Objetivos

É dar ao treinamento uma base de conhecimentos sobre as características da linguagem C++, permitindo que o mesmo possa se desenvolver na elaboração de programas aplicativos utilizando esta linguagem.

Enfoque

O enfoque está nas características das linguagens C e C++, na sintaxe dos programas e na lógica de resolução de problemas. São utilizados diversos programas-exemplos curtos e dirigidos ao assunto tratado no momento.

Pré-requisitos

Noções de programação de computadores e conhecimento de manipulação de arquivos e diretórios no DOS.

Histórico

- 1970 Ken Thompson desenvolve em um DEC PDP-7 a **linguagem B**;
- Era uma linguagem com muitas **limitações**;
- Em 1972, **Dennis Ritchie** e **Ken Thompson** criaram a **linguagem C**;
- A linguagem C passou um bom tempo sendo utilizada apenas por algumas pessoas;
- Em 1978, um ano histórico, **Brian Kernigham** e **Dennis Ritchie** escrevem **The C programing language**. E com este simples livro, tudo mudou;
- C deixou de ser de conhecimento restrito e **passou ao conhecimento público**;
- Em pouco tempo surgiram compiladores C para os computadores de 8 bits e sistema operacional CP/M;
- Quando começou a evolução dos PCs a linguagem C entrou em evidência;
- Esta popularidade se deve a um bom motivo. Ao contrário de muitas outras linguagens, C oferece ao programador um **grande poder sobre o computador**. C permite que você realize coisas que outras linguagens

não permitem. Com o poder da linguagem, vem, a **responsabilidade**. Você pode fazer com C coisas que **destruirão** seu programa ou **afetarão** o funcionamento do computador. Mas, desta vez, os programadores encontraram uma linguagem que **é uma ferramenta** e não um obstáculo.

O padrão ANSI

Esta popularidade fez com que vários compiladores de C surgissem no mercado, cada um caminhando em uma direção diferente. Isto poderia detonar a linguagem. Para evitar isto, O American National Standards Institute (ANSI) formou uma subcomissão especial chamada X3J11 para elaborar uma versão do C. Esse desenvolvimento garantiu que aquela linguagem que estava se perdendo em várias direções diferentes ao mesmo tempo, se tornasse padronizada e coerente novamente.

O objetivo final do ANSI era padronizar C para todos os tipos de computadores. Mas, existem muitas diferenças entre os vários tipos de computadores. Por exemplo, a maneira como cada uma trata a memória. Assim, os compiladores incorporam todos os padrões do C ANSI, mas se diferenciam nas particularidades de cada computador para os quais foram desenvolvidos. Um exemplo é a manipulação da tela gráfica que é muito diferente entre um PC, um computador com plataforma RISC.

O surgimento de C++

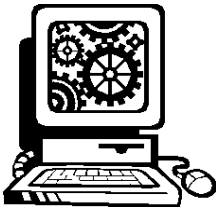
Com a popularidade de C, cada vez mais aplicativos eram escritos em C, e cada vez mais complexos ficam os aplicativos. Percebeu-se que as construções lógicas disponíveis em C não correspondiam mais as necessidades. Era preciso construções que permitissem particionar o programa cada vez mais para ficar fácil de escrever programas complexos, livrar o programador da necessidade de controlar centenas de variáveis e funções. Nesta época a idéia de objeto já começava a se destacar. Já existiam linguagens que permitiam uma programação orientada a objetos. Assim, em 1983, **Bjarne Stroustrup** apresentou a **linguagem C++**. Em essência C++ é muito semelhante a C. Ou melhor, **C é um subconjunto de C++**. Tudo que se faz em C pode-se fazer em C++. Foi incorporado à linguagem o potencial da programação orientada a objetos, introduzindo as **classes**. Mas não é só isto. Em C++ pode-se, também, fazer o que se chama de **sobrecarga de funções**. Ou seja, pode-se fazer com que uma mesma função funcione de maneiras diferentes de acordo com o número de parâmetros que ela recebe e/ou o tipo de parâmetro, pode-se também **sobrecarregar os operadores**. Por exemplo, fazer com que o operador && (E lógico) tenha outra função além da de fazer a operação E.

Mas, sem dúvida nenhuma a orientação a objetos é o grande trunfo de C++. Ela permitiu que os programas fossem modularizados de maneira estruturada e natural. Diz-se que programar com objetos é **imitar a vida**. Isto pode parecer um exagero. Mas, se você se dedicar, poderá verificar se é isso mesmo.

Introdução

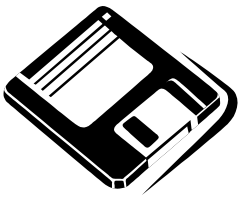
Os computadores **não** são capazes de entender as instruções e comandos das linguagens utilizadas para programá-los. Eles só entendem a **linguagem de máquina** (0s e 1s). Portanto, para que seja possível escrever programas em linguagens do tipo BASIC, Pascal, C, C++ etc, é necessário, depois de escrito os programas, chamados **programas-fonte**, transformá-los em linguagem de máquina. E além disto, adaptar o programa para ser executado no hardware do computador, já que eles podem ter várias configurações.

A transformação do programa escrito nestas linguagens para o programa escrito em linguagem de máquina escrito nestas linguagens para o programa escrito em linguagem de máquina é chamada de **compilação** e é feito por um programa chamado **compilador.** O compilador gera um arquivo que possui os códigos das instruções do programa-fonte, escritos em linguagem de máquinas. Este arquivo é chamado de **arquivo-objeto**. Pode-se, então, esquematizar o processo da seguinte forma:



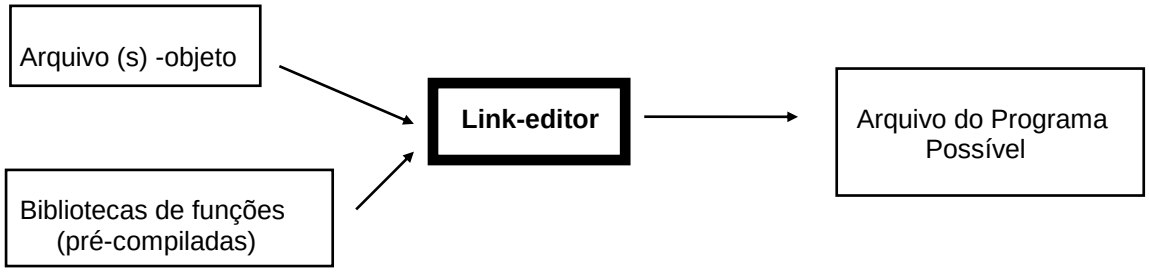
Programa editado em linguagem C++

Programa Compilador



arquivo-objeto

Após a compilação, pode ser necessário acrescentar ao programa compilado outros arquivos-objeto (por exemplo, em projetos feitos de forma particionada, onde cada equipe de desenvolvimento desenvolve uma parte do aplicativo). Ou até, acrescentar bibliotecas de funções que foram utilizadas no programa-fonte. Só depois disto é que é possível gerar o **arquivo executável**. Este processo de **"agrupar"** arquivos-objeto e gerar o arquivo executável é chamado de **link-edição** e é feito por um programa chamado **link-editor**. Pode-se, então, esquematizar o processo da seguinte forma:



Explicação ao treinando

Nosso curso é sobre a linguagem C++. A linguagem C é um subconjunto de C++. É de se esperar, então, que um programador que saiba programar em C++, saiba também programar em C. Isto é verdade. Só que em linguagem C ou C++ a quantidade de funções e objetos disponíveis fazem com que isto não seja verdade em todas as situações. Estudaremos toda a base de programação em C++ que é a mesma de C. Mas, em alguns casos, usaremos construções lógicas e principalmente objetos que são só possíveis em C++.

Portanto, pode ocorrer de você se deparar com um programa em C e não entender algumas funções. Mas, com certeza, uma consulta no sistema de help do compilador ou nos manuais do fabricante do compilador resolverá este problema.

Daremos ênfase a notação da linguagem C++ em detrimento da C.

Esperamos com isto torná-lo familiarizado com o uso dos objetos que esta linguagem nos oferece, que permitem, na maioria dos casos, construções robustas e seguras. E lembre-se, em C ou C++ construções seguras são essenciais, já que você mesmo pode destruir seu programa.

