INTRODUÇÃO À LINGUAGEM C++

1 - VARIÁVEIS

Variáveis – espaço de memória reservado para armazenar tipos de dados, com um nome para referenciar seu conteúdo.

Observações importantes

- Todas as variáveis devem ser declaradas antes de serem usadas.
- Mais de uma variável do mesmo tipo: separam-se seus nomes por vírgulas.

Nomes de variáveis

- Escolher nomes significativos;
- Apenas 32 caracteres do nome da variável são considerados.
- Pode-se usar quantos caracteres forem necessários, sendo o primeiro, obrigatoriamente, uma letra ou o caracter "_".

Tipos de variáveis

Tipo **int**:
- Representam inteiros.
- **long int** – aumenta a capacidade de armazenamento do **int**.
- **unsigned int** – faz o programa aceitar apenas valores positivos, economizando memória.

Regras:
- Número positivo – não é necessário o sinal de +.
- Números negativos – usa-se o sinal de -.
- Pontos decimais não podem ser usados quando escrevemos inteiros.
- Vírgulas não podem ser usadas para escrever inteiros.

Tipo **float**:
- Representação de números reais.
- Sinais: como em int.
- Valores devem ser expressos com o uso de ponto. Ex : 10.34

Tipo **double**:
- Semelhante ao float, porém com mais memória.

Tipo **char**:
- Representação de caracteres.
- Incluem letras do alfabeto (maiúsculas e minúsculas), dígitos de 0 à 9 e símbolos especiais, tais como #, &, !, +, -, *, /, etc.
- Devem ser escritos entre apóstrofos.

String :

- Não é uma variável.
- Representa conjunto de caracteres.
- Aparece entre aspas. EX: letra=”A”;

INICIALIZANDO VARIÁVEIS

É possível combinar uma declaração de variável com o operador de atribuição para que uma variável tenha um valor inicial no instante da sus declaração.

Exemplo:

```
#include<iostream.h>
#include<conio.h>
void  main( )
{
   int evento=5; //cria a variável do tipo inteiro que recebe o valor 5
   char corrida=’C’; //variável do tipo caracter que recebe o valor C
   float tempo = 27.25; //variável real que recebe 27.25
   cout<<”\nTempo: “<<tempo;
   cout<<”\nCorrida: “<<corrida;
   cout<<”\nCompetição: “<<evento;
   getch( );
}
```

2 - OPERADORES

Atribuição: =
Ex: x=3;
x=y=3;

Aritméticos: +, -, *, /, %.
OBS: / (divisão inteira)
% (resto da divisão – módulo)
Ex: int result, resto;
result = 5/2; //o conteúdo de result será 2 (parte inteira)
resto=5%2; // o conteúdo de resto será 1 (resto da divisão)

PRECEDÊNCIA DE OPERADORES

Forma de avaliação da expressão.

```
int x=5, y=3, z=1, n;
n=z + y*x; //O valor de n será 16
```

O operador * tem precedência maior que o operador +.
O uso de parênteses pode mudar essa ordem. Expressões entre parênteses são avaliadas primeiro.

n = (z + y) * x; //O valor de n será 20

Os operadores + e - têm a mesma precedência e a regra de associatividade é da esquerda para a direita.

Os operadores *, / e % têm a mesma precedência e associatividade da esquerda para a direita.

n = z % y / x //o resto da divisão de z por y será dividido por x.

ORDEM DE AVALIAÇÃO

- Parênteses
- *, /, %
- + , -

OPERADOR DE INCREMENTO E DECREMENTO

X= X + 1;
X++; X--; (pós-fixados)
++X, --X; (pré-fixados)

EX:
a = x + y; → a=8 x=3 e y=5
a = x++; → a=3 e x=4 // a atribuição precede o incremento mas não a soma
a = ++y; → a=6 e y=6 //incrementa o y e atribui ao a.
(a = y = y+1)

OPERADORES ARITMÉTICOS DE ATRIBUIÇÃO

+= (exemplo: x+=6 ⇔ x = x+6)
= (exemplo: x=6 ⇔ x = x*6)
/= (exemplo: x/=6 ⇔ x = x/6)
-= (exemplo: x-=6 ⇔ x = x-6)
%= (exemplo: x%=6 ⇔ x = x%6)

OPERADORES RELACIONAIS

OPERADOR	SIGNIFICADO
= =	Igual a
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que
!=	D**if**erente

OPERADORES LÓGICOS

OPERADOR	SIGN**IF**ICADO
&&	E
||	OU
!	NÃO

OPERADOR CONDICIONAL TERCIÁRIO

?:

valor = (a>b)?a:b; //se for verdadeira, pega o valor anterior aos dois pontos e armazena na variável valor

Exemplo:

Resposta = (a+b>c)?a:b;

a = 3;

b = 2

c = 5

Analisando a expressão, veremos que o seu resultado será FALSO. Logo, o valor armazenado em resposta será 2.

