela seguite temos alguns tipos de variáveis.

tipo		faixa de valores
int	número inteiro	-32767 a 32767
long int	número inteiro	-2147483647 a 2147483647
float	número real	$3{,}4.10^{-38}$ a $3{,}4.10^{38}$
double	número real	$1{,}7.10^{-308}$ a $1{,}7.10^{308}$
long double	número real	$3{,}4.10^{-4934}$ a $3{,}4.10^{4932}$
char	alfanumérico	-127 a 127

(Ascencio e Campos, 2007. Com adaptações.)

Se estamos trabalhando apenas com números inteiros (sem casas decimais, podemos trabalhar com variáveis do tipo *int*, se precisamos de casas decimais, optamos por tipo *long*. Se é necessária uma precisão maior nos cálculos, passamos a utilizar os tipos *float*, *double* ou *long double*. Se precisamos trabalhar com um nome ou outro dado alfanumérico, o tipo deve ser *char*.

Nas linhas *cin >> n1;* e *cin >> n2;* é feita a entrada dos dois números para o cálculo da soma, o primeiro número é armazenado na variável *n1* e o segundo em *n2*. Esta entrada de dados também poderia ser feita pelo comando *scanf*:
scanf("%f",&n1);
scanf("%f",&n2);

Aqui *%f* indica o tipo de variável que queremos ler (f=*float*), entre aspas, seguido pela variável que armazenará o valor.

Na linha *soma = n1 + n2;* é calculada a soma dos valores armazenados nas variáveis *n1* e *n2*, o resultado então é atribuído à variável soma. Esta atribuição é representada pelo comando =.

A saída do resultado na tela é feita no programa usando o comando *cout*, mas também pode ser feita usando-se o comando *printf*:
printf("\nsoma = %f\n",soma);

No comando, *%f* indica onde o valor da variável soma (especificada no comando printf após a vírgula) deve aparecer na frase de saída, que é especificada na primeira parte do comando:

printf("frase",variáveis na ordem que aparecem na frase);
Usamos *%f* pois trabalhamos com variável tipo *float*. Se fosse *int* usaríamos *%i*, se fosse *long* usaríamos *%ld*, se fosse *double* usaríamos *%d* e *char* usaríamos *%c*.

Se quisermos formatar a saída, usamos o *printf* da seguinte forma: *printf("5.2f",soma);*, o que produzirá uma saída tipo *float* (*printf("5.2**f**",soma*);), com 5 casas no máximo para a parte inteira do número (*printf("**5**.2f",soma);*), e duas casas depois da vírgula (*printf("5.**2**f",soma);*). Para modificar isto para uma saída com quatro casas decimais, bastaria usar *printf("5.4f",soma);*.

Exercício: Escrever um programa que calcule a diferença entre dois números inteiros.

Exercício: Escrever um programa que calcule a divisão de dois números e que dê o resultado com uma casa decimal.

Programação de Computadores (lab) - aula 2

profa. Dra. Fabíola Ribeiro

1 Operadores aritméticos

Para efeturar cálculos em programas em C/C++ podemos utilizar os seguintes operadores aritméticos.

operador	nome	exemplo
+	soma	x1 + x2
-	subtração	x1 - x2
*	multiplicação	x1 * x2
/	divisão	x1 / x2
%	módulo*	x1 % x2
++	incremento**	x1++
−−	decremento***	x1−−

* O módulo é o resto da divisão de dois números inteiros, e é também um número inteiro. Exemplo: 7 % 3 = 1.
** Aumenta um inteiro de uma unidade. O mesmo que x1 = x1+1.
*** Diminui um inteiro de uma unidade. O mesmo que x1 = x1 - 1.

Quando comparamos o conteúdo de duas variáveis, precisamos das seguintes relações:

operador	nome	exemplo
==	igual a	x1 == x2
!=	diferente de	x1 != x2
>	maior que	x1 > x2
<	menor que	x1 < x2
>=	maior-igual	x1 >= x2
<=	menor-igual	x1 <= x2

Em alguns cálculos precisamos de funções matemáticas, tais como seno, cosseno ou raiz quadrada. Algumas destas funções estão listadas a seguir. Para utilizar estas funções precisamos incluir no código a biblioteca *math.h* com *#include <math.h>*.

função	nome	exemplo
pow(,)	potenciação	2^4 = pow(2,4)
sqrt	raíz quadrada	$\sqrt{2}$ = sqrt(2)
sen	seno(*)	sen(x1)
cos	cosseno(*)	cos(x1)
tan	tangente(*)	tan(x1)
exp	exponencial de base e	exp(x1)
log	logaritmo de base e	log(x1)
log10	logaritmo de base 10	log10(x1)

(*)Ângulo em radianos

Nas funções trigonométricas os ângulos devem ser dados em radianos. A conversão de um ângulo de graus para radianos é simples, basta multiplicar o ângulo pelo fator $\frac{\pi}{180}$.

2 Aplicações

2.1 Quadrado de um número

No programa a seguir é calculado o quadrado de um número inteiro dado.

```
/* Calcula o quadrado de um dado numero inteiro */
/* 06ago2011                                    */

#include<iostream>
#include<conio.h>
using namespace std; //uso conjunto de nomes padrao

main()
{     int numero, quadrado;
      cout << "\n entre com um numero\n";
      cin >> numero;
      // faco o calculo do quadrado deste numero
      quadrado = numero*numero; // ou pow(numero,2) com #include<math.h>
      // imprimo resultado na tela
      cout << "\no quadrado de " << numero << " eh " << quadrado;
      // espero por um enter para fechar a janela
      getch();
}
```

O cálculo do quadrado pode ser feito multiplicando o número por ele mesmo (*quadrado = numero*numero;)* ou ainda usando a função que calcula potências (*pow(numero,2);*). Neste último caso precisamos incluir a biblioteca *math.h* no programa *#include <math.h>*.