Neste primeiro módulo do curso, a diferença principal está na forma de imprimir no vídeo e ler dados do teclado. Em C utiliza-se muito as funções **fprintf (), fscanf ()** e suas semelhantes e derivadas. Em C++ utiliza-se muito os objetos **cout** e **cin**, o que simplifica os programas e leva todos os detalhes para dentro destes objetos, os quais ficam escondidos do programador (para sorte dele!!?).

A partir de agora vamos decolar!!!

Tipos de dados

Apesar de, a princípio, um computador só entender 0s e 1s, é possível utilizando-se alguns controles, fazer com que um computador manipule vários tipos de informação. Na verdade, o compilador esconde do programador as dificuldades de manipulação do formato dos dados, permitindo que o programador se preocupe com o processamento do dado propriamente dito. Em C++ pode-se utilizar vários tipos de dados:

Numérico
Podem ser inteiros ou reais (ponto flutuante) ou reais de dupla precisão.

Literais
Estão divididos em dois tipos: os caracteres e as strings.

Strings:
É uma sequência de caracteres contendo letras, dígitos e/ou símbolos especiais que fazem parte da tabela ASCIL. É também como dado alfanumérico ou cadeia de caracteres. Normalmente, nos algoritmos e nas linguagens de programação, uma string é delimitada no início e no fim por um caracter aspas (“). Diz-se que uma string possui um comprimento dado pelo número de caracteres nela contido. Exemplos:

“Qual? string de comprimento 6
“” string de comprimento 1
“qUaL ?! @#” string de comprimento 9
“123” string de comprimento 3
“” string de comprimento zero

OBS. Espaço também é um caractere.

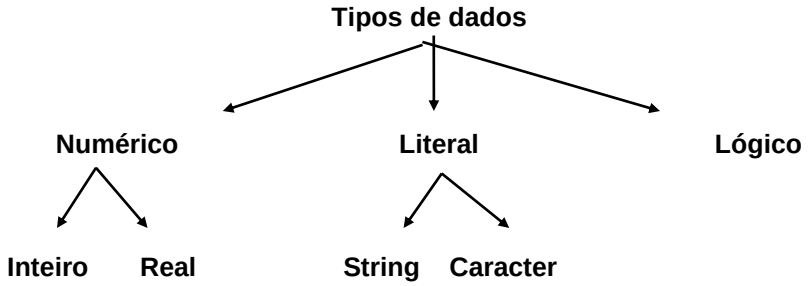
Caracteres:
São diferentes das strings por possuir apenas um caractere e são tratados pelas linguagens de maneira diferente de uma string de comprimento 1. Para diferenciar os dois utiliza-se, no caso dos caracteres, o símbolo apóstrofo (‘) para identificar um caractere. Exemplos:

‘ A ‘ ‘ a ‘ ‘ 1 ‘ ‘ @ ‘

Lógicos

Este tipo de dado está intimamente ligado com o funcionamento dos computadores. São também chamados de dados booleanos devido as contribuições de George Boole à área da lógica matemática. Este tipo de dado pode assumir apenas dois valores que são representados por verdadeiro ou falso (true ou false), sem ou não, 1 ou 0.

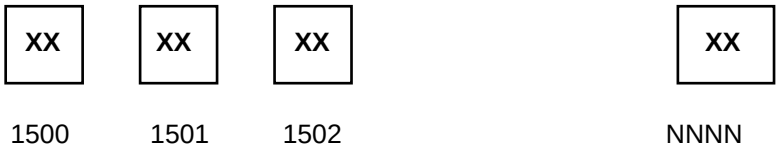
Resumo



Conceito de variável

A memória de um computador

Pode-se entender a memória de um computador como se fosse uma pilha de caixas onde em cada caixa é guardada uma informação (instrução ou dado). Para que o computador possa tratar estas informações sem cometer erros, é preciso que cada caixa tenha um endereço. Nos computadores os endereços são números. Nos computadores atuais cada informação é codificada por um número binário de 89 dígitos (BYTE. Assim, a memória dos computadores está organizada em bytes).



Variáveis

Variáveis são posições de memórias nas quais programas podem armazenar (ler e/ou escrever) valores importantes, ou não, durante a execução de um programa. Como estes valores, normalmente, variam durante a execução do programa, daí o nome variável.

Para acessar individualmente cada uma destas informações, é necessário saber o tipo de dado e o endereço da mesma. Imagine um programador tendo de memorizar o endereço das dezenas de variáveis que seu programa usa, ou tendo de consultar uma tabelinha escrita para saber os endereços. Percebe-se que esta sistemática de acesso é difícil de trabalhar. Para contornar isto pode-se associar a estas variáveis um nome, uma etiqueta, um apelido que, facilmente, é chamado de nome de variável.

Portanto, uma variável possui 3 atributos: um **nome**, um **tipo** de dado que ela armazena é o **conteúdo** propriamente dito. Todas as variáveis possuem um nome e em cada linguagem de programação existem regras para nomeá-las. Por exemplo, em C++ tem-se as seguintes **regras (práticas) básicas:**

- O nome de uma variável deve começar com letra ou sublinhado (_);
- O nome de uma variável não deve conter símbolos especiais, exceto o sublinhado;
- O nome de uma variável deve lembrar o que ela guarda;
- O compilador diferencia letras maiúsculas das minúsculas (nome é diferente de Nome).

Exemplos

n1, n2, N1, N2 correto
1ano, 2ano incorreto
nome 1, nome 2 incorreto
nome-1, nome-2 correto
media, nota, recup correto
a*b, $c, dado!, ac@et incorreto

Declarar é preciso

Em C++ é preciso declarar uma variável antes de utilizá-la. Declarar uma variável é informar ao compilador de que tipo é a variável, seu nome e quando necessário se ela tem um valor inicial. Esta declaração é utilizada pelo compilador para reservar na memória do computador o espaço necessário para armazenar os valores desta variável. A declaração de variáveis é um item importante nos programas e deve ser feita com muito cuidado e critério.

Tabela de tipos básicos

A linguagem C++ possui um grande número de tipos e para cada tipo é utilizada uma palavra para declará-lo. Esta palavra é que indica ao compilador o tipo de dado que será utilizado.

Nome do tipo	tipo	Bytes ocupados na memória	Faixa de valores
char	caracter	01	- 128 a + 127
int	inteiro	02	- 32768 a + 32767
float	ponto flutuante	04	$\pm$ 3.4 x 10^{-38} a $\pm$ 3.4 x 10^{+38}
double	dupla precisão	08	$\pm$ 1.7 x 10^{-308} a $\pm$ 1.7 x 10^{+308}
void	vazia	00	----------

Modificadores de tipo

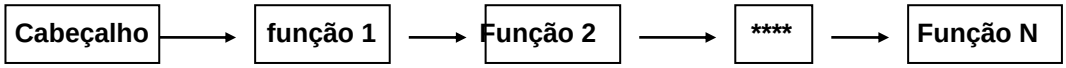
É possível, acrescentando-se palavras chaves à frente dos nomes dos tipos, modificar suas faixas de valores. Os modificadores de tipo são: long, short e unsigned. Assim a tabela fica:

Modificadores de tipo	Nome do tipo	Tipo	Bytes	Faixa de valores
	char	caracter	01	-128 a +127
unsigned	char	caracter	01	0 a 255
	int	inteiro	02	-32768 a +32767
long	[int]	inteiro longo	04	-2147483648 a +2147483647
unsigned	[int]	inteiro sem sinal	02	0 a 65535
unsigned long	[int]	longo sem sinal	04	0 a 429467295
short	[int]	inteiro curto	02	-32768 a +32767
	float	ponto flutuante	04	$\pm$ 3.4 x 10^{-38} a $\pm$ 3.4 x 10^{+38}
	double	dupla precisão	08	$\pm$ 1.7 x 10^{-308} a $\pm$ 1.7 x 10^{+308}
long	double	dupla precisão	10	$\pm$ 3.4 x 10^{-4932} a $\pm$ 3.4 x 10^{+4932}
	void	vazia	00	----------

OBS:. [ ] significa opcional.

Estrutura básica de um programa

Diz-se que um programa em linguagem C++ é uma parede construída com vários tijolos. Onde, a parede é o programa aplicativo e os tijolos são as funções. Assim, as funções são à base da linguagem C++. Um programa simples em linguagem C++ é 90% um conjunto de funções. Algumas fornecidas pelo fabricante do compilador/link-editor e outras escritas pelo próprio programador. As funções fornecidas pelo fabricante do compilador/link-editor podem ser utilizadas como uma "caixa preta". Enquanto que as escritas pelo programador devem ser desenvolvidas dentro do corpo do programa. Em C++ o nome função se confunde com o nome programa. Já que um programa em C++ terá pelo menos uma função.



