

## O que me incentivou ?

Já vai fazer um ano que eu me cadastrei no fórum PDJ. Desde então, reparei que a grande dúvida de todos os iniciantes é: POR ONDE EU COMEÇO

Pensando nisso, comecei a escrever esse e-book para ensinar desde o mais básico até o mais avançado da programação de jogos. Os exercícios, como vocês podem ver, não têm “respostas no final”. Não tem mesmo, não foi por que eu esqueci. A idéia é que todos resolvam os exercícios utilizando o fórum (do PDJ e/ou do GameDev), por que assim, ajudando um ao outro, acredito que seja mais fácil aprender.

No final do “curso”, o “aluno” deverá saber:

- Algoritmos
- C++
- Blender 3D
- OpenGL
- GIMP

Bom, chega de conversa fiada e vamos ao trabalho !!!

Site PDJ: <http://www.pdj.com.br/>

Site GameDev: <http://www.gamedev.com.br/>

## Introdução

A idéia desse “curso” é ensinar ao leitor como montar algoritmos e logo depois traduzi-los para c++ sendo os algoritmos voltados à programação de jogos. Nesta parte do curso você não aprenderá a utilizar imagens, objetos ou modelos seja em 2D ou 3D. Você aprenderá a montar a estrutura do jogo em c++. Para utilizar imagens,... você usará como auxílio o OpenGL por exemplo. Mas isso é assunto para outra parte do livro. Por hora, nos preocupemos em aprender a melhor parte. Programação.

No começo, é apenas um livro de c++ normal. Quando já soubermos repetições, variáveis, funções e estruturas entraremos na programação de joguinhos simples (coisas como RPG's de papel, estilo D&D). Não desanime. Tudo tem um começo. O começo para fazer jogos é o que você verá a seguir.

Obs.: Algumas vezes os usuários mais experientes verão que eu escrevi uma ou outra besteira... for proposital ! Leia o livro todo e verá “correções” mais para a frente.

## Capítulo 1 – As variáveis

*{Zankoku na tenshi no you ni  
Shounen yo shinwa ni nare - Evangelion}  
Como um anjo cruel  
jovem torne-se uma lenda*

Não vou ficar com conversa fiada. Vamos ao que interessa !!

Primeiramente é necessário aprender o funcionamento de algoritmos. Para isso mostrarei um algoritmo bem simples e explicarei o seu funcionamento logo depois.

### Algoritmo

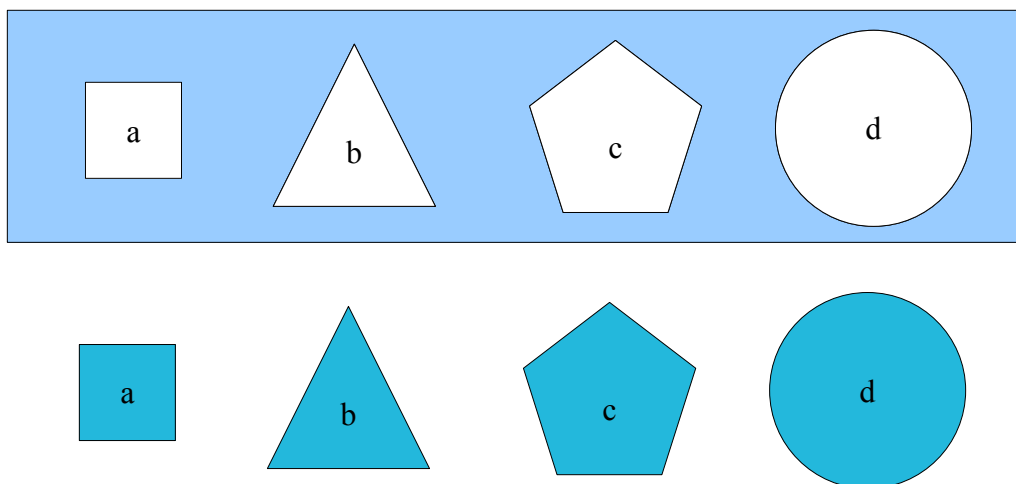
```
declare i, j numérico  
i <- 0  
j <- 0  
escreva "digite um valor"  
leia i  
escreva "digite outro valor"  
leia j  
escreva "a soma dos valores é" i+j
```

### fim algoritmo

“Algoritmo” e “fim algoritmo” dispensa comentários.

“declare” é uma palavra-chave. Ela diz que você está declarando uma variável.

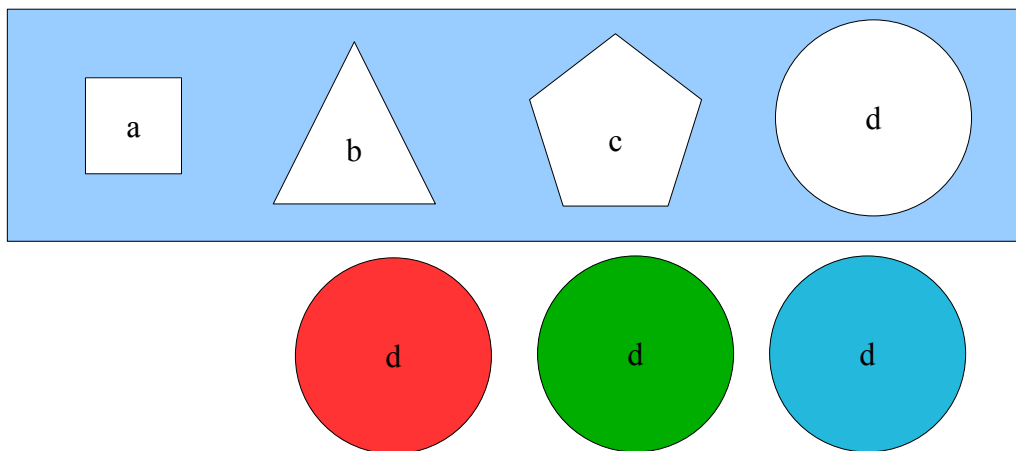
O que é uma variável ouço você dizer. Para entender o funcionamento de uma variável imagine aquele “brinquedinho” em que temos vários buracos de diferentes formas e devemos introduzir o objeto correspondente à sua forma. (Fig. 1)