3 - ANALISANDO UM PEQUENO EXEMPLO

```
1  #include<iostream.h>
2  //Programa Exemplo 1
3  void main( )
4  {
5          int idade;
6          idade=22;
7          cout<<"A minha idade é"<<idade;
8  }
```

1ª linha: inclui um arquivo de biblioteca (conjunto de funções existentes no ambiente Borland C++)

2ª linha: comentário

3ª linha: declaração da função principal

4ª linha: início da função principal

5ª linha: determinando uma variável

6ª linha: atribuição do valor 22 para a variável idade.

7ª linha: impressão de uma mensagem e o conteúdo da variável idade.

8ª linha: fim da função principal.

4 - COMANDOS DE ENTRADA E SAÍDA

As instruções **cin** e **cout** são encontradas na biblioteca **iostream.h**. são responsáveis pela entrada e saída de dados.

cin>>idade; //coloca na variável idade o valor lido do teclado

cout<<"A minha idade é"<<idade; //emite para a tela todos os dados inteiros armazenados em variáveis, além de strings e caracteres entre aspas duplas.

A função getch( ) está contida na biblioteca **conio.h**. Faz com que o programa aguarde o pressionamento de alguma tecla para continuar.

5 - COMANDOS DE DECISÃO ESTRUTURA CONDICIONAL

Até o momento, estudamos apenas a estrutura seqüencial, ou seja, a estrutura na qual as ações são executadas seqüencialmente, na ordem que aparecem no programa.

Na estrutura condicional é possível determinar qual será a ação a ser executada, com base no resultado de uma expressão condicional

O COMANDO IF

Sintaxe do comando **if**

```
if ( condição)
{
        instrução 1;
        instrução 2;
        instrução 3;
 }
```

Exemplo:

```
#include<iostream.h>
#include<conio.h>
void main ( )
{
   int anos;
   cout<<"Quantos anos você tem? ":
   cin>>anos;
   if (anos<30)
           cout<<"\n Você é muito jovem!";
   getch( );
}
```

O COMANDO IF-ELSE

Neste caso temos dois blocos de ações (instruções)> o primeiro deles está associado ao comando **if** e o segundo, ao comando **else**.

```
if (condição)
{
    instrução 1;
    instrução 2;
    instrução n;
}
else
{
    instrução 7;
    instrução 8;
    instrução n;
}
```

O COMANDO IF-ELSE-IF

Sintaxe

```
if (condição 1)
{
    instrução 1;
    instrução 2;
    instrução n;
}
else if (condição 2)
{
    instrução 7;
    instrução 8;
    instrução n;
}
else
{
    instrução;
}
```

IF's ANINHADOS

Um **if** aninhado é uma declaração **if** que é colocada dentro de um outro **if** ou um **else**.

Ex:

```
if (X > 0 )
{
   if (y == 0)
   {
            a = x/y;
   }
}
else
{
   a = y;
}
```

SWITCH

Sintaxe:

```
switch ( variável)
{
      case constante 1;
            instrução1;
            instrução2;
            break;
      case constante 2;
            instrução;
      default
            instrução;
            instrução;
```

Há três coisas importantes sobre o comando **switch**:

1. Ele difere do **if**, pois só testa igualdades.
2. Nunca, dentro do mesmo **switch**, duas constantes **case** poderão ter valores iguais.
3. Uma declaração **switch** corresponde a um encadeamento **if-else-if**, porém é mais eficiente.

No **switch**, uma variável é sucessivamente testada contra uma lista de inteiros ou constantes caracter. Quando uma igualdade é encontrada, a instrução ou seqüência de instruções associada ao **case** da constante é executada.

O corpo de cada **case** é composto por um número qualquer de instruções. Geralmente a última instrução é o break. O comando break causa a saída imediata de todo o corpo do **switch**. Na falta do comando break, todas as instruções após o **case** escolhido serão executadas, mesmo as que pertençam ao **case** seguinte.