Exercício: Faça um programa que calcule a raiz cúbica de um dado número.

Exercício: Faça um programa que calcule 2^n, onde n é um número inteiro dado.

2.2 Volume de um cilindro

No programa a seguir calculamos o volume de um cilindo, dados o raio de sua base e a altura. O volume do cilindo é o produto da área da base pela altura:

$$V = A_{base}.h = (\pi.R^2).h \tag{1}$$

```
/* calculo do volume de um cilindro */
/* 06ago2011                        */

#include <iostream>
#include <conio.h>
#include <math.h>

#define PI 3.1415; // defino o valor de PI como uma constante

using namespace std; // conjunto de nomes padrao

main()
{     float raio, altura, volume;
      // entrada dos dados
      cout << "entre com o raio da base do cilindro. ";
      //cin >> raio;
      scanf("%f", &raio); // com scanf não preciso do #include  <conio.h>
      cout << "\nentre com a altura do cilindro. ";
      cin >> altura;

      // calculo o volume do cilindro V = Abase*h
      volume = raio * raio * altura*PI;

      // imprimo na tela o resultado
      //cout << "\n\nvolume = " << volume;
      printf("\n\nvolume = %5.2f\n",volume); // 5 casas antes da virgula e
                                             // 2 casas depois da virgula
                                             // f de float (precisao da variavel)
      // espero por um enter para fechar a janela
      getch();
}
```

Exercício: Faça um programa que calcule o volume V de uma esfera de raio R dado. Para a esfera, $V = \frac{4}{3}\pi.R^3$.

Exercício: Faça um programa que calcule a área lateral A de um cilindro de base de raio R e altura h. (Imagine que desmontamos o cilindro, cortando a lateral de cima a baixo e desenrolando. Temos então um retângulo de altura h e base igual ao perímetro do círculo da base do cilindro, ou seja, $2\pi.R$. A área então será $A = (h).(2\pi.R)$.)

2.3 Resto de uma divisão

Quando queremos calcular o resto da divisão de um número inteiro por outro, devemos usar o operador artmético %. Por exemplo, o resto da divisão de 7 por 3 é 1, ou seja, $7\%3 = 1$. Note que o operador % só se aplica a números inteiros, já que não podemos falar de resto de divisão entre números decimais. O código a seguir calcula o resto da divisão de dois números inteiros. Utilizamos o comando scanf para entrada de dados e printf para a saída.
scanf("%i",&n1);→ scanf("tipo",&variável);
printf("\n O resto da divisao de %i por %i = %i\n",n1,n2,resto); → printf("frase com tipo das variáveis", variáveis na ordem que aparecem na frase);

```
/* Resto da divisão de dois numeros inteiros */
/* 16 ago 2011                               */

#include<stdio.h>
#include<math.h>
#include<conio.h>
main()
{     int n1, n2, resto;

      printf("\n Entre com dois numeros\n");
      scanf("%i",&n1);
      scanf("%i",&n2);

      resto = n1%n2;

      printf("\n O resto da divisao de %i por %i = %i\n",n1,n2,resto);
      getch();
}
```

2.4 Seno, cosseno e tangente de um ângulo

O ponto fundamental do cálculo de funções trigonométricas em C é lembrar que os ângulos devem sempre ser dados em radianos, e não em graus. Para converter um ângulo de graus para radianos, basta multiplicar por $\frac{\pi}{180}$. O código abaixo calcula o seno, cosseno e a tangente de um ângulo dado em graus. As entradas e saídas estão escritas tanto com cin/cout e scanf/printf, basta tirar o comentário da que se deseja usar e comentar a outra opção.

```
// SENO E COSSENO DE UM ANGULO EM GRAUS
// 16 ago 2011
//
// (as funcoes trigonometricas em C/C++ usam entrada em radianos)
//
// Estamos trabalhando com dois modos de entrada/saida, quando quiser alterar
// excluir o comentario de um e comentar o outro.

#include<stdio.h>
#include<iostream>
#include<conio.h>
#include<math.h>

#define pi 3.14159265;

using namespace std;

main()
{
      float angulo, seno, cosseno, tangente;

      // faco a entrada do angulo em graus
      cout << "\n Entre com o angulo em graus: ";
      //cin >> numero;      //usando cin
      scanf("%f",&angulo);  // usando scanf

      // faco a conversao do angulo de graus para radianos
      angulo = angulo/180*pi;

      // calculo as funcoes trigonometricas
      seno = sin(angulo);
      cosseno = cos(angulo);
      tangente = tan(angulo);

      // passo para a saida os resultados
      //cout << "\n cosseno = " << cosseno;
      //cout << "\n seno = " << seno;
      //cout << "\n tangente = " << tangente;
      printf("\n cosseno = %5.2f", cosseno);
      printf("\n seno = %5.2f", seno);
      printf("\n tangente = %5.2f", tangente);
      // saida do printf com 5 casas antes da virgula e duas depois - %5.2f

      // aguardo pelo pressionamento de uma tecla para fechar a janela
      getch();
}
```

Exercício: Faça um programa que dê o seno e cosseno dos ângulos 0^o, 30^o, 45^o, 60^o e 90^o.