Quando você declara uma variável você pode imaginar analogamente que você está fazendo um buraco de uma determinada forma. As letras “a”, “b”, “c” e “d” são os nomes desses buracos. A palavra-chave “numérico” está indicando a forma desse buraco (no algoritmo, “i” e “j” possuem a mesma forma. Analogamente você poderia dizer que “i” e “j” são circulares por exemplo).

As duas linhas seguintes dizem “i<-0” e “j<-0”. Isso significa que você está dizendo que o valor numérico de “i” é “0” e o mesmo para o “j”.

Como o nome diz, variável significa “que varia”, ou seja, você pode atribuir o valor que você quiser à variável no momento em que desejar. Analogamente ao “brinquedo das formas” você pode imaginar que os objetos podem ter cores diferentes (as variáveis “i” e “j” podem ter vários valores, contanto que seja um número). (Fig 2)



A palavra-chave “escreva” diz ao computador “escreva isso aqui”. Uma frase deve ficar entre parênteses. Quando você se refere a uma variável você não coloca parênteses.

A palavra-chave “leia” diz ao computador “manda o usuário digitar um valor para essa variável”.

Traduzindo para o c++, teríamos:

```
#include <iostream.h>
int main()
{
    int i = 0;
    int j = 0;
    cout << “digite um valor”;
    cin >> i;
    cout << “digite outro valor”;
    cin >> j;
    cout << “a soma dos valores é” i+j;
    return(0);
}
```

“#include <iostream.h>” significa: “Computador: Procure o significado dos meus códigos nessa biblioteca (iostream.h, no caso)”.

Em “int main(){}” você disse ao computador que o seu código principal está lá dentro

(dentro das chaves). Funções ou qualquer outra coisa, veremos mais pra frente. Por enquanto vamos trabalhar dentro do “main(){}” somente.

“int i = 0” é o mesmo que

“declare i numérico

i<=0”,

o mesmo serve para o “j”.

“cout << “texto”,” é o mesmo que “escreva “texto” ”

“cin >>,” é o mesmo que “leia”

Por enquanto, esqueça o “return(0);”, digite e não reclame ;-).

Obs.: Repare que na declaração das variáveis “i” e “j”, utiliza-se a palavra-chave “int”. Isso se deve ao fato de que as linguagens de programação não aceitam um valor que tenha um ponto flutuante (3,1415, por exemplo) atribuído a uma variável declarada como inteira (int). Para utilizar números com ponto flutuante, utiliza-se, no lugar de “int”, a palavra-chave “float”.

Agora veja o seguinte algoritmo:

### Algoritmo

declare a, b numérico

a <- 0

b <- 0

escreva “Digite um valor para a:”

leia a

escreva “Digite um valor para b:”

leia b

escreva a+b

fim algoritmo

Esse algoritmo é muito simples. Ele pede para digitar um valor para “a” e um valor para “b”. Depois de digitados ele escreve a soma dos dois. Você pode fazer isso com todos os operadores. Veja a lista deles:

<i>Operador</i>	<i>Função</i>	<i>Exemplo</i>
+	Soma	a+b
-	Subtração	a-b
*	Multiplicação	a*b
/	Divisão	a/b

Em C++:

```
#include <iostream.h>
```

```
int main()
```

```
{
```

```
    int a = 0;
```

```
    int b = 0;
```

```
    cout << “Digite um valor para a:”;
```

```
    cin >> a;
```

```
    cout >> “Digite um valor para b:”;
```

```
cin >> b;  
cout << a+b;  
return(0);  
}
```

Nos jogos:

Na programação de jogos você vai usar muuuitas variáveis. Mas é fácil perceber que você poderia utilizar esse recurso para, por exemplo, armazenar quantos tiros, vidas, HP, mana, etc o seu personagem possui. Como incrementar ou decrementar você verá adiante.

### **Exercícios:**

- 1) Faça um algoritmo que declare 3 variáveis (escolha o nome que quiser, com quantas letras quiser\* (minha\_variavel, por exemplo)). Organize 2 delas em ordem crescente (qualquer que seja o valor digitado).
- 2) Faça um algoritmo peça ao usuário para digitar valores para x na seguinte equação:  $y = x^2 + 2x + 1$ . Escreva o valor de y para cada valor de x digitado.
- 3) Traduza os algoritmos para c++.

\* Não se esqueça que as regras de nomenclatura são semelhantes às da internet (sem espaços, pontuação) e também não pode começar com números.

## Capítulo 2 – Condicionais

*{ Hakuna Matata, isto é, sem problemas – Timão e Pumba }*

Imagine que você queira que, dependendo do valor digitado, o programa execute uma função diferente, por exemplo:

### Algoritmo

```
declare cond, a, b numérico
a <- 0
b <- 0
cond <- 0
escreva "Digite um valor para a"
leia a
escreva "Digite um valor para b"
leia b
escreva "O que você quer fazer? 1 = a+b e 2 = a-b"
leia cond
se cond = 1
    então
        escreva a+b
fim se
se cond = 2
    então
        escreva a-b
fim se
fim algoritmo
```

Nesse caso o algoritmo pergunta se o usuário quer somar ou subtrair os dois valores digitados. É simples não ?