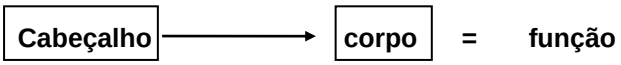
Cabeçalho

É composto basicamente por:

- Identificação do programa;
- Diretivas para o pré-processador (# inclusive, # define etc);
- Declaração de variáveis globais;
- Protótipo das funções;
- Declaração de classes.

Funções

Como dissemos, são as bases dos programas. Possuem um header (cabeçalho) e um body (corpo).



No cabeçalho, temos a definição do tipo de função (podem ser do mesmo tipo que as variáveis), seu nome e seus parâmetros. No corpo temos, a declaração de variáveis locais e seus comandos. Pode-se também, a partir de uma função, fazer uma chamada à outra função.

Como as funções são a base, começaremos por elas!!

Sintaxe das funções

As funções em C++ têm o seguinte aspecto em termos de sintaxe:

```
tipo_do_valor_de_retorno nome (tipo_dos_parâmetros_formais parâmetros_formais)
{
declaração de variáveis locais
    ...
    corpo da função (comandos e outras funções);
    ...
```

Exemplo

(não se assuste!)

	tipo do valor de retorno
	nome da função
float divisão (float n1, float n2	*tipo e nome dos parâmetros formais*
{	*chave de abertura*
float res;	*declaração de variáveis locais*
res = n1/n2;	*corpo da função*
return (res);	
}	*chave de fechamento;*

Detalhes

- O nome de uma função segue as mesmas regras práticas para os nomes das variáveis;
- O que diferencia as funções de variáveis são os parênteses. Aliás, isto as identifica;
- Os parâmetros formais são as variáveis receptoras dos parâmetros passados para as funções;
- As variáveis declaradas dentro das funções são locais, ou seja, só são visíveis dentro da função;
- O corpo da função é composto por qualquer comando da linguagem C++, por chamadas às outras funções, por instanciamentos de classes etc;
- Uma função em C++ pode retornar um valor para a função chamadora. Isto é feito através do comando return;
- Se uma função não vai retornar nenhum valor dizemos que ela é vazia e deve-se declará-la como do tipo void;

De todas as características de uma função, apenas três são obrigatórias:

- O nome seguido de parênteses;
- As chaves de abertura e fechamento;
- E seu protótipo no cabeçalho do programa.

OBS. - Iremos no começo usar as funções simples, sem todas as suas características.

- Em alguns compiladores não é obrigatório a prototipagem das funções.

Outro detalhe é que uma função é obrigatória em todos os programas em C++. Esta função se chama ***main()***. Ela é o ponto inicial do programa e ***não precisa de protótipo***. Assim, um programa mínimo em C++ seria:

```
main
{
}
```

ou

```
void main()
{
}
```

Apesar deste programa não fazer nada, ele é um programa válido.

Uma característica importante na sintaxe das funções é que a posição dos elementos, na maioria dos casos, não é importante, mas sim, a sequência com que eles aparecem.

Portanto, todos os programas a seguir são válidos:

main
(
)
{
}
	main(){}	main()
		{}
			main ( )
			{
			}

E, os a seguir são inválidos:

()main{}	main{} ()

OBS. A melhor opção é aquela que garante uma boa legibilidade do programa.

Ambiente Integrado de Desenvolvimento

O IDE (Integrated Developement Enviroment) da empresa BORLAND é um ambiente integrado que possui várias ferramentas que facilitam a vida do programador. Neste ambiente pode-se digitar o programa-fonte em um editor de textos, gravar ou carregar programas em disco, compilar o programa, verificar se ele possui erros de sintaxe e outros erros, link-editar o programa e executar o programa para verificar se o mesmo funciona corretamente.

Além disto, ele possui ferramentas de depuração como: execução passo-a-passo, break-points etc.

É um ambiente totalmente configurável, inclusive em determinados detalhes técnicos, como a possibilidade de compilar o programa usando um compilador C ou C++, o tipo de modelo de memórias a ser utilizado etc.

Os primeiros passos

Alguns detalhes precisam ser estudados antes de começarmos a escrever nossos programas que fazem alguma coisa.

Vale repetir que um programa C++ é composto por várias funções, declaração de classes, instancias de classes (objetos) etc. Os fabricantes de compiladores para C++ oferecem aos programadores uma vasta gama de funções, classes e objetos prontos, não sendo necessários desenvolvê-los. Isto facilita enormemente o trabalho e permite ao programador criar funções, classes e objetos mais complexos, utilizando estes, que já estão prontos, como base. Uma parte destes elementos prontos faz parte do conjunto de elementos do padrão ANSI e, portanto, são oferecidos por praticamente todos os compiladores. E outros são específicos de cada fabricante.

Um detalhe importante na sintaxe das funções em C++ é que é obrigatória a prototipagem das funções, prototipar uma função é declará-la ao compilador. No protótipo de uma função declara-se de que tipo é a função (valor de retorno) e de que tipo são seus parâmetros formais. Assim, no cabeçalho dos programas devemos ter os protótipos das funções.

Neste momento pode surgir uma dúvida. Como fazer o protótipo das funções prontas fornecidas pelo fabricante do compilador?

Para evitar que o programa tenha muito trabalho, afinal o objetivo é o oposto, os fabricantes oferecem junto com o compilador, arquivos contendo o protótipo das

funções que ele oferece ao programador. Estes arquivos são chamados de **hearders** e devem ser **incluídos** no seu programa caso você utilize funções prontas nele. Como é praticamente impossível não as usar, todo programa vai ter estes arquivos incluídos. Os protótipos estão divididos por tipo de função. Então é necessário incluir um ou mais arquivos headers em seus programas, dependendo das funções utilizadas. Para incluir um arquivo em um programa, utiliza-se uma **diretiva** para o pré-processador chamada **#include**. Sua sintaxe é a seguinte:

#include <nome do arquivo> ou #include "path:\nome do arquivo"

Na 1ª opção o compilador procura pelo arquivo em um diretório padrão. Normalmente o diretório é path:\include. Na 2ª opção o compilador procura pelo arquivo no caminho especificado.

Além dos protótipos das funções, os fabricantes colocam nos arquivos headers algumas definições de constantes de uso comum e até mesmo definições de classes e objetos. A seguir, estão alguns dos headers mais comuns.

cstdio cmath iostream cstdlib.h iomanip

OBS:. a diretiva #include não serve apenas para os arquivos headers. Pode-se incluir qualquer tipo de arquivo no seu programa, deste de que ele tenha uma sintaxe que o compilador entenda. Perceba que é possível incluir programas-fonte dentro de outro programa-fonte.

Exemplo

```
#include <iostream>
#include<cstdlib>
using namespace std;
 main()
{
        system("CLS"); // funcao que limpa a tela do DOS (clear screen)
        ----
}
```

Ainda os primeiros passos

Comentários

Uma parte importante nos programas em C++ é a colocação de linhas de comentários nos blocos lógicos do programa para explicar para outro programador que porventura possa vir a ler seu programa, como aquele bloco funciona. Isto serve também para documentar o programa.

São possíveis duas sintaxes para os comentários.

Na primeira, chamados comentários de linha, utilizam-se duas barras seguidas, //, para indicar que o que está depois destas barras até o final da linha, é apenas um comentário e deve ser desprezado pelo compilador.

Na segunda forma os comentários devem ficar dentro de dois conjuntos de símbolos que indicam o início e o fim do comentário. Os símbolos são: /* e */.

OBS:. Não se deve aninhar comentário, ou seja, colocar um comentário dentro de outro.

Exemplos:

```
//      EXEMPLO 000.0
// Programa que mostra o uso de comentário de linha
// Curso C++ - Professor Rene
#include <iostream >
using namespace std;

main()
{
}


/*      EXEMPLO 000.1
Exemplo do uso de comentários em diversas linhas
Prof. Rene
01/01/99
*/
#include <iostream>          /* arquivo de prototipos */
#include <cstdlib>
using namespace std;

main()                    // funcao principal
{
}
```

Portabilidade

Os criadores da linguagem C estavam em busca de um a linguagem que fosse **portável**, já que os sistemas computacionais estavam se multiplicando.