Exercício: Faça um programa que calcule o angulo cujo seno é igual a um dado valor. Lembre de dar esse valor entre -1 e +1, pois o seno está sempre limitado entre esses valores.

Programação de Computadores (lab) - aula 3

profa. Dra. Fabíola Ribeiro

1 Condicionais - *if* e *if...else*

Até agora desenvolvemos programas que eram executados de forma linear, linha por linha, incluindo todas as linhas, do início até o final do programa. Podemos alterar este comportamento usando condicionais.

Um exemplo de condicional:

Se (o farol de pedestres está aberto para você)
{atravesse a rua}
senão
{espere o farol de pedestres abrir}

Em C/C++, um condicional simples assume a forma:

```
if condição
{       comando1;
        comando2;
        comando3;
}
```

onde os comandos 1, 2 e 3 só serão executados se a condição ao lado do *if* for satisfeita, senão o programa passa direto por esta parte. Note que se tivermos mais de um comando, estes devem estar delimitados por parênteses {} desde o começo do *if* até o final. O *if* pode ser entendido como a palavra *SE* em português.

O condicional também pode incluir comandos a serem executados caso a condição do *if* não seja satisfeita. Este caso é chamado de condicional composto.

```
if condição
{       comando1;
        comando2;
} else
        {       comando3;
        }       comando4;
        }
```

Neste caso, se a condição for satisfeita, serão executados os comandos1 e comando2, relacionados logo após o *if*, entre parênteses. Se a condição não for satisfeita, serão executados os comandos3 e comando4, relacionados após o *else*. Aqui o *else* pode ser entendido como o *SENÃO* em português.

Temos os seguintes operadores lógicos para usar em programação em C/C++:

nome	operador
E	&&
OU	||
NAO	!
IGUAL	==
DIFERENTE	!=
MAIOR	>
MAIOR-IGUAL	>=
MENOR	<
MENOR-IGUAL	>=

2 Aplicações

2.1 Classificar se um número é positivo ou negativo

O seguinte código classifica um número n como positivo ($n > 0$) ou negativo ($n < 0$).

```c
/* Classifica um numero como positivo ou negativo */
/* 20/08/2011                                     */

#include <stdio.h>
#include <conio.h>

main()
{     float numero;

      /* faco a entrada de dados */
      printf("Entre com um numero: ");
      scanf("%f",&numero);

      /* se o numero eh maior-igual a zero*/
      if(numero > 0)
                printf("\n\n O numero %5.3f eh positivo \n",numero);
      if(numero < 0)
                printf("\n\n O numero %5.3f eh negativo \n",numero);
      if(numero == 0)
                printf("\n\n O numero %5.3f eh nulo \n", numero);
      /* Uso o numero com 3 casas decimais na saida */

      /* aguardo at que uma tecla seja pressionada para fechar a tela */
      getch();
}
```

2.2 Par ou ímpar

O programa abaixo diz se um número n dado é par ou ímpar. Um número é par se ele for da forma

$$n = 2.m, m = 1, 2, 3...$$

ou seja, a divisão $n/2$ deve ser exata, retornando resto zero, $n\%2 = 0$. Um número é ímpar se ele for da forma

$$n = 2.m + 1, m = 1, 2, 3...$$

ou seja, a divisão $n/2$ deve ter resto igual a 1, $n\%2 = 1$. Como se um número não é par, ele é obrigatóriamente ímpar, ou vice-versa, basta testar uma das hipóteses.

```c
// diz se um numero inteiro eh par ou impar
// 20 ago 2011

#include <stdio.h>
#include <conio.h>

main()
{     int numero;

      // faco a entrada de dados
      printf("Entre com um numero inteiro: ");
      scanf("%i",&numero);

      // testo se numero eh par
      if (numero%2 == 0)
         printf("\n\n %i e' par", numero);
         else // se nao eh par, numero tem que ser impar
              printf("\n\n %i e' impar", numero);

      // espero por um caracter para encerrar o programa
      getch();
}
```

Exercício: Faça um programa que diga se um número é múltiplo de 3.

2.3 Aprovado ou reprovado

Vamos supor que um aluno faça duas provas, p1 e p2, e que a média aritmética dessas provas deva ser maior que 7 para que o aluno seja aprovado na matéria. O programa a seguir dá a média e a situação do aluno, se aprovado ou reprovado, dadas as notas das duas provas.

```
/* situacao de um aluno - aprovado ou reprovado */
/* 20 ago 2011                                  */

#include <stdio.h>
#include <conio.h>

main()
{     long p1, p2, media;

      /* faco a entrada de dados */
      printf("\nentre com a nota da p1: ");
      scanf("%d",&p1);
      printf("\nentre com a nota da p2: ");
      scanf("%d",&p2);

      /* calculo a media das notas */
      media = (p1 + p2)/2;

      /* dou a situacao do aluno (media > 5 - aprovado)*/
      if (media >=5)          // se tirar 5 o aluno passa
      {  printf("\n Aluno aprovado - media = %d", media);
      } else
        {   printf("\n Aluno reprovado - media = %d", media);
        }

      /* espero por uma tecla para fechar a tela */
      getch();
}
```

Exercício: De a situação de um aluno (aprovado ou reprovado) em uma matéria. São feitas duas provas, e na média a primeira prova tem peso 1 e a segunda peso 2. O aluno precisa tirar média maior ou igual a 5 para ser aprovado.

Exercício: A média é calculada como $\frac{p1 + p2}{2}$. Você precisa tirar média 7 para ser aprovado, sem precisar fazer exame. Faça um programa que, dada a nota na p1, diga quanto vc precisa tirar na p2 para não ficar de exame. Lembre que a nota máxima na prova é 10, se o valor da p2 for maior que 10, o programa deve retornar uma mensagem.