Agora veja em C++:

```
#include <iostream.h>
int main()
{
    int a = 0;
    int b = 0;
    int cond = 0;
    cout << "Digite um valor para a";
    cin >> a;
    cout << "Digite um valor para b";
    cin >> b;
    cout << "O que você quer fazer? 1 = a+b e 2 = a-b";
    cin >> cond;
    if(cond == 1){
        cout << a+b;
    }
    if(cond == 2){
        cout << a-b;
    }
}
```

```
    return(0);  
}
```

Imagine agora que você gostaria que uma certa ação seja repetida um número finito de vezes, sendo que este número pode ou não ser determinado por você, por exemplo: Você tem 100 alunos. Você quer digitar as 4 notas dos seus alunos em um programa e quer que este calcule a média. Você poderia fazer assim:

#### Algoritmo

```
declare nota1, nota2, nota3, nota4, média numérico  
escreva "digite a nota 1"  
leia nota1  
escreva "digite a nota 2"  
leia nota2  
.  
.  
.  
média <- (nota1+nota2+nota3+nota4)/4  
escreva média
```

fim algoritmo

E cada vez que você quisesse calcular a média de um novo aluno você deveria reiniciar o programa ou chamar a função de alguma maneira ↯. Ou, talvez fosse melhor utilizar uma estrutura de repetição.

Veja um método inteligente de utilizar uma estrutura de repetição nesse caso (das notas):

#### Algoritmo

```
declare nota, média, contador, contaaluno numérico  
média <- 0  
contador <- 1  
contaaluno <- 1  
repita  
    se contaaluno > 100  
        então interrompa  
    fim se  
    repita  
        se contador > 4  
            então interrompa  
        fim se  
        escreva "digite a nota" contador  
        leia nota  
        média <- média + nota  
        contador <- contador + 1  
    fim repita  
    escreva "a média do aluno" contaaluno "é" média/4  
    contaaluno <- contaaluno + 1
```

fim repita

fim algoritmo

O programa ficou mais complexo, porém, ficou mais funcional. Antes você tinha todos

aqueles problemas. Agora ficou mais inteligente. Essa é a função de uma estrutura de repetição.

Veja agora a explicação do código (o que você ainda não sabe):

A palavra-chave `repita`, dispensa comentários. O que está dentro de:

`repita`

`.  
. .  
. .`

`fim repita`

É executado um certo número de vezes. O número de vezes é determinado pela condição de parada. Repare nas linhas:

`se` (alguma coisa ocorrer)

`então interrompa`

`fim se`

A palavra-chave “interrompa” se refere à repetição em que esta está incluída. Resumindo, quando o programa verifica e vê que a condição ( $x > 9$ , por exemplo) é verdadeira ele interrompe a repetição em que o “se...fim se” está incluído. Para que o número atinja a condição de parada, alguma variável deverá ser incrementada a cada ciclo. Veja:

```
/* declaração e inicialização da variável x */
```

```
repita
```

```
se x > 9
```

```
então interrompa
```

```
fim se
```

```
escreva x
```

```
x <- x + 1
```

```
fim repita
```

Quando o computador ler “fim repita” ele voltará para “repita” e começará um novo ciclo. Se o  $x$  não for maior que 9 (5, por exemplo) ele continuará. Na tela aparecerá:

5

Depois o  $x$  (que está valendo 5) aumentará 1 (passará a ser 6)

Como 6 não é maior que 9, na tela aparecerá:

6

Enfim, acho que você entendeu.

Como o C++ possui várias estruturas de repetição, essas serão abordadas em capítulos distintos.

Para finalizar este capítulo, imagine que você queira que o programa apresente dois comportamentos distintos, dependendo do resultado da condição. Veja:

```
/* declaração e inicialização da variável x */  
repita  
    se x > 9  
        então interrompa  
        senão  
            escreva x  
    fim se  
    x <- x + 1  
fim repita
```

Enquanto o x não for maior que 9, o programa escreverá “ainda não chegou lá”

Para entender melhor, veja a simulação (suponha que x começa valendo 1):

Vamos simular o algoritmo abaixo: (Obs.: o que estiver entre “/\* \*/” ou depois de “//” são comentários meus... não aparecerá na tela ;-))

```
/* declaração e inicialização da variável x */  
repita  
    se x > 9  
        então interrompa  
    fim se  
    escreva x  
    x <- x + 1  
fim repita
```

```
// (1>9... NÃO)  
1  
// (2>9... NÃO)  
2  
// (3>9... NÃO)  
3  
// (4>9... NÃO)  
4  
// (5>9... NÃO)  
5  
// (6>9... NÃO)  
6  
// (7>9... NÃO)  
7  
// (8>9... NÃO)  
8  
// (9>9... NÃO)  
9  
// (10>9... SIM)  
/* o programa finaliza */
```

Agora veja a simulação do seguinte algoritmo:

```
/* declaração e inicialização da variável x */
repita
    se x > 9
        então interrompa
        senão se x < 5
            então escreva "x ainda é menor que 5"
        fim se
    fim se
    escreva "x é igual a" x
    x <- x + 1
fim repita
```

```
// (1>9... NÃO)
x ainda é menor que 5
x é igual a 1
// (2>9... NÃO)
x ainda é menor que 5
x é igual a 2
// (3>9... NÃO)
x ainda é menor que 5
x é igual a 3
// (4>9... NÃO)
x ainda é menor que 5
x é igual a 4
.
.
// (10>9... SIM)
x é igual a 10
```

Repare que enquanto x foi menor que 5 ele escreveu "x ainda é menor que 5". Isso se deve ao "senão se". Uma boa idéia para entender algoritmos é fazer os seus próprios e simulá-los. Simule também os que eu não simulei e veja os resultados.