Uma linguagem portável é aquela que permite que se escrevam programas que podem ser executados em vários tipos de computadores com as mais diversas plataformas de hardware e software. Ou seja, o mesmo programa em máquinas diferentes. Como isto poderia ser possível?

Na verdade, o que é **portável é o programa-fonte** escrito em linguagem C e não programa executável.

A dificuldade de tornar um código portátil é que cada máquina tem seus próprios sistemas de acesso ao hardware interno e aos periféricos. Então, como compatibilizar isto?

A idéia é simples, mas muito inteligente.

Esconde-se do programador, ou seja, da linguagem, o acesso ao hardware. As instruções que controlam o microprocessador e o hardware básico são colocadas em bibliotecas escritas em código de máquina e para cada tipo de hardware, compra-se o compilador e as bibliotecas adequadas. Assim, o mesmo programa fonte, compilado com o compilador adequado, pode ser executado em vários tipos de máquinas.

Mas, e os periféricos?

O acesso aos periféricos: vídeo, teclado, unidade de disco etc. é muito interessante. Foram criados periféricos **"virtuais**" chamados **stream**. A palavra stream significa uma **fila de dados**. O programador, quando quer escrever ou ler um destes periféricos, faz isto numa stream que durante a compilação do programa vai ser **"ligada"** ao periférico específico e os dados da fila enviados ao periférico ou

recebidos dele. Foram definidas algumas streams padrões que estão associadas aos periféricos mais comuns.

Periférico	stream padrão em C
vídeo	stdout
teclado	stdin
impressora	stdpm
video	stdaux
video	stderr

Toda esta filosofia foi observada pela linguagem C++ e além destas streams padrões foram acrescentadas outras, só que na forma de **objetos de stream**.

Periférico	stream padrão em C	objetos stream em C++
Video	stdout	cout
Teclado	stdin	cin
Impressora	stdprn	
Video	stdaux	
Video	stderr	cerr

A partir de agora, os segundos passos:

Impressão em streams de saída

Função fprint() - permite a impressão formatada em qualquer stream
fprintf(stream, "string a ser impressa mais caracteres de controle", lista de parâmetros);

printf() - imprime vários tipos de dados no vídeo (stdout);
printf("string a ser impressa mais caracteres de controle", lista de parâmetros);

putchar() - imprime um caracter da tabela ASCII no vídeo;
putchar(caracter);

putch() - imprime um caracter no vídeo;
putch(caracter);

cout - objeto de stream que permite a impressão de qualquer tipo de variável no vídeo;
cout<< "string de controle" << parâmetros;

Os caracteres de controle mais comuns que podem ser utilizados dentro da string de controle são:

\a	alerta (emite um beep)
\n	nova linha (envia o cursor para a próxima linha margem esquerda)
\b	Retrocesso
\r	carriage return eqüivale a tecla ENTER
\f	Alimentação de formulário (impressora)
\t	Tabulação horizontal
\v	Tabulação vertical
****	Imprime barra invertida
\'	Imprime apóstrofo
\"	Imprime aspas
\0 (\zero)	Nulo – usado para finalizar strings

Os principais caracteres de controle de formatação de impressão são:

%d ou %i	Imprime um parâmetro no formato inteiro (decimal)
%c	Imprime um parâmetro no formato caracter (tabela ASCII)
%f	Imprime um parâmetro no formato ponto flutuante
%s	Imprime um parâmetro no formato string

```
//      EXEMPLO 001
//      Uso da funcao fprintf() para imprimir no video

#include <cstdlib >
#include <iostream> // prototipo da funcao clrscr() - clear screen
using namespace std;

main()
{
        system("CLS");
        fprintf(stdout, "Meu primeiro programa que faz alguma coisa");
}
```

```
//      EXEMPLO 002
//      Uso da funcao fprintf() para imprimir na impressora

#include <cstdlib>
#include <iostream>
using namespace std;
oid main()
{
        system("CLS");
        fprintf(stdprn,"Meu segundo programa que faz alguma coisa");
}
```

```
//      EXEMPLO 002A
//      Uso da funcao printf() para imprimir no video

#include <cstdlib>
#include <iostream> //
using namespace std;

main()
{
        system("CLS");
        printf("Meu nome começa com %c", 'R');
}
```

```
//      EXEMPLO 002B
        //      Uso da funcao printf() com parametros
        using namespace std;

        #include <cstdlib>
        #include<iostream>
        using namesapce std;
        main()
        {
                system("CLS");
                printf("Eu tenho %d anos \n e me chamo %s", 34, "Jose");
        }
```

Iremos por questões de robustez, segurança, facilidade no uso, melhor tratamento das variáveis etc, utilizar para imprimir no vídeo o objeto de stream cout em todos os programas e exemplos.

Exercício 01

O objeto de stream cout

cout << "string de controle" << parâmetros;

- << operador de inserção, utilizado para conectar a string de controle ao objeto cout;
- string de controle mostra a cout o que e como imprimir;
- parâmetros valores a serem impressos.

A definição do objeto cout está no **header iostream.h.**

```
//      EXEMPLO 003
//      Uso do objeto cout
#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
        system("CLS");
        cout << "Meu primeiro programa verdadeiramente em C++";
}
```

Observe:

- As diretivas #include;
- a string de controle não possui caracteres de controle, apenas a string a ser impressa.

```
//   EXEMPLO 004
//   Uso do objeto cout caracteres de controle

#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
        system("CLS");
        cout << "Meu nome e:\n";
        cout << "Rene Graminhani\n";
        cout << "0\t\a1\t\a2\t\a3\t\a4\t\a5\t\a6\a";
}
```

```
//      EXEMPLO 005
//      Uso do objeto cout com multiplas strings

#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
        system("CLS");
        cout << "Meu nome e:\n" << "Rene\a\a\a\n";
        cout << "Uso" << "de" << "multiplas" << "strings";
        cout << "\nimpressao de constantes numericas:" << 10.293;
}
```

Para evitar o uso de vários cout, em cada linha, pode-se usar apenas um e escrever as strings em linhas diferentes. Só que não se pode colocar ";" (ponto e vírgula) na linha anterior.

```
//      EXEMPLO 006
//      Uso de um objeto cout com multiplas strings
#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
        system("CLS");
        cout << "Meu nome e:\n" << "Rene \a\n"
             << "Uso" << " de" << "multiplas" << " strings\n\n"
             << "Impressao de constantes numericas: " << 10.293
             << "\nImpressao tabulada: "
              << "\n0\t1\t2\t3\t4\t5\t6";
}
```

Exercício 02

Leitura do teclado com cin e imprimindo variáveis

cin >> nome da variável que armazenará o valor lido;

A definição do objeto cin está no header **iostream.h.**

Como se observa, quando um dado é lido do teclado é necessário armazená-lo em uma variável. E como já foi estudado, antes de usar uma variável é preciso declará-lo. O C++ permite que uma variável seja declarada em qualquer lugar do programa, desde que antes de usá-la. Mas, por questões de estruturação e de legibilidade iremos declarar; pelo menos neste início, as variáveis sempre no início das funções, antes do corpo da mesma.

Declarar uma variável é atribuir a ela um nome, um tipo e se for necessário um valor inicial.

```
//      Exemplo 007
//      Uso de cin e declaração de variáveis
//      Pode-se utilizar caracteres acentuados e o cecidilha apenas
//      dentro das strings de controle e nos comentarios.
//      Não e possível usá-los em nome de variáveis, funções ou classes e
        objetos

#include <iostream >
#include <cstdlib>

using namespace std;
main()
{
int idade;
          system("CLS");
          cout << "Digite sua idade:";
          cin >> idade;
          cout << "\nVoce ainda e bem novo(a), tem apenas " <<idade<< " anos";
}
```

```
//      EXEMPLO 008
//      Uso de cout e declaração de variáveis locais

#include <iostream >
#include <cstdlib>

using namespace std;
main()
{
float pi;
          system("CLS");
          cout << "Digite o valor de pi:";
          cin >> pi;
          cout << "O valor de pi e" << pi;
}
```

Múltiplas entradas com cin

Os valores devem ser digitados separados por espaço

```
//      EXEMPLO 009
//      Uso de cin com múltiplas entradas
#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
int idmae,idpai;

        system("CLS");
        cout << "Digite a idade de seu pai e de sua mãe:";
        cin >> idpai >> idmãe;
        cout << "\nSeu pai e sua mãe tem juntos " << idpai+idmãe << " anos";
}
```

O C++ calcula o valor de idpai + idmae e envia para cout o resultado, que será impresso no vídeo.