2.4 Logaritmo de um número

O logaritmo de um número só pode ser calculado se este número for maior que zero. O programa a seguir calcula o logaritmo de um número, dando uma mensagem de erro se este for menor ou igual a zero, fazendo novamente a entrada de dados.

```
// logaritmo de um numero (log de base 10)
// 20 ago 2011

// O logaritmo pode ser calculado apenas para numeros maiores ou
// iguais a zero, entao temos que testar isto no inicio do codigo.

# include <stdio.h>
# include <conio.h>
# include <math.h>

main()
{     float numero;

      // entrada de dados
      printf("Entre com um numero maior ou igual a zero: ");
      scanf("%f",&numero);

      // se o numero for maior que zero calculo o log
      // o calculo do log é feito dentro do printf em 'log10(numero)'
      if (numero > 0)
         printf("\n\n log(%5.2f)=%7.4f",numero,log10(numero));
         else
         {   printf("\nO numero deve ser maior ou igual a zero, tente novamente: ");
             scanf("%f",&numero);
             printf("\n\n log(%5.2f)=%7.4f",numero,log10(numero));
         }

         // espero por um caracter para encerrar
         getch();
}
```

Exercício: Faca um código que dê o arcoseno de um número (o ângulo cujo seno é n). Lembre-se que o seno está restrito entre valores -1 e +1, fora dessa faixa, o programa deve dar uma mensagem de erro explicando isso.

Programação de Computadores (lab) - aula 4

profa. Dra. Fabíola Ribeiro

1 Condicionais - *switch... case*

Uma outra forma de programar um condicional é usando o *switch... case*. Este comando tem a seguinte estrutura. Com o `switch(variável)` identificamos a variável que determinará a condição, a seguir se o `case(condição)` for obedecido, executamos a parte seguinte do programa, parte que é encerrada pelo comando `break` (traduzindo: interrompa). Podemos ter uma série de *cases* com diversas condições. Se nenhuma for satisfeita, é executada a operação *default* logo abaixo dos *cases*.
O *switch* não permite o uso de operadores lógicos no *case*, apenas a verificação se a variável é de um determinado valor.

A estrutura do *switch... case* tem a seguinte forma:

```
switch(variável)
{  case valor1:
        execute aqui;
        break;
   case valor2:
        execute esse outro;
        break;
   default: execute este último;
}
```

2 Aplicações

2.1 Classificar se um número é par ou ímpar

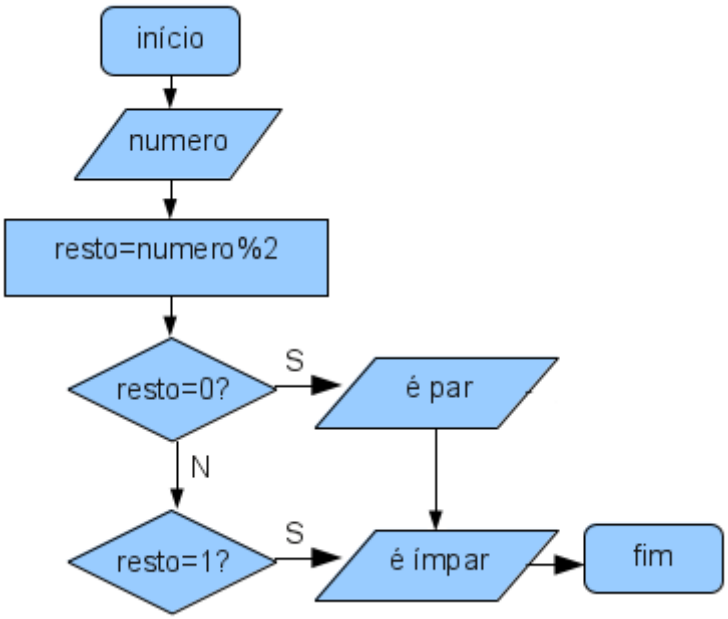
Um número x é par se ele puder ser escrito da forma $x = 2.n$, $n = 0, 1, 2, 3....$, ou seja, ele é múltiplo de dois. Portanto, o resto da divisão deste número por 2 deve ser zero: $x\%2 = 0$.

Um número x é ímpar se ele puder ser escrito da forma $x = 2.n + 1$, $n = 0, 1, 2, 3....$, ou seja, ele não é múltiplo de dois. O resto da divisão deste número por 2 deve ser um: $x\%2 = 1$.

O código a seguir usa o comando *switch... case* para verificar se um dado número é par ou ímpar:

```
// Determina se um numero eh par ou impar
// usando switch...case

#include <iostream>
#include <conio.h>
using namespace std;

main()
{
      int numero,resto;

      // Entrada de dados
      cout << "Entre com um numero inteiro: ";
      cin >> numero;

      // Calculo o resto da divisao de numero por 2
      resto = numero%2;

      // condicional usando switch... case
      switch(resto)
      {             case 0:     // caso resto = 0
                        cout << "\n" << numero << " eh par";
                        break;
                    case 1:     // caso resto = 1
                        cout << "\n" << numero << " eh impar";
                        break;
      }

      getch();    // espero por uma tecla para fechar a tela
}
```

Em fluxograma:



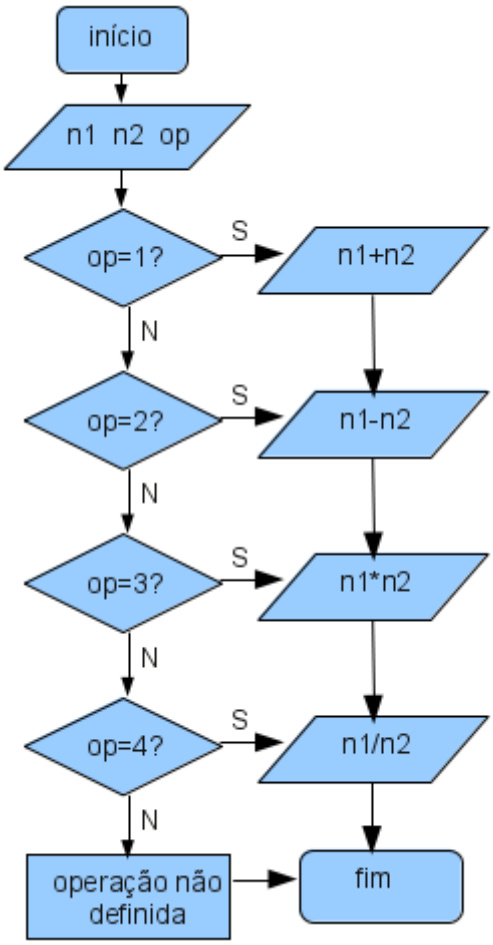
Exercício: Escreva um código para verificar se um dado número inteiro é múltiplo de 7.

Calculadora simples

Vamos fazer um código que, dados dois números, executa a operação desejada com estes números. Vamos adotar as opções (1) adição, (2) subtração, (3) multiplicação e (4) divisão.

```c
/* Calculadora simples */

#include <stdio.h>
#include <conio.h>

main()
{     float n1,n2,resultado;
      int op;

      /* entrada de dados */
      printf("Entre com dois numeros: ");
      scanf("%f",&n1);
      scanf("%f",&n2);

      /* escolha da operacao */
      printf("\n\n Escolha a operacao: ");
      printf("\n(1) adicao \n(2) subtracao \n(3) multiplicacao \n(4) divisao");
      printf("\n operacao:");
      scanf("%i",&op);

      /* condicional com switch... case */
      switch(op)
      {     case 1:     // soma
                  printf("\n\n %5.2f + %5.2f = %5.2f", n1, n2, n1+n2);
                  break;
            case 2:     // subtracao
                  printf("\n\n %5.2f - %5.2f = %5.2f", n1, n2, n1-n2);
                  break;
            case 3:     // multiplicacao
                  printf("\n\n %5.2f x %5.2f = %5.2f", n1, n2, n1*n2);
                  break;
            case 4:     // divisao
                  printf("\n\n %5.2f / %5.2f = %5.2f", n1, n2, n1/n2);
                  break;
            default: printf("\n\n opcao nao definida");
            // se a opcao eh diferente das 4 definidas, dou uma mensagem
            // de erro
      }
      getch(); // espero por uma tecla para fechar a janela
}
```

Em fluxograma:



Exercício: Faça uma calculadora trigonométrica, que, dado um ângulo em graus, o usuário escolha entre as opções (1) seno, (2) cosseno e (3) tangente. Lembre-se que para o cálculo o ângulo deve ser convertido em radianos.

Programação de Computadores (lab) - aula 5

profa. Dra. Fabíola Ribeiro

1 Estruturas de repetição - *while* e *do... while*

Até agora estudamos apenas condicionais, onde uma dada ação era executada se uma condição fosse satisfeita. Em programação, frequentemente precisamos repetir um cálculo diversas vezes, então utilizamos as estruturas de repetição. Aqui o *while* pode ser entendido como *enquanto*, já o *do... while* como *faça... enquanto*.

Vamos tomar o caso onde desejamos elevar um número até a quinta potência, ou seja, o número deve ser multiplicado por ele mesmo 5 vezes. Usando as estruturas de repetição acima, temos:

```
n=1;
while(n=!5)
{ resultado=numero*numero;
  n=n++;
}
```

Na estrutura acima, o teste do while é feito antes de executarmos qualquer ação dentro das chaves {}. Enquanto a condição for satisfeita, os comandos dentro do while são executados. É importante fazer o incremento da variável que usamos como contador das repetições (`n=n++` ou *n=n+1*). Os seguintes passos são executados, supondo que o numero dado é 2. Enquanto n for diferente de 5, repetimos o looping.

- **1a repetição:**n=1, numero = 2*2 = 4, n=n+1=2
- **2a repetição:**n=2, numero = 4*2 = 8, n=n+1=3
- **3a repetição:**n=3, numero = 8*2 = 16, n=n+1=4
- **4a repetição:**n=4, numero = 16*2 = 32, n=n+1=5 **saio do looping**

De fato, $2^5 = 32$. Temos que tomar cuidado com a condição dentro do *while*, se tivéssemos escolhido `while(n<=5)` teríamos mais uma execução do looping, retornando o resultado errado.

Outra forma de fazer a repetição é usando o *do... while*.

```
n=1;
do
{ resultado=numero*numero;
  n=n++;
} while (n<5);
```

Na estrutura acima, executamos o looping uma primeira vez (`do`), e fazemos o teste apenas no final dessa primeira execução. Os passos da execução são os mesmos do caso anterior, mas agora saio do looping pois na quarta repetição, já tenho n=5, e o comando executa o looping apenas enquanto n<5 (`while (n<5)`).