Nos jogos: Você poderia usar uma estrutura de repetição, por exemplo, para que enquanto o contador de vidas fosse maior que 0 a tela de game-over não aparecesse. Poderia ainda, utilizar para que a cada vez que você pressionasse o botão "ctrl" seria decrementado 1 do contador de tiros. Existem centenas de utilidades nos jogos.

### Exercícios:

1) Faça um algoritmo (não traduza para c++) que calcule com grande precisão a raiz de qualquer número (positivo óbvio).

Obs.: Para calcular a raiz de um número, use o seguinte esquema:

número que se deseja descobrir a raiz = x

possível valor da raiz = y

$y = (x/y+y)/2$       Por exemplo, raiz de 2:  $(2/1 + 1)/2 = 1,5$  >>  $(2/1,5 + 1,5)/2 = 1,41666$   
>> ...

A repetição deve ocorrer um número grande de vezes (10.000 vezes, por exemplo).

## Capítulo 3 – O uso do “se”, ou melhor, “if” no c++

*{ Me diga seu nome... Shinta... Muito suave para um espadachin  
de agora em diante seu nome será  
KENSHIN – OVA's Kenshin Himura}*

Algumas vezes, simplesmente queremos que o programa escolha um caminho dependendo do resultado e outro caminho no caso contrário. Para isso serve o “if”. Veja o trecho de algoritmo: (você já viu “se...fim se” em algoritmo !!)

```
//declaração de variáveis e tudo mais
repita
    se x > 9
        então interrompa
        senão
            escreva x
    fim se
    x <- x + 1
fim repita
//qualquer coisa relevante
```

Repare que o se está dividido em duas partes: “então” e “senão”. Ou seja, “se alguma coisa” então faça isso, senão faça aquilo. Essa é a idéia do if. Agora vamos ao que interessa. A sintaxe básica é essa:

```
//começo do programa
if(condição){
    //ações
}
else{
    //mais ações
}
//fim do programa
```

A parte “if(condição){}” equivale a “se condição então”. Já “else{ }” equivale a “senão”. Veja o primeiro exemplo do capítulo traduzido para o c++ (sem o repita por enquanto ok?).

```
//declaração de variáveis e tudo mais
//alguma estrutura de repetição
if(x > 9){
    break;
}
else{
    cout << x;
}
x = x + 1;
//fim da estrutura de repetição
//qualquer coisa relevante
```

Não há segredos como você pôde ver. Assim funciona a estrutura if. Antes de continuarmos, vejamos a tabela de operadores:

Operador	Significado	Exemplo
==	Igual a	A == B
!=	Diferente de	A != B
<	Menor que	A < B
>	Maior que	A > B
<=	Menor ou igual	A <= B
>=	Maior ou igual	A >= B

Esses são os operadores de comparação. Use-os nas estruturas que necessitarem de algum tipo de comparação como o if.

Há uma técnica chamado “aninhamento”, onde você coloca um se dentro de outro, ou seja: se isso então, se isso, então se isso... faça isso senão isso.... Veja:

//coisas relevantes

```
se x > 0
    então //alguma coisa
    senão
        se x < 100
            então //mais alguma coisa
            senão
                se x > 51
                    //mais mais mais alguma coisa
                    senão
                        //outra coisa
                fim se
            //mais coisas XD
        fim se
    //finalmente faça isso !!
fim se
//fim do programa
```

Não há nada novo nesse código. É só uma técnica. Veja como fica em c++:

//coisas relevantes

```
if(x > 0){
    //alguma coisa
}
else{
    if(x < 100){
        //mais alguma coisa
    }
    else{
        if(x > 51){
            //mais mais mais alguma coisa
        }
        else{
```

Curso C++ para criação de jogos (Básico) – Por: Rodrigo da Silva Vaz

```
        //outra coisa
    }
    //mais coisas
}
//finalmente faça isso !!
}
//fim do programa
```

Eu sei, eu sei é meio confuso mas você se acostuma.

### Exercícios:

- 1) Faça um algoritmo que retorne resultados diferentes dependendo do número que o usuário digitar (de 0 a 5). Se o número for maior que 5 ou menor que 0, o algoritmo deve retornar (digite um número válido). Traduza para o c++.

## Capítulo 4 – Para escolher... use o switch case

*{ De um lado Narusegawa san... de outro Mutsumi san... cara isso é tortura  
Keitarô Urashima }*

Você deve ter ficado um pouco invocado e confuso com o aninhamento. Pois na hora de usar o aninhamento para apresentar escolhas, use o switch case (aninhamento sucks). Veja:

Escolha com aninhamento:

```
//coisas relevantes
if(letra == 'A'){
    //alguma coisa
}
else{
    if(letra == 'B'){
        //mais alguma coisa
    }
    else{
        if(letra == 'C'){
            //mais mais mais alguma coisa
        }
        else{
            //outra coisa
        }
    }
}
//fim do programa
```

Escolha com switch case:

```
//coisas relevantes
letra = toupper(letra);
switch (letra)
{
case 'A': /*alguma
coisa*/
break;
case 'B': /*mais alguma
coisa*/
break;
case 'C': /* mais mais mais
alguma coisa*/
break;
default: /*outra
coisa*/
}
//fim do programa
```

Acredito que não houve problemas a não se na palavra “toupper” e “break”. Toupper

Curso C++ para criação de jogos (Básico) – Por: Rodrigo da Silva Vaz

apenas converte uma letra para maiúscula caso o usuário utilize minúscula (c++ é case sensitive). O “break” para a repetição. Só isso.

Nos jogos: Nos jogos a utilização do switch case é evidente. Você mostra várias opções ao jogador e ele escolhe uma. Simples. É claro que se o menu do jogo for desenvolvido em ambiente gráfico, o switch case ficará esquecido (é só adicionar uma função para cada botão ↵). Existem algumas outras utilizações. Mas essa é óbvia.