Exercício 03

Operadores aritméticos

Operador	Ação
=	Atribuição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo da divisão ou resto
++	Incremento
- -	Decremento

- **Soma, subtração, multiplicação e divisão:**
  Funcionam como na matemática;

- **Atribuição**
  Atribui o valor da expressão que está do lado direito para a expressão que está do lado esquerdo
  Exemplo: a = b + c - > primeiro calcula o resultado da soma para depois atribuí-lo a variável a

- **Módulo**
  Retorna o resto da divisão
  Exemplo: div = 10 % 3 -- > primeiro faz a divisão depois atribui o resto
  (1) a div. Observe que a divisão é inteira, senão não haveria resto.

- **Incremento e decremento**
  Incrementam ou decrementam a variável que é seu operando. Podem ser pré-fixados ou pós-fixados.

  Exemplos: maças = 4
  Frutas = maças ++ (pós-fixado) - primeiro atribui 4 a frutas depois incrementa maças

  Maças = 4
  Frutas = ++maças - primeiro incrementa maças e depois atribui maças (5) a frutas

```
//      EXEMPLO 010
//      Uso dos operadores * e  /
#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
float idade, dias, horas, minutos;
        system("CLS");
        cout << "Digite sua idade:";
        cin >> idade;
        dias = idade * 365;
        horas = dias * 24;
        minutos = horas * 60;
        cout << "\nSua idade em dias e "<< dias
                << "\n              em horas e "<< horas
                << "\n               em minutos e " << minutos
                << "\n\n Em Venus voce teria " << idade/0.8 << " anos venusianos";
        system("PAUSE");
}
```

```
//        EXEMPLO 011
//        Uso do operador %
#include <iostream >
#include <cstdlib>

using namespace std;
main()
{
int amigos, laranjas, cadaum, restam;
        system("CLS");
        cout << "Digite o numero de laranjas que estao dentro da cesta:";
        cin >> laranjas;
        cout << "\nDigite o numero de amigos comiloes que voce tem:";
        cin >> amigos;
        restam = laranjas%amigos;
        cadaum = (laranjas - restam)/amigos;
        cout << "\nCada amigo seu recebera"<< cadaum << " laranjas"
              << "\ne sobrara(m)" << restam << "laranja(s) no cesto";
        system("PAUSE");
}
```

```
//    EXEMPLO 012
// Uso dos operadores ++ e –
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
int peras, frutas;
        system("CLS");
        cout << "Digite o número de peras que estao dentro da cesta: ";
        cin >> peras;
        cout << "peras =" << peras;
        cout << "\n ++peras = " <<++peras;
        cout <<"\n--peras = "<< --peras;
        cout << "\nperas ++ = "<<peras++;
        cout << "\nperas -- ="<<peras--;
}
```

Manipuladores de formação de impressão

Manipulador	Ação
setw(int n)	Fixa em n, o número de casas a serem utilizadas para a impressão do próximo dado
setprecision(int p)	Fixa em p, o número de casas decimais do próximo dado a ser impresso (float)
setfill(char ch)	Fixa em ch, o caractere de preenchimento das colunas em branco do próximo número a ser impresso

OBS:. Os manipuladores estão definidos no arquivo **header iomanip.h**

```
//    EXEMPLO 013
//  Uso do manipulador setw()
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
main ()
{
int jose, pedro, luiz, marcio;
        system("CLS");
        cout << "Notas (0 a 100) para os alunos Jose, Pedro, Luiz e Marcio: ";
        cin >> jose >> pedro >> luiz >> marcio;
        cout << "\n\n\n"
        << "\nJose     " << setw(12) << jose
        << "\nPedro    " << setw(12) << pedro
        << "\nLuiz     " << setw(12) << luiz
        << "\nMarcio   " << setw(12) << marcio;
        system("PAUSE");
}
```

```
//      EXEMPLO 014
//      Uso do manipulador setprecision () e setfill()
#include <iostream>
#include <iomanip >
#include <cstdlib>
using namespace std;
main()
{
float video,teclado,mouse,cpu,winchester;
        system("CLS");
        cout <<"\nDigite o preço do video:"; cin >> video;
        cout <<"\nDigite o preço do teclado:";cin >> teclado;
        cout <<"\nDigite o preço do mouse:"; cin >> mouse;
        cout <<"\nDigite o preço da CPU:"; cin >> cpu;
        cout <<"\nDigite o preço do winchester:"; cin >> winchester;
        cout <<"\nVideo       "<<setw(12)<<setprecision(2)<<setfill('.') << video
             <<"\nTeclado     "<<setw(12)<<setprecision(2)<<setfill('.') << teclado
             <<"\nMouse       "<<setw(12)<<setprecision(2)<<setfill('.') << mouse
             <<"\nCPU         "<<setw(12)<<setprecision(2)<<setfill('.') << cpu
             <<"\nWinchester "<<setw(12)<<setprecision(2)<<setfill('.')<< winchester;
        system("PAUSE");
}
```

Manipuladores de base numérica

Manipulador	Ação
dec	Imprime o próximo valor no formato decimal (default)
hex	Imprime o próximo valor no formato hexadecimal
oct	Imprime o próximo valor no formato octal

```
//    EXEMPLO 015
//     Uso dos manipuladores de base numérica
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
main()
{
int numero;
        system("CLS");
        cout << "Digite um numero inteiro: ";
        cin >> numero;
        cout << "\n\n";
        cout << "\nNumero em decimal: "<< numero;
        cout << "\nNumero em hexadecimal: " << hex << numero;
        cout << "\nNumero em octal: "<< oct << numero;
        system("PAUSE");
}
```

Exercício 04

Outros detalhes

Três outras características devem ser estudadas antes de prosseguirmos.

Declaração de constantes

É muito comum o uso de constantes em programas. Constantes numéricas, caracteres ou até mesmo constantes do tipo string (mensagens de erro, por exemplo).

A C++ oferece duas formas de definir constantes: usando a diretiva para o pré-processador #define ou a palavra chave const (qualifier). Usaremos a palavra chave const por ela permitir que se declare o tipo de constante, fazendo com que desta forma o compilador trabalhe melhor com estes tipos de dados na memória. Na diretiva #define não se especifica o tipo de constante, apenas seu valor.

```
//        EXEMPLO 016
          #include <iostream>
          #include <cstdlib>
          const float pi = 3.141592;
          const char alerta = '\a';

          using namespace std;
          main()
          {
          float raio;
                    system("CLS");
                    cout << "Digite o raio da circunferencia em milimetros: ";
                    cin >> raio;
                    cout << "\n\n";
                    cout << "\nO perimetro vale"<< 2*pi*raio <<"mm "<< alerta;
                    cout << "\nA area do circulo equivalente vale "<< pi*raio*raio <<
"mm2";
                    system("PAUSE");
          }
```

Conversão forçada de tipo usando cast (molde)

Em determinadas situações é desejável imprimir um dado em um formato diferente do que foi especificado na declaração de variáveis. Para isto forçamos o formato da impressão deste dado.

```
//   EXEMPLO 017
//
#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
char letra;
          system("CLS");
          cout << "Digite uma letra da tabela ASCII: ";
          cin >> letra;
          cout << "\n\n";
          cout << "Seu equivalente numerico e "<< int(letra);
          system("PAUSE");
}
```

Pode-se forçar uma impressão para qualquer tipo válido da linguagem C++: int, char, float etc. Basta para isto colocar o tipo desejado na frente da variável entre parênteses.

Comandos de controle de fluxo

Os comandos de controle de fluxo permitem que se implemente as diversas estruturas lógicas da linguagem C++. O estudo destes comandos será acumulativo. Uma vez aprendido um comando ele será utilizado nos programas seguintes. Assim, os programas irão ficando cada vez mais complexos.

Comandos de decisão

Os comandos de decisão são um dos principais tipos de comandos de controle de fluxo. Eles podem desviar o fluxo do programa a partir do resultado de um teste realizado no comando. Se este teste resultar verdadeiro o programa executa uma tarefa, se resultar falso executa outra. Para realizar os testes de decisão são utilizados os operadores relacionais.

Operadores relacionais

Operador	Teste
>	É maior que
>=	É maior ou igual a
<	É menor que
<=	É menor ou igual a
==	É igual a
!=	É diferente de

Verdadeiro ou falso?