A declaração **default** é executada se nenhuma igualdade for encontrada.

Nota: o **default** é opcional.

Exemplo: programa que executa operações matemáticas

```
#include<iostream.h>
#include<conio.h>
void main ( )
{
        float n1, n2;
        char opcao;
        cout<<"\nDigite numero operador numero";
        cin>>n1>>opcao|>>n2;
        switch (opcao)
        {
                case '+':
                        cout<<(n1+n2);
                        break;
                case '-':
                        cout<<(n1-n2);
                        break;
                case '*':
                        cout<<(n1*n2);
                        break;
                case '/':
                        cout<<(n1/n2);
                        break;
               default:
                        cout<<"\nOperador desconhecido!";
          }
          getch();
}
```

6 - COMANDOS DE REPETIÇÃO

Os comandos de repetição, também chamados de laços, permitem que um conjunto de instruções seja repetido enquanto uma certa condição estiver sendo satisfeita.

O laço **for** é geralmente usado quando queremos repetir algo, um número fixo de vezes.

Sintaxe

```
for (inicialização; condição de teste ; incremento)
{
        Instrução;
        Instrução2;
        Instrução n;
}
```

A inicialização é uma instrução de atribuição e é sempre executada uma única vez, antes do laço ser iniciado.

O teste é uma condição avaliada como verdadeira ou falsa que controla o laço. Esta expressão é avaliada a cada volta no laço.

Quando o teste se torna falso, o laço é encerrado.

O incremento define a maneira pela qual a variável será alterada a cada volta no laço. É executada sempre, imediatamente após cada volta no laço.

Exemplo:

```
#include<iostream.h>
void main ( )
{
   for (int i=1; i,=20; i++)
   {
          cout<<"\n"<<i;
   }
}
```

FLEXIBILIDADE DO LAÇO FOR

1 – O OPERADOR VÍRGULA

A vírgula possibilita que dentro de um laço **for** estejam contidas várias instruções separadas por vírgulas. Um par de expressões separadas por vírgula é avaliada da esquerda para a direita.

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

for (int i=0, j=0; (i+j)<100 ; i++, j++)
   cout<<(i+j);

for (int i=0, j=10; i<10; i++, j++)
```

2 – USANDO CARACTERES

A variável do laço **for** pode ser do tipo char.
Exemplo:

```
for (char ch='a'; ch<='z'; ch++)
   cout<<"\nO codigo ASC de"<<ch<<"e"<<int(ch);
```

3 – OMITINDO EXPRESSÕES NO LAÇO FOR

Qualquer uma das três expressões de um laço **for** pode ser omitida, embora os ponto-e-vírgulas devam permanecer. Se a expressão de inicialização ou a de incremento forem omitidas, serão simplesmente desconsideradas. Se a condição de teste não estiver presente, será considerada permanentemente verdadeira.

O LAÇO WHILE

Um laço **while** é apropriado em situações em que o laço pode ser terminado inesperadamente, por condições desenvolvidas dentro do laço.

Sintaxe:

```
while (expressão teste)
{
        instrução;
        instrução;
 }
```

Se o teste for verdadeiro, o laço **while** é executado uma vez e a expressão de teste é avaliada novamente. Este ciclo de teste e execução é repetido até que o teste se torne falso.

O corpo do **while** pode ser uma única instrução terminada por ponto-e-virgula, ou várias instruções entre chaves ou, ainda, nenhuma instrução, mantendo o ponto-e-vírgula.

Em geral, o laço **while** pode substituir um laço **for** da seguinte forma:

```
for (int i=0; i<20; i++)
    cout<<i;

        ⇓
int i=0;
while (i<20)
{
    cout<<i;
    i++;
}
```

LAÇO DO-WHILE

É bastante similar ao **while** e é utilizado em situações em que é necessário executar o corpo do laço uma primeira vez e, depois avaliar a expressão ou teste e criar um ciclo repetido.

Sintaxe:

```
do {
        instrução;
        instrução;
}while (condição-teste);
```

7 – COMPARAÇÕES

PORTUGOL	**C++**
←	=
=	= =
≠	!=
<	<
>	>
≤	<=
≥	>=
/	/
div	/
resto	%
E	&&
OU	||
NÃO	!
se	**if**
senão	**else**
senão se	**else if**
enquanto	**while**
para	**for**
inteiro	**int**
real	**float**
caracter	**char**