**Exercícios:**

- 1) Faça o mesmo exercício da aula anterior utilizando o switch case (não faça o algoritmo).

## Capítulo 5 – “While” e “do... while”

*{ I do not know what strength is in my blood  
but I swear to you, I will not let the White City fall  
nor our people fail  
Eu não sei quanta força há no meu sangue  
mas eu lhe juro, eu não vou deixar a Cidade Branca cair  
nem nosso povo falhar – Aragorn – Senhor dos Anéis }*

Se você está curioso para saber como se escreve o “repita... fim repita” no c++, aqui vai um dos métodos:

### Algoritmo

```
declare a, contador numérico
a ← 0
contador ← 0
leia a
repita
    se contador > 2
        então interrompa
    fim se
    a ← a*a
    contador ← contador + 1
fim repita
escreva a
```

### fim algoritmo

Esse algoritmo é bem básico (simule e verá que é bem básico). Ele apenas eleva um número ao cubo.

Agora veja em c++:

```
#include <iostream.h>
int main()
{
    int a = 0;
    int contador = 0;
    cin >> a;
    while(contador < 3)
    {
        a = a*a;
        contador+=1;
    }
    cout << a;
    return(0);
}
```

While significa enquanto, ou seja, segundo o programa, enquanto “contador” for menor que 3, faça o que estiver dentro das chaves ( { ... } ). JAMAIS se esqueça de aumentar o contador em um determinado valor a cada iteração. Se você omitir o “contador+=1;” (que é exatamente a mesma coisa que “contador = contador + 1;”) o loop ficará infinito (o nome dessas “repetições” ou “iterações” é loop). Repare agora que no

algoritmo, a condição era (se contador for maior que 2 então interrompa). Já em c++ a condição é (enquanto contador for menor que 3, faça isso). Isso se deve ao fato de no algoritmo estarmos usando uma condicional “se” para parar a repetição. É apenas questão de ponto de vista, acho que você não se assustou com isso... certo ?

Veja agora o seguinte código:

```
#include <iostream.h>
int main()
{
    int a = 0;
    int contador = 0;
    cin >> a;
    while(++contador < 3)
    {
        a = a*a;
    }
    cout << a;
    return(0);
}
```

Na verdade ele faz EXATAMENTE a mesma coisa que o código anterior... O nome “disso” é notação prefixada. Ela funciona assim:

- O contador é aumentado
- A condição é testada
- Se for “true (verdadeiro)”, ele entrará na repetição. Se for “false (falso)” ele irá interromper

Há ainda a notação posfixada. Ela funciona assim:

- A condição é testada
- O contador é aumentado
- Se a condição (que já foi testada) for “true”, ele entrará na repetição. Se for “false” ele irá interromper

Veja o mesmo exemplo usando a notação posfixada:

```
#include <iostream.h>
int main()
{
    int a = 0;
    int contador = 0;
    cin >> a;
    while(contador++ < 3)
    {
        a = a*a;
    }
    cout << a;
    return(0);
}
```

Repare que no esquema acima, ele calculará o número elevado a quarta potência. Para corrigir isso, faça assim: “while(contador++ < 2)”. Pronto !! Erro corrigido. Agora, vejamos o “do... while”:

```
#include <iostream.h>
int main()
{
    int a = 0;
    int contador = 0;
    cin >> a;
    do
    {
        a = a*a;
        contador += 1;
    }while(contador < 2);
    cout << a;
    return(0);
}
```

Você deve estar pensando: Mas não é a mesma coisa ?!?!?!?! Não exatamente. O loop “do while” funciona de um jeito semelhante à notação posfixada. Mas a grande diferença entre o “while” e o “do while” é que no “do while” os comandos são executados pelo menos uma vez !! Isso se deve a ele testar a condição somente no final. Resumindo: “faça ... (ou seja ele fará de qualquer jeito) enquanto algo for verdadeiro”.

### Exercícios:

1) Traduza para o c++:

#### Algoritmo

```
declare nota, média, contador, contaaluno numérico
média <- 0
contador <- 1
contaaluno <- 1
repita
    se contaaluno > 100
        então interrompa
    fim se
    repita
        se contador > 4
            então interrompa
        fim se
        escreva “digite a nota” contador
        leia nota
        média <- média + nota
        contador <- contador + 1
    fim repita
    escreva “a média do aluno” contaaluno “é” média/4
    contaaluno <- contaaluno + 1
fim repita
fim algoritmo
```

## Capítulo 6 – O loop “for”

*{ One, two, three, four... I love marine corp. - Recrutas – Full Metal Jacket }*

Depois de ver o loop “while” e “do while”, o loop for parece meio inútil. Quando você já estiver experiente verá que não é necessariamente inútil. Veja o código:

```
#include <iostream.h>
int main()
{
    int a = 0;
    int contador;
    cin >> a;
    for(contador = 0; contador == 3; contador++)
    {
        a = a*a;
    }
    cout << a;
    return(0);
}
```

Veja a enorme semelhança com o loop “while” posfixado. Agora repare que neste caso a condição funciona de uma maneira diferente. No loop while, a condição normalmente é modificada (você verá isso mais para frente) dentro das condicionais “if”. No loop for, há um determinado número de vezes para acontecer a iteração, ou seja, você tem certeza de quantas vezes “a coisa” irá acontecer. No nosso caso, 3 vezes. Entenda agora, a sintaxe:

```
for(contador = 0; contador == 3; contador++)
```

```
for() { ... } // você apenas indicou que é um loop for
contador = 0; // você inicializou a variável
contador == 3; // condição (repita 3 vezes)
contador++ // incremente a variável contador
```

Você deve estar confuso com duas coisas (espero que sejam somente essas duas coisas). Primeiro, inicializar uma variável e segundo, incrementar a variável contador.