Os operadores relacionais funcionam de forma semelhante aos operadores aritméticos, só que eles não devolvem o resultado da operação e sim a resposta de uma comparação. Esta resposta pode ter dois valores: verdadeiro ou falso, mas como será que o computador processa as palavras verdadeiro e falso? Vejamos o seguinte exemplo:

```
//      EXEMPLO 019
#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
int a=10,b=2,c;
        clrscr();
        cout << "\nO resultado da comparacao 10 < 2 e" << (a<b); //parenteses obrig.
        c=30<50;
        cout << "\nO resultado da comparação 30<50 e "<< c;
        system("PAUSE");
}
```

Então o computador representa verdadeiro como 1 e falso como 0.

Lembre-se, o computador realiza as operações pedidas, aritméticas ou relacionais, e coloca no lugar da operação o seu resultado. É por isso que podemos mandar cout imprimir 10>2, porque o computador realiza a operação e coloca no seu lugar o resultado 1 e cout imprime este 1.

Vejamos um exemplo mais complicado:

```
//        EXEMPLO 020
//
#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
int a,b;
          system("CLS");
          cout << "Digite os valores de a e b: ";
          cin >> a >> b;
          cout << "\n\n";
          cout << "O resultado de a+23<4-b e "<< (a+23<4-b);
          system("PAUSE");
}
```

Em primeiro lugar, que confusão, misturar operadores aritméticos com relacionais?

Isto é perfeitamente possível em C++. Mas, a dúvida é outra. Qual a seqüência de operações que o computador realiza para determinar o resultado. Primeiro a soma, depois o menor e depois a subtração? Ou será que é ao contrário?

Bom, quando desejamos que as operações sejam feitas em uma determinada seqüência deve, se for o caso, forçar a seqüência. Mas, existe uma hierarquia entre os operadores e o computador respeita esta hierarquia. Esta hierarquia é chamada de precedência de operadores.

Operadores em ordem decrescente de hierarquia (precedência)
*
/
%
+
-
<
<=
>
>=
==
!=

Assim, podemos afirmar, que no exercício anterior, o computador realizaria primeiro a soma, depois a subtração e depois a comparação. Observe que os operadores aritméticos têm precedência em relação aos relacionais.

Mas, e se desejássemos que o computador fizesse a comparação primeiro para depois realizar as operações aritméticas? Deveríamos então forçar a realização da comparação. Para isto usam-se os parênteses:

a+(23<4)-b

O computador realizará sempre o que estiver dentro dos parênteses primeiro para depois fazer o que está fora. Costuma-se dizer, para fixar o conceito, que o computador realiza as operações de dentro para fora. É claro que de dentro dos parênteses para fora destes!

Pode-se, também, utilizar vários níveis de parênteses: ((10+(a+(23<4)-b))-5!=4))

Comando de decisão if

Sintaxe:

```
if (teste) comando;

ou

if (teste)
  {
        comando;
        comando;
        comando;
        ...
        comando;
  }
```

```
//      EXEMPLO 021
//
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
int numero;
        system("CLS");
        cout << "Digite um numero inteiro:";
        cin >> numero;
        if(numero <= 1000)
          {
                cout << "\n\n";
                cout << "Este numero e muito baixo!!!";
          }
        system("PAUSE");
}
```

Comando de decisão if else

Sintaxe:

```
if (teste) comando;
  else comando;
ou
if (teste)
  {
        lista de comandos;
  }
  else
    {
        lista de comandos;
    }
```

```
//        EXEMPLO 022
//
#include <iostream>
#include <cstdlib>

using namespace std;
main()
{
int senha;
          system("CLS");
          cout << "Você está entrando no grande castelo da sabedoria\n";
          cout << "Digite a senha de acesso !! (4 dígitos)\n\n";
          cout << "SENHA:";
          cin >> senha;
          if(senha == 2573)cout << "Pode entrar você acertou!!";
            else cout << "Você errou e não pode entrar!!\a\a\a\a";
          system("PAUSE");
}
```

```
//       EXEMPLO 023
//
#include <iostream>
#include <iomanip >
#include <cstdlib>

using namespace std;
main()
{
char op;
float op1,op2;
          system("CLS");
          cout << "Digite o tipo de operacao desejada:\n";
          cout << "(*) (/)\n";
          cout << "Operação:";
          cin >> op;
          if(op=='*')
                {
                          cout << "digite os dois valores a serem multiplicados:";
                          cin >> op1 >> op2;
                          cout << "\nO resultado e" << setprecision(2) << op1*op2;
                }

           else
                  {
                          cout << "digite os dois valores a serem divididos:";
                          cin >> op1 >> op2;
                          cout<< "\nO resultado e: "<< setprecision(2) << op1/op2;
                  }
          system("PAUSE");
}
```

Exercício 06

Operadores lógicos

Os operadores lógicos são utilizados para realizar operações lógicas entre dois operandos, um do lado esquerdo do operador e outro do lado direito. Os operandos podem ser também expressões.

Os operadores lógicos da linguagem C++ são: && (operador E), II (Operador OU) e ! (operador negação). As operações lógicas estão intimamente ligadas ao funcionamento do computador já que os resultados de cada operação podem assumir apenas dois valores: verdadeiro (1) ou falso (0). Costuma-se representar o funcionamento destes operadores na forma de tabelas. Estas tabelas são denominadas tabelas-verdade.

Operador E (&&)

A	B	R
F	F	F
F	V	F
V	F	F
V	V	V

Operador OU (II)

A	B	R
F	F	F
F	V	V
V	F	V
V	V	V

Operador Negação (!)

A	R
F	V
V	F

Assim, estes operadores devolvem 1 ou 0 conforme o resultado, seguindo as tabelas acima.

```
//        EXEMPLO 024
#include <iostream>
#include <iomanip>
#include <cstdlib>

using namespace std;
main()
{
float nota1, nota2, media;
          system("CLS");
          cout << "Digite suas notas:";
          cin >> nota1 >> nota2;
          media = (nota1+nota2)/2;
          if(media>=60) cout << "Parabens, voce esta aprovado!";
          if(media>=50 && media<60) cout << "Voce foi aprovado, mas estude mais";
          if(media<50) cout << "Sinto muito, mas voce esta reprovado!";
          system("PAUSE");
}
```

```
//    EXEMPLO 025
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
main()
{
float salario,nsalario;
          system("CLS");
          cout << "Digite o salario atual do funcionario:";
          cin >> salario;
          if(salario >= 4000 && salario <= 5000) nsalario = salario * 1.10;
          if(salario >= 2000 && salario <= 3999) nsalario = salario * 1.20;
          if(salario >= 500 && salario <= 1999) nsalario = salario * 1.30;
          if(salario <= 499) nsalario = salario * 1.50;
          cout << "O novo salario do funcionario e "<< setprecision(2) << nsalario;
          system("PAUSE");
}
```

```
//        EXEMPLO 026
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
main()
{
float nota1,nota2,media,presenca;
        system("CLS");
        cout << "Digite a 1ª nota: "; cin >> nota1;
        cout << "Digite a 2ª nota: "; cin >> nota2;
        cout << "Digite a % da presenca: "; cin >> presenca;
        media = (nota1+nota2)/2
        if(media>=60 && presenca >= 75 II media>=50 && presenca>=85)
                cout << "Aluno APROVADO";
                else cout<< "Aluno REPROVADO";
        system("PAUSE");
}
```

Exercício 07

Comando swith case

- comando swith case é if múltiplo que permite várias comparações e vários caminhos no fluxo do programa. Ele pode ser comparado com uma chave seletora.

Sintaxe:

```
swith (variável)
{
        case opção 1: {lista de comandos;}
        case opção2: {lista de comandos;}
        ....
        case opçãoN: {lista de comandos;}
        default: {lista de comandos;}
}
```

```
//      EXEMPLO 027
//      Calculadora de 4 operacoes
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
main()
{
char operacao;
float op1,op2;

        system("CLS");
        cout << "CALCULADORA DE 4 OPERACOES\n\n"
             << "Digite os operandos: "; cin >> op1 >> op2;
        cout << "\n Escolha a operação desejada:\n\n
             << "\n(+) - Soma"
             << "\n(-) - Subtracao"
             << "\n(*) - multiplicacao"
             << "\n(/) - Divisao\n"
             << "\nOpcao: "; cin >> operacao;
        switch(operacao)
        {
            case'+': cout << "\nResultado: "<< setprecision(2) << op1+op2;break;
            case '-': cout << "\nResultado: "<< setprecision(2) << op1-op2;break;
            case'*': cout << "\nResultado: " << setprecision(2) << op1*op2;break;
            case '/': cout << "\nResultado: " << setprecision(2) << op1/op2;break;
            default: cout << "\n\n\n\a\a\aOperacao invalida, sinto muito!!";
        }
        system("PAUSE");
}
```

Exercício 08

Comandos de loop (repetição)

Os comandos de loop, são comandos que permitem estruturas de repetição de comandos ou blocos de comandos em função do resultado de um teste feito nestes comandos.