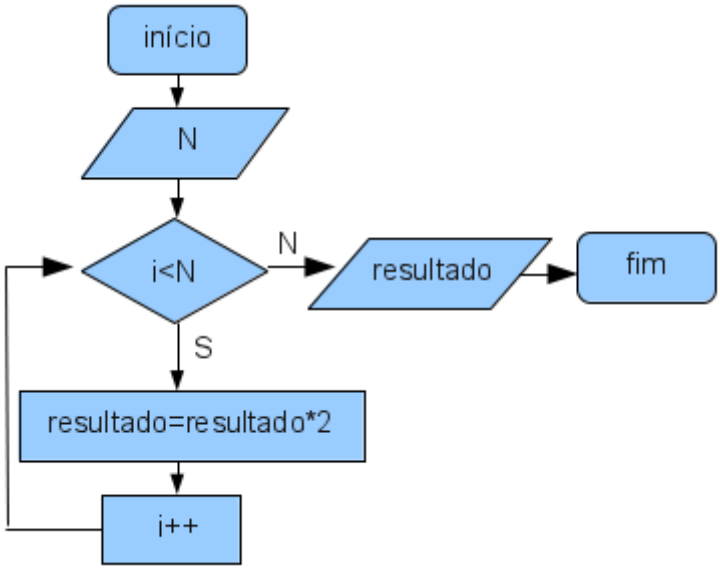
2 Aplicações

2.1 Calcular a enésima potência de 2.

O código a seguir calcula 2^n, para n dado, usando repetições.

```c
/* Calcula dois elevado a potencia N */

#include <stdio.h>
#include <conio.h>

main()
{       int N;
        float resultado;
        int i; //contador

        i=0; //inicio o contador em zero

        printf("entre com a potência N: ");
        scanf("%i",&N);

        // caso nao entre no looping, N=0, o resultado de 2^0=1
        resultado=1;

        while (i<N)
        {       resultado=resultado*2;
                i=i++; //incremento o contador
        }

        printf("\n\n2^%i=%.0f\n\n",N,resultado);

        getch();
}
```

Em fluxograma:



Exercício: Escreva um código para calcular 3^n usando repetições.

2.2 Calcula o logaritmo de um numero.

O código a seguir calcula o logaritmo na base 10 de um dado número. É importante lembrar que o logaritmo pode ser calculado apenas a partir de números maiores que zero. Enquanto o usuário não inserir um número que se enquadre nessa condição, o problema segue pedindo a entrada adequada.

```
// calcula o logaritmo de um numero maior que zero
// usando while ate' a entrada adequada

#include <stdio.h>
#include <math.h>
#include <conio.h>

main()
{       float numero;

        do
        {       printf("\n\nEntre com um numero maior que zero: ");
                scanf("%f",&numero);
        } while (numero < 0);

        printf("\n log(%5.2f)=%5.2f\n", numero, log(numero));

        getch();
}
```

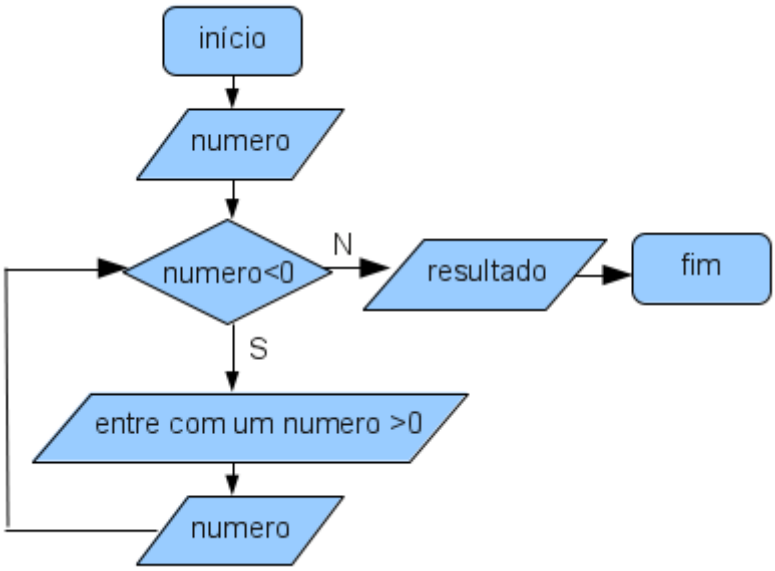
Como o looping é do tipo `do... while`, ele é executando pelo menos uma vez, já que o teste para saber se saímos ou não do looping é feito no final da estrutura. Escrevemos o texto pedindo pela entrada da variável, em seguida o valor é lido. É então feito o teste, que repete esses últimos comandos até que um número maior que zero seja inserido (para que o log possa ser calculado). Assim que o número adequado é fornecido, saímos do looping e o resultado do cálculo é impresso na tela.

Podemos escrever o mesmo código usando o `while` apenas.

```
// calcula o logaritmo de um numero maior que zero
// usando while ate' a entrada adequada

#include <stdio.h>
#include <math.h>
#include <conio.h>

main()
{       float numero;


        printf("\n\nEntre com um numero maior que zero: ");
        scanf("%f",&numero);

        while (numero < 0)
        {       printf("\n\nEntre com um numero maior que zero: ");
                scanf("%f",&numero);
        }

        printf("\n log(%5.2f)=%5.2f\n", numero, log(numero));

        getch();
}
```

Nesse caso, é escrita na tela a frase pedindo pelo número, e este é lido. Temos então um looping que entramos apenas se o número for menor que zero, lendo novamente o número até que seja digitado um número maior que zero. Em seguida, o resultado do cálculo é impresso na tela.