Inicializar uma variável é atribuir um valor inicial a ela.

A inicialização da variável não é exatamente obrigatória. Porém, mais para frente você verá que alguns erros ocorrem pela falta de inicialização. Para evitar isso faça como um bom programador (espero que essa seja sua meta). INICIALIZE TODAS AS VARIÁVEIS... SEMPRE. A partir de agora inicializaremos todas.

Incrementar a variável significa apenas aumentar seu valor em (+1). Você também pode decrementar (-1). A escolha é sua. Veja agora o “gabarito” para usar o loop for. Nunca fuja disso e você nunca terá problemas.

Curso C++ para criação de jogos (Básico) – Por: Rodrigo da Silva Vaz

```
#include <iostream.h>
int main()
{
    int contador;
    for(contador = 0; contador < 10; contador++)
    {
        cout << contador << endl; //endl pula uma linha
    }
    cout << a;
    return(0);
}
```

O programa irá mostrar os números de 0 a 9

Ou você também pode decrementar a variável:

```
#include <iostream.h>
int main()
{
    int contador;
    for(contador = 9; contador > 0; contador--)
    {
        cout << contador << endl; //endl pula uma linha
    }
    cout << a;
    return(0);
}
```

O programa irá mostrar os números de 9 a 0 XD

## Capítulo 7 – Algo sobre variáveis básicas...

{ Veja esta técnica pela última vez..  
o melhor presente de adeus que eu podia lhe dar... - Dohko }

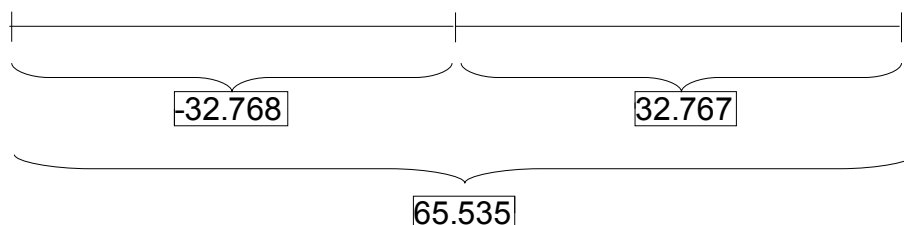
Lembra do primeiro capítulo, onde as variáveis podiam ter “formas” diferentes? Vou explicar direito nesse capítulo.

Para começar, a variável (sem viagens agora) é um espaço reservado na memória de seu computador. Ao declarar uma variável, você está reservando um espaço que somente o seu programa pode ter acesso e estará disponível até a finalização do programa. As variáveis podem ser de vários tipos básicos. São eles:

Tipo	Tamanho	Valores
unsigned short int	2 bytes	0 a 65.535
short int	2 bytes	- 32.768 a 32.767
unsigned long int	4 bytes	0 a 4.294.967.295
long int	4 bytes	-2.147.483.648 a 2.147.483.647
char	1 byte	256 valores de caractere
bool	1 byte	Verdadeiro ou falso
float	4 bytes	1,2e-38 a 3,4e38
double	8 bytes	2,2e-308 a 1,8e308

Você deve ter notado a “incrível” semelhança entre os nomes long int e short int e mais ainda entre unsigned (short int e long int). Na verdade as variáveis são somente int e short. Repare nos valores numéricos da “unsigned short” e “short” e da “unsigned int” e “int”. Some o módulo dos dois valores da short int:

$|-32.768 + 32.767| = 65.535$ . O resultado ser igual ao valor da “unsigned short” não é coincidência. Unsigned significa sem sinal, ou seja, ao dizer “sem sinal” você não aumentou o valor da variável. Você apenas afirmou que o resultado dela é positivo com certeza e com isso aumentou (no campo dos números positivos) o intervalo de valores sem utilizar o dobro de memória. O mesmo vale para a “unsigned int” que é igual à soma dos módulos dos dois valores da “int”.  $|-2.147.483.648| + |2.147.483.647| = 4.294.967.295$ . Para entender melhor (se você ainda não entendeu, veja o esquema:



Curso C++ para criação de jogos (Básico) – Por: Rodrigo da Silva Vaz

Nos jogos: Em jogos muito grandes, cada mínimo de memória que você puder economizar, não hesite... ECONOMIZE !!!

## Capítulo 8 – Variáveis compostas homogêneas unidimensionais

{ Rozan Hyakuryu-Ha – Cólera dos 100 dragões – Dohko }

Imagine a seguinte situação: Você precisa acessar, somar, multiplicar... centenas de valores um pelo outro. Então você pega e declara 400 variáveis inteiras (int) e 1000 variáveis com ponto flutuante(float) ⇝. CLARO QUE NÃO!! Para isso existem arrays ou variáveis estruturadas ou variáveis compostas. Veja o seguinte algoritmo:

### Algoritmo

```
declare s, i, a, b numérico
declare A[1:10] numérico
i <- 1
s <- 0
repita
    se i > 10
        então interrompa
    fim se
    escreva "Digite um valor:"
    leia a
    leia b
    A[i] <- a + b
    s <- s+A[i]
    i <- i+1
fim repita
escreva s
```

### fim algoritmo

Repare que não foi necessário declarar 10 variáveis A (A1, A2, A3, A4...). Foi necessário apenas digitar A[1:10] para o computador entender que na verdade não é uma são 10 !!! Repare também que eu usei uma outra variável para acessa-la. Isso foi feito para que o programa “percorresse o vetor”. A cada iteração o valor de “i” aumenta em uma unidade. Portanto a cada repetição, o programa acessa uma posição diferente do vetor. Veja o esquema:

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]

Suponha agora que o usuário digitou os 3 primeiros valores de a e b: a(1, 3, 5) e b(2, 4, 6). Veja o resultado final:

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
3	7	11							

Acho que a imagem fala por si só. O valor dentro dos colchetes (A[1], B[2]...), é o valor de i.