Loop while

sintaxe:

```
while (teste)
        {
                comando1;
                comando 2;
                .......
                comando n;
        }
```

Enquanto o teste for verdadeiro o bloco de comandos é repetido. É fácil perceber que o controle da repetição deve ser feito dentro do bloco de comandos, por exemplo, através de uma variável que vai mudando de valor até que em um determinado momento ela faz o teste resultar falso e parar a repetição.

```
//      EXEMPLO 028
//      Uso do comando while
//      Contador de 0 a 20

#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
int i=0;

        system("CLS");
        while(i!=20)
        {
                cout << "\ni= "<< i;
                _sleep(500);
                i++;
        }
        system("PAUSE");
}
```

```
//    EXEMPLO 029
//   Uso do comando while
//    Jogo de advinhacao
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
int i,num;
char resp=0;

        while(resp != 'n')
        {
                system("CLS");
                cout << "Voce tem 5 tentativas para acertar o numero \n\n";
                i=0;
                while(i!=5)
                {
                        cout << "\ndigite sua "<< i+1 << "a tentativa: "; cin >> num;
                        if(num == 3450)
                        {
                                cout << "Parabens, voce acertou";
                                system("PAUSE");
                                exit(0);
                        }
                        cout << "Voce ERROU!!\n";
                        i++;
                }
                cout << "Puxa que azar, errou as cinco vezes!!!\n"
                     << "Quer tentar novamente? (s/n)";
                cin >> resp;
        }
}
```

Loop do while

sintaxe:

```
do
        {
                comando1;
                comando2;
                .....
                comandoN;
        }
        while(teste);
```

Observe que neste caso o teste é feito no final do loop. A diferença principal entre o while e o do while é que no do while o bloco de comando é executado pelo menos uma vez, já que o teste é feito no final.

```
//      EXEMPLO 030
//      Uso do comando do while
//      Um contador simples

#include <iostream.h>
#include <cstdlib>
using namespace std;
main()
{
int i=0,num;
        system("CLS");
        cout << ********************************************\n"
             << *****        Contador simples        *****\n"
              << ********************************************\n"
        cout << "\n\n\n\n";
        cout << "Contando:  ";
        do
             {
                        cout << i << "";
                        _sleep(200);
                        i=i+1;
             }
        while(i<50);
        system("PAUSE");
}
```

Loop for

O loop for permite a repetição de um bloco de comandos. Mas, o controle da repetição é feito dentro do comando e não por instruções dentro do bloco que vai ser repetido.

Sintaxe:

```
for(valor inicial ; teste ; variação)
        {
                comando1;
                comando2;
                ......
                comandoN;
        }
```

```
//      EXEMPLO 031
//      Uso do comando de loop for
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
int i;
        system("CLS");
        for (i=0 ; i<10 ; i++ ), cout << "\n Hello, World!!";
        system("PAUSE");
}
```

```
//      EXEMPLO 032
//      LOOP for
#include <iostream>
#include <cstdlib>
using namespace std;
main()
{
int ascii;
        system("CLS");
        for (ascii = 32; ascii < 256; ascii++)
        {
                cout << "\t" << (char)ascii;
                _sleep(100);
                if(ascii%9 == 0) cout << "\n";
        }
        system("PAUSE");
}
```

```
//  EXEMPLO 033
//  Uma moldura para a tela
#include <iostream>
#include <cstdlib>
#include <cstdio>

using namespace std;
main()
{
int col,in;
        system("CLS");
        lin = 1;
        for(col=1;col<81;col++)
        {
                gotoxy(col,lin);printf("#");
                gotoxy(col,lin+23);printf("#");
        }
        col = 1;
        for(lin=1;lin<24;lin++)
        {
                gotoxy(col,lin);printf("#");
                gotoxy(col+79,lin);printf("#");
        }
        system("PAUSE");
}
```

Comandos break e continue

Algumas vezes é necessário sair de um loop devido à ocorrência de um determinado evento. O comando break tem dois usos diferentes: para sair de um case do comando switch ou para sair imediatamente de um loop while, do while ou for, ou seja, quebrar a execução do loop.

```
//      EXEMPLO 034
//      Lancamento do Onibus Espacial
#include <iostream>
#include <cstdio.h>
#include <cstdlib>
using namespace std;

main()
{
int cont = 10;
        system("CLS");
        cout << "LANCAMENTOS ESPACIAIS\n";
        cout << "Contagem Iniciada...\n\n";
        cout << "Para parar aperte enter\n";
        while (cont > -1)
        {
                if(kbhit())
                {
                        system("PAUSE");
                        break;
                }
                cout << "\n" << cont;
                _sleep(1000);
                cont--;
        }
        if (cont == 1)
                cont << "\nLancamentos Espaciais";.
        else
        {
                cout << "\n"Cuidado";
                cout << "\nContagem regressiva em -  "<<cont<<" s";
        }
        system("PAUSE");
}
```

O comando continue, como o break, causa à saída do loop, só que o comando break termina o loop, enquanto que o comando continue "pula" as instruções que estão a sua frente e vai direto para o próximo teste.

```
//      EXEMPLO  035
//      Imprime so os pares

#include <iostream.h>
#include <cstdio>
#include <cstdlib>
using namespace std;
main()
{
int num = 0;
        system("CLS");
        while (num <= 20)
        {
                num++;
                if (num % 2 != 0) continue;
                cout << num << "\n";
        }
        system("PAUSE");
}
```

```
//      EXEMPLO 036
//      Exemplo de break e continue

#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;
main()
{
int numero;
        system("CLS");
        while(cout << "Digite um numero par: ")
        {
                cin >> numero;
                if ((numero % 2) == 1)
                {
                        cout << "Eu disse um numero par";
                        continue;
                }
                break;
        }
        cout << "Obrigado. Eu precisava disso!\n";
        system("PAUSE");
}
```

Exercício 09

Criando suas próprias funções

Os programas que você fez até agora usaram funções como: clrscr(),getch(),kbhit() etc. Estas e, aproximadamente, outras 400 funções já estão definidas e compiladas nas bibliotecas de funções do compilador da BORLAND. Para usá-las basta incluir o arquivo header apropriado no seu programa e checar no help do compilador qual a sintaxe de cada uma.

Agora chegou a hora de escrever funções. E, lembre-se, são à base da linguagem C++. Mas, porque a base?

Para que os programas fiquem estruturados e fáceis de serem desenvolvidos, eles devem ser "quebrados" em pequenos blocos lógicos. É ai que entram as funções. Estes blocos podem ser cada um uma função que recebe valores dos outros blocos, trabalha estes valores e devolve um resultado para outro bloco. Veja que coisa maravilhosa. Blocos independentes, portanto pequenos e fáceis de serem desenvolvidos.

Usaremos um exemplo inicialmente simples e a partir deste exemplo iremos mostrando todas as características e o potencial das funções.

Uma função vazia

Uma função totalmente vazia é aquela que não recebe parâmetros e não deve nenhum valor. Tudo é feito dentro da própria função.

```
//      EXEMPLO 037
//      Funcao totalmente vazia

#include <iostream>
#include <cstdio>
#include <cstdlib>
using namesapace std;

void soma(void);     // prototipo da funcao

main()                  // programa principal
{
        system("CLS");
        soma();
        system("PAUSE");
}

void soma()         // funcao soma()
{
float a,b;          // variaveis locais da funcao soma()
        cout << "Digite oprimeiro numero:  "; cin >> a;
        cout << "\nDigite o segundo numero:  "; cin >> b;
        cout << "\nO resultado da soma dos dois numeros e  "<< a+b;
}
```

Observe:

- O protótipo da função mostrando ao compilador que ela é vazia;
- O corpo da função pode estar depois ou antes de main();
- As funções têm a mesma sintaxe que a função main();
- Você pode criar suas funções e mostrar uma biblioteca delas.

Abrindo um parêntese

Já foi visto que temos variáveis do tipo global e do tipo local. Existem vários outros tipos. Mas, por enquanto, ficaremos com estes dois.