Em fluxograma:



Exercício: Faça um programa que calcule o produto de dois números positivos. Use um looping para garantir que o usuário entre com o número adequado, repetindo a entrada de dados até quando for necessário.

Programação de Computadores (lab) - aula 6

profa. Dra. Fabíola Ribeiro

1 Estruturas de repetição - *for*

Na aula passada vimos as estruturas de repetição, ou *looping* `while` e `do...while`. Veremos nessa aula mais uma estrutura de repetição, o `for`, que podemos entender como `para`. O `for` possui a seguinte estrutura:

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

temos que passar para o `for` 3 parâmetros, separadas por ponto e vírgula (;):

- i=0 - define a variável que usaremos como contador das repetições, e seu valor inicial.
- i<10 - condição obedecida para permanecermos no looping, a partir do momento que ela não é mais satisfeita, saímos do looping do `for`.
- i++ - incremento do contador, aqui *i* é aumentado de uma unidade (*i=i+1* ou *i++*).

O bloco a ser executado dentro do for deve ser delimitado por chaves {} se ele possuir mais que um comando.

Por exemplo, se quisermos repetir 3 vezes o comando `printf(‘‘*’’)` para imprimir 3 asteriscos na tela, usando `for`:

```
for(i=0;i<3;i++)
 printf(‘‘*’’);
```

Na primeira iteração temos *i=0*, é impresso um asterisco, e o contador *i* é incrementado para *i=0+1=1*. A condição *i<3* ainda é satisfeita, então permanecemos no *looping*.
Na segunda iteração temos *i=1*, é impresso outro asterisco, e o contador *i* é incrementado para *i=1+1=2*. A condição *i<3* ainda é satisfeita, então permanecemos no *looping*.
Na terceira iteração temos *i=2*, é impresso outro asterisco, e o contador *i* é incrementado para *i=2+1=3*. A condição *i<3* não é mais satisfeita, então saímos no *looping*.
Note que é executado o bloco abaixo do `for` e depois é feito o incremento no contador.

Podemos adaptar o *looping* acima para imprimir 200 asteriscos na tela, alterando apenas a condição de permanência no *looping* para *i<200*. Se usarmos uma condição que é sempre verdadeira (exemplo, *1<2*), o programa ficara em *looping*, sem terminar sua execução até que seja interrompido.

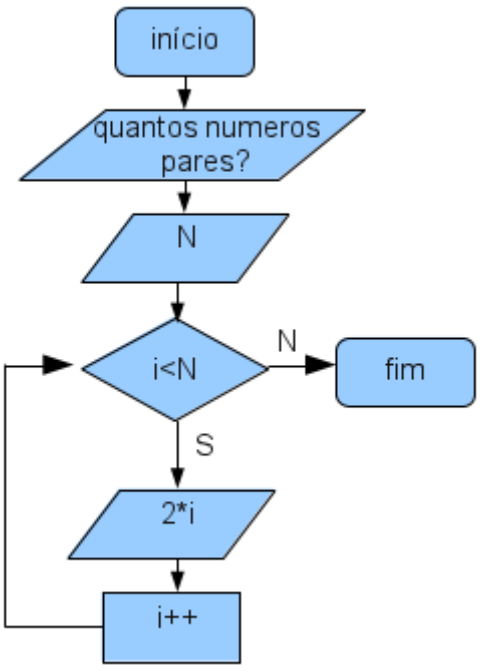
2 Aplicações

2.1 Escrever na tela os N primeiros números pares

Os números pares são aqueles da forma *2.n*, com *n=0,1,2....* Como é pedido os N primeiros números pares, N dado pelo usuário, iremos utilizar um `for` para N repetições. O primeiro número par é zero (*2.0=0*), logo o contador se inicia em *i=0*, e a condição de permanência no looping é *i<N*.

```
// Escreve os N primeiros numeros pares
// 11 out 2011


#include <iostream>      // biblioteca para cin e cout
#include <conio.h>       // biblioteca para o getch

using namespace std;     // para usar cin e cout

main()
{       int N,i;         // defino as variaveis
                         // int = numeros inteiros

        // escrevo na tela a frase pedindo pelo valor de N
        cout << "\n\n Quantos numeros pares devo imprimir? ";

        cin >> N;        // leio o valor de entrada e armazeno
                         // na variavel N

        cout << "\n\n"; // pulo duas linhas antes do resultado

        // looping para i de 0 ate' N, incrementando i de uma
        // unidade a cada repeticao
        for(i=0;i<N;i++)
        {       cout << " " << 2*i; // imprimo um espaco seguido
                                    // de um numero par (2*n)
        }

        getch();         // espero por uma tecla para fechar a
                         // janela
}
```

Como fluxograma:



Exercício: Faça um programa que escreva os N primeiros números múltiplos de 7, onde N deve ser dado pelo usuário do programa.

Exercício: Faça um programa que escreva os valores de 2^x para $0 \leq x < N$, onde N é dado pelo usuário.