Veja agora o algoritmo traduzido:

```
#include <iostream.h>
int main()
{
    int s = 0;
    int i = 0;
    int a = 0;
    int b = 0;
    int A[9];
    //estrutura de repetição
    if(i > 10){
        break;
    }
    cout << endl<< "Digite um valor:";
    cin >> a;
    cin >> b;
    A[i] = a + b
    s = s+A[i]
    i = i+1
    //fim da estrutura de repetição
    cout << "\n" << s;
    return(0);
}
```

Repare que eu coloquei A[9] ao invés de A[10]. Isso se deve ao fato de o limite inferior do C++ ser 0 por padrão. De 0 a 9, temos 10 números. Não se preocupe com limite inferior por enquanto (mesmo porque no C++ não precisa se preocupar com isso). Uma “bitolinha” é sempre subtrair um do tamanho de seu vetor.

Nos jogos: Na programação de jogos com OpenGL ou DirectX, você ainda vai ouvir muito falar de arrays... Não tenha dúvidas.

### Exercícios:

- 1) Faça um algoritmo que leia e escreva 10 valores numéricos.
- 2) Faça um algoritmo que leia 2 arrays (A e B). Some os valores de A e B (A[1] + B[1], A[2] + B[2]...) e insira o resultado em um array C. Escreva os valores do vetor C.

## Capítulo 9 – Variáveis compostas homogêneas bidimensionais

{ Do you know: What It Is ? - The Matrix is everywhere – Morpheus }

Arrays bidimensionais são apenas os arrays comuns (unidimensionais) com várias linhas. Pense como se fossem matrizes. Matriz pode ser  $M_{1 \times 2}$ ,  $A_{1 \times 4}$  quando só possui colunas e  $M_{2 \times 2}$ ,  $B_{5 \times 8}$ , ou genericamente  $MATRIZ_{i \times j}$ , sendo  $i$  = linhas e  $j$  = colunas. Veja uma matriz  $1 \times 4$ :

$i1, j1$	$i1, j2$	$i1, j3$	$i1, j4$
----------	----------	----------	----------

Agora veja uma matriz  $3 \times 5$

$i1, j1$	$i1, j2$	$i1, j3$	$i1, j4$	$i1, j5$
$i2, j1$	$i2, j2$	$i2, j3$	$i2, j4$	$i2, j5$
$i3, j1$	$i3, j2$	$i3, j3$	$i3, j4$	$i3, j5$

Resumindo: Arrays bidimensionais servem para armazenar dados que contenham uma relação próxima entre si, por exemplo: Você quer armazenar todas as notas dos alunos de uma sala (sala com 20 alunos) e o número do aluno, portando uma matriz que tenha 2 colunas e 20 linhas ou 2 linhas e 20 colunas.

<i>Número do aluno</i>	<i>Nota do aluno</i>
1	7
2	6
3	8
4	2
.	.
.	.
.	.
20	9

A forma de representar em algoritmo é  $Matriz[1:2, 1:20]$ . Veja um exemplo simples para armazenar os dados:

### Algoritmo

```
declare s, i numérico
declare A[1:2, 1:20] numérico
i <- 1
s <- 0
repita
    se i > 20
        então interrompa
    fim se
    escreva "Digite a nota do aluno:"
    A[i][1] <- i
    leia A[i][2]
```

```
        s <- s+A[i][2]
        i <- i+1
    fim repita
    escreva s/20
fim algoritmo
```

O algoritmo apenas lê as notas e a cada repetição adiciona o número correspondente ao aluno. Veja a simulação:

<i>Número do aluno</i>	<i>Nota do aluno</i>
1	7

//repetição

<i>Número do aluno</i>	<i>Nota do aluno</i>
1	7
2	6

//acho que você entendeu !!!

Agora, como de praxe, veja o algoritmo traduzido:

```
#include <iostream.h>
int main()
{
    int s;
    int i;
    int A[1][19] // olha que legal... o c++ não faz questão de A[1:2][1:20]
    i = 0;
    s = 0;
    //estrutura de repetição
    if(i > 19){
        break;
    }
    cout << "Digite a nota do aluno:";
    A[i][0] = i;
    cin >> A[i][1];
    s = s+A[i][1];
    i = i+1;
    //fim da repetição
    cout << s/20;
    return(0);
}
```

Você deve ter se perguntado (por que diabos A[1:2]. Isso se deve a linguagens como pascal. O padrão é o seguinte: Matriz[limite inferior:limite superior]. Eu poderia falar muita coisa e te deixar confuso. Mas a função disso é a seguinte: Veja um exemplo com limite inferior 1 e limite superior 4 (Matriz[1:4]:

1	2	3	4

Agora veja uma matriz com limite inferior 3 e limite superior 8 (Matriz[3:8])

3	4	5	6	7	8

Mas é só isso?!?!?!?!... Sim é só isso ¬¬. Linguagens como o pascal são muito formais... exigem coisas meio chatas mesmo, vá se acostumando.

Nos jogos: Como dito no capítulo passado, você ainda vai ver muitas arrays na hora de fazer jogos (3D). Se acostume com isso também.