Uma variável global não perde seu valor ao longo da execução do programa, ou seja, uma vez declarada ela tem um espaço na memória garantindo, independentemente do que está acontecendo no programa.

Já as variáveis do tipo local, só existem enquanto o programa está sendo executado dentro da função onde elas foram declaradas. Quando o programa sai desta função estas variáveis são "destruídas", ou seja, o espaço na memória ocupado por elas é liberado.

Perceba que é importante manter na memória só as variáveis que realmente interessam. Aquelas que são de uso momentâneo, podem ser destruídas e recriadas conforme a necessidade.

Uma função recebendo parâmetros

Vamos ver a mesma função do exemplo anterior, só que agora recebendo da função main(), os valores a serem somados.

```
//      EXEMPLO 038
//      Funcao recebendo parametros

#include <iostream>
#include <cstdio>
#include <cstdlib>

using namespace std;
void soma(float, float);      // prototipo da funcao
main()                        // programa principal
{
float a,b;
        system("CLS");
        cout << "Digite o primeiro numero:  "; cin >> a;
        cout << "\nDigite o segundo numero:  "; cin >> b;
        soma(a,b);          // chamada da funcao passando parametros
        system("PAUSE");
}
void soma(float n1, float n2)       // funcao soma()
{
        cout << "\nO resultado da soma dos dois numeros e  "<< n1+n2;
}
```

Observe:

- No protótipo da função não é necessário colocar o nome dos parâmetros, apenas seus tipos;
- A função continua sendo vazia porque não retorna nenhum valor;
- O nome dos parâmetros passado não pode ser os mesmos dos parâmetros recebidos, já que quando a execução do programa passar para a função soma(), as variáveis a e b serão destruídas;
- Na hora de escrever a função, praticamente reescrevemos seu protótipo;
- Depois de executada a função o programa volta para a função chamadora e continua executando o programa.

Função recebendo parâmetros e retornando um valor

```
//    EXEMPLO 039
//   Funcao recebendo parametros e retornando valor
#include <iostream>
#include <cstdio>
#include <cstdlib>

using namespace std;
float soma(float , float);          // prototipo da funcao

main()                          // programa principal
{
float a,b,res;
        system("CLS");
        cout << "Digite o primeiro numero:  "; cin >> a;
        cout << "\nDigite o segundo numero:  "; cin >> b;
        res = soma(a,b);
        cout << "\nO resultado da soma e   "<< res;
        system("PAUSE");
}

float soma(float n1, float n2)                  // funcao soma()
{
        return (n1+n2);
}
```

Observe:

- como a função vai retornar um valor, ela deve ser do tipo do valor retomado;
- é possível atribuir uma função a uma variável, já que o que vai atribuído é o seu valor do retorno e não a própria função. Costuma-se dizer que a função se torna o valor do retorno;
- só é possível retornar um valor por função;
- como a função se transforma no valor de retorno, é preciso reservar espaço na memória para armazenar este valor. É por isso que é preciso fazer o protótipo da função declarando a função como sendo do tipo do valor de retorno;

```
//      EXEMPLO 040
//    Calculo da medida de um aluno
#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;
float media(float, float);      // prototipo da funcao
main()                          // programa principal
{
float n1,n2;
        system("CLS");
        cout << "Digite a primeira nota:  "; cin >> n1;
        cout << "\nDigite a segunda nota:  "; cin >> n2;
        cout << "\nA media deste aluno e  "<< media(n1,n2);
        system("PAUSE");
}

float media(float nota1, float nota2)     // funcao media()
{
        return ((nota1+nota2)/2);
}
```

```
//    EXEMPLO 041
      //    Transformação de nota em conceito

      #include <iostream>
      #include <cstdio>
      #include <cstdlib>

      using namespace std;
      float media(float , float);      // prototipo das funcoes
      char conceito(float);

      main()                          // programa principal
      {
      float n1,n2,med;
      char con;
              system("CLS");
              cout << "Digite a primeira nota:  "; cin >> n1;
              cout << "\nDigite a segunda nota:  "; cin >> n2;
              med = media(n1,n2);
             cout << "\nA media deste aluno e  "<< med;
              con = conceito(med);
              cout << "\nEsta media corresponde ao conceito " << con;
              system("PAUSE");
      }

      float media(float nota1, float nota2)
      {
              return ((nota1+nota2)/2);
      }
      char conceito(float m)
      {
              if(m >= 90) return('A');
              if(m >= 75 && m <= 89) return('B');
              if(m >= 50 && m <= 74) return('C');
              if(m >= 35 && m <=49) return('D');
              if(m <= 34) return ('E');
      }
```

```
//      EXEMPLO 042
//      Transformação de nota em conceito

#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;

float media(float , float);      // prototipo das funcoes
char conceito(float);

main()                          // programa principal
{
float n1,n2,med;
char con;
        clrscr();
        cout << "Digite a primeira nota;  "; cin >> n1;
        cout << "\nDigite a segunda nota:  "; cin >> n2;
        cout << "\nA medida deste aluno e  "<< media(n1,n2);
        cout << "\nEsta media corresponde ao conceito  " <<conceito(media(n1,n2));
        system("PAUSE");
}

float media(float nota1 , float nota2)
{
        return ((nota1+nota2)/2);
}

char conceito(float m)
{
        if(m >= 90) return('A');
        if(m >= 75 && m <= 89) return('B');
        if(m >= 50 && m <= 74) return('C'_;
        if(m >=35 && m <= 49) return('D');
        if(m <= 34) return('E');
}
```

Exercício 10

Matrizes

Conceito

Em C++ uma matriz é um conjunto de variáveis do mesmo tipo. Este conjunto pode ter uma ou mais dimensões: unidimensional, bidimensional ou tridimensional.

Cada elemento é uma variável independente e pode ser acessada usando os comandos que estudamos até agora.

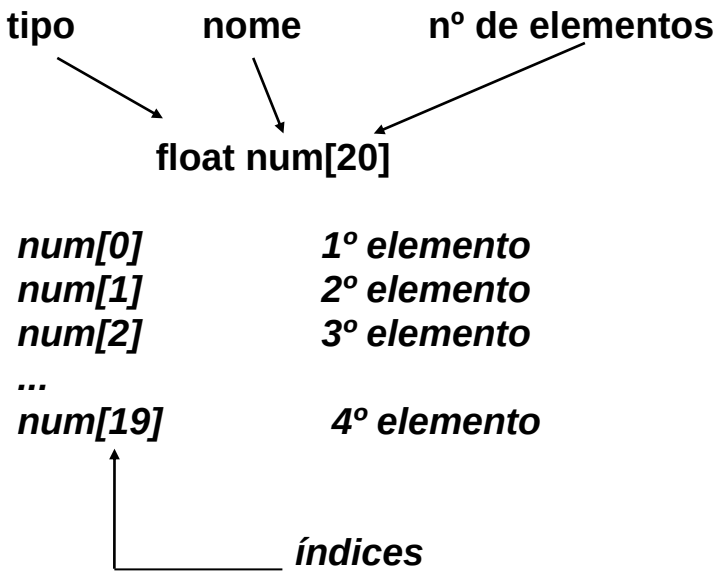
Mas, a matriz pode ser acessada como um todo usando a sintaxe adequada.

Cada variável da matriz é chamada de elemento da matriz e para diferenciar um do outro é utilizado um ou mais índices.

Como uma matriz é um conjunto de variáveis, ela precisa ser declarada. Quando se declara uma matriz é preciso indicar ao compilador de que tipo são os elementos da matriz e quantos elementos ela tem.

Sintaxe da declaração de matrizes:

A declaração de uma matriz é feita como a declaração de uma variável. Primeiro indica-se o tipo de variáveis que a matriz contém, depois indica-se o nome da matriz e em seguida o número de elementos:



Observe que o primeiro elemento é sempre o de índice 0 (zero).

Exemplos

int n[30] matriz unidimensional de 30 elementos inteiros
elementos: n[0], n[1], n[2], n[3], n[4], n[5], ..., n[29]

char letras[20] matriz unidimencional de 20 elementos do tipo char
elementos: letras[0], letras[1], letras[2],..., letras[19]

float xy[10][15] matriz bidimensional de 150 elementos sendo 10 linhas por 15 colunas
elementos: xy[0][0], xy[0][1], xy[0][2], ..., xy[0][14]
xy[1][0], xy[1][1], xy[