2.2 Soma de uma série

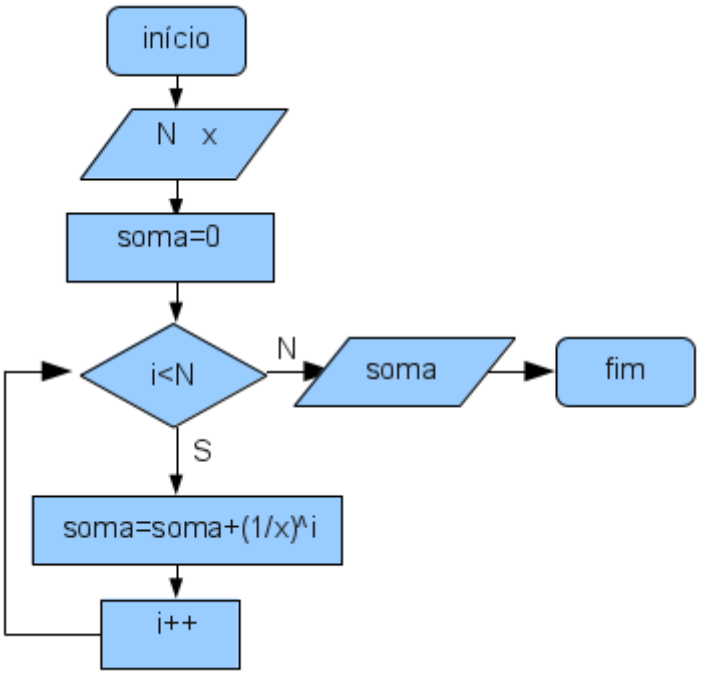
O programa a seguir calcula a soma dos números de uma série, onde cada termo é dado por

$$x_n = (1/x)^n \tag{1}$$

para um dado valor de x e para um dado número de termos n.

```c
/* Soma os N termos de uma serie, para um dado valor de x */
/* termo da forma x_n=(1/x)^n                             */
/* 11 out 2011                                           */

#include <stdio.h>
#include <math.h>
#include <conio.h>

main()
{       int N,i;
        float x, soma;

        /* atribuo valor inicial zero para a soma */
        soma = 0;

        printf("\n\nEntre com o numero de termos da série (N): ");
        scanf("%i",&N);

        printf("\n\nEntre com x: ");
        scanf("%f",&x);

        /* calculo a soma dos termos da série */
        for(i=1;i<=N;i++)
        {       soma=soma+pow(1/x,i);    //pow(a,b)=a^b
        }

        printf("\n\nA soma da serie e' %5.3f\n\n",soma);

        getch();
}
```

Em fluxograma:



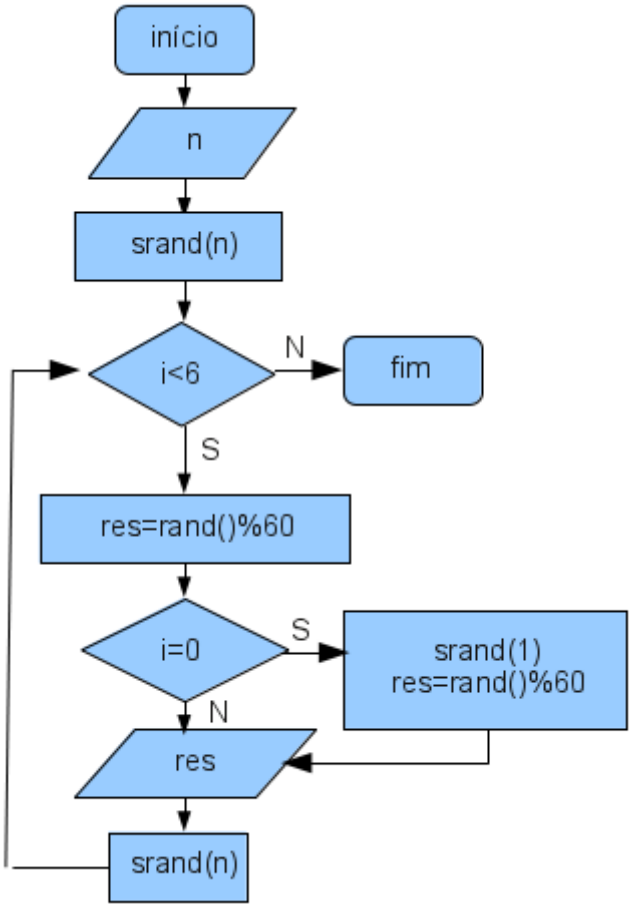
2.3 Gerador de palpite para megasena

No programa a seguir geramos 6 números aleatórios entre 1 e 60. Para gerar um número aleatório precisamos passar uma 'semente' para o gerador de números aleatórios. Se fornecermos sempre a mesma 'semente', é gerado sempre o mesmo número. Uma forma de garantir um número realmente aleatório seria utilizar o relógio como 'semente' para o gerador de numeros aleatórios, mas como nosso propósito não requer esse nível de detalhe, o usuário fornecerá a 'semente'.

Temos um `while` no programa para garantir que não seja gerado o número 0, já que a megasena aceita apostas de 1 a 60.

```
// palpite para a megasena
// o programa gera 5 numeros aleatorios entre 1 e 60
//
// 11 out 2011

#include <iostream>
#include <conio.h>
#include <stdlib.h>
using namespace std;

main()
{       int i;
        float n,res;

        cout << "Entre com um numero para gerar seus numeros da sorte: ";
        cin >> n;
        srand(n);       // passo a 'semente' a partir do qual os numeros
                        // aleatorios serao gerados

        cout << "\n\n";

        for(i=0;i<6;i++)          // looping para gerar os 6 numeros
        {       res = rand()%60; // gera um numero entre 0 e 60
                while(res==0)    // para garantir que nao saia numero 0
                {       srand(1);
                        res=rand()%60;
                }
                cout << " " << res; // imprime na tela
                srand(rand());
        }

        cout << "\nBoa sorte\n";
        getch();
}
```

Em fluxograma:


início
n
srand(n)
i<6
N
fim
S
res=rand()%60
i=0
S
srand(1)
res=rand()%60
N
res
srand(n)