Exercícios:

- 1) Leia o algoritmo abaixo, simule-o e traduza-o:

### Algoritmo

```
declare média, i, j numérico
declare n_alunos[1:20][1:6]
i <- 1
j <- 1
média <- 0
repita
  se i > 20
    então interrompa
  fim se
  escreva "Aluno " i ": "
  repita
    se j > 4
      então interrompa
    fim se
    escreva "Escreva a nota " j "do aluno " i " : "
    n_alunos[i][j] <- i
    leia n_alunos[i][j+1]
    média <- média + n_alunos[i][j+1]
    j <- j+1
  fim repita
  n_alunos[i][6] <- média/4
  escreva "A média do aluno " i " é " n_alunos[i][6]
  média <- 0
  i <- i+1
fim repita
```

fim algoritmo

## Capítulo 10 – Variáveis compostas homogêneas tridimensionais

*{ Even now in this very room – Morpheus }*

Você já viu arrays unidimensionais (faça relação com uma reta), já viu bidimensionais (pense em um plano). Agora, veremos tridimensionais (sólido).

Veja a “evolução”:

j1	j2	j3
----	----	----

Unidimensional

i1, j1	i1, j2	i1, j3
i2, j1	i2, j2	i2, j3
i3, j1	i3, j2	i3, j3

Bidimensional

i1, j1, k1	i1, j2, k1	i1, j3, k1
i2, j1, k1	i2, j2, k1	i2, j3, k1
i3, j1, k1	i3, j2, k1	i3, j3, k1

<b>Página 1</b>	Página 2	Página 3
-----------------	----------	----------

Tridimensional (Página 1)

i1, j1, k2	i1, j2, k2	i1, j3, k2
i2, j1, k2	i2, j2, k2	i2, j3, k2
i3, j1, k2	i3, j2, k2	i3, j3, k2

Página 1	<b>Página 2</b>	Página 3
----------	-----------------	----------

Tridimensional (Página 2)

i1, j1, k3	i1, j2, k3	i1, j3, k3
i2, j1, k3	i2, j2, k3	i2, j3, k3
i3, j1, k3	i3, j2, k3	i3, j3, k3

Página 1	Página 2	<b>Página 3</b>
----------	----------	-----------------

Tridimensional (Página 3)

É só uma continuação intuitiva. É claro que se já temos linhas e colunas, faltam as páginas (cada página pode estar relacionada a um cliente, as colunas podem ser o salário e o número de horas de trabalho e as linhas cada mês do ano !!).

Veja:

<i>Mês</i>	<i>Horas extra</i>	<i>Salário</i>
1	10	3.000
2	4	2.400
<b>José(Pág. 1)</b>	João(Pág. 2)	Manoel(Pág. 3)

Simple não ? Agora veja como armazenar esses dados :

### Algoritmo

```
defina fator 100
declare i, conta, k numérico
declare tabela[1:12][1:3][1:20] numérico
i <- 1
conta <- 0
k <- 1
repita
  se k > 20
    então interrompa
  fim se
  repita
    se i > 12
      então interrompa
    fim se
    escreva "Digite os dados referentes ao mês" i
    repita
      se conta = 1
        então interrompa
      fim se
      escreva "Digite quantas horas extra" k "trabalhou:"
      leia tabela[i][2][k]
      tabela[i][3][k] <- (tabela[i][2][k]*fator)+2000
      conta <- 1
    fim repita
    conta <- 0
    i <- i+1
  fim repita
  k <- k+1
fim repita
```

### fim algoritmo

Quando você bateu o olho deve ter pensado "NOSSA!!!!!! que código gigante". Acredite. Esse código é muito pequeno e simples de entender. Leia, releia, trileia até entender tudo. Além disso, uma palavra nova apareceu "defina". Isso significa que você definiu uma constante, ou seja, um espaço na memória que não pode ser modificado e tem um valor determinado (ao contrário de uma variável que pode ser modificada a hora que você quiser).

Em c++:

```
#include <iostream.h>
#define fator 100
int main()
{
    int i = 0;
    int k = 0;
    int tabela[11][2][19];
    while(k < 20)
    {
        while(i < 12)
        {
            cout << "Digite os dados referentes ao mês" << i;
            for(conta = 0; conta < 1; conta++)
            {
                cout << "Digite quantas horas extra" << k <<
"trabalhou:";

                cin >> tabela[i][1][k];
                tabela[i][2][k] = (tabela[i][1][k]*fator)+2000;
            }
            i += 1;
        }
        k += 1;
    }
    return(0);
}
```

Só repare em duas coisas:

Primeiro: A constante foi declarada antes do main()

Segundo: Usei o "loop for"

### Exercícios:

1) Faça um sistema de cadastro de notas.

Aluno = página

Coluna { 1<sup>a</sup> = número do aluno, 2<sup>a</sup> = primeira nota, 3<sup>a</sup> = segunda nota e 4<sup>a</sup> =  
média final [ ( nota1+nota2 )/2 ] }

Linhas = bimestres (4 no total)

Obs.: Faça em algoritmo e logo após em c++.

## Capítulo 11 – Variáveis compostas heterogêneas

*{ Jiu-jitsu.. I going learn Jiu-jitsu ? – Neo }*

Quando você viu o sistema de cadastro, deve ter pensado: Será que eu não posso declarar uma variável composta homogênea unidimensional, cujo tipo seja caracteres para armazenar os nomes dos empregados, sendo esta variável intimamente ligada com a principal ??

Sim !! Existe uma técnica para isso. Vamos agora aos arrays paralelos.

**Próximos capítulos:**  
**Funções (capítulo muito longo XD)**  
**Estrutura (um pouco extenso também)**  
**Sobrecarga de operador**  
**Tratamento de arquivos**  
**OOP (muuuuuuito longo)**  
**Polimorfismo**  
**Ponteiros**  
**// aqui acaba a parte do c++ básico e intermediário**  
**C++ Avançado (bmp, wav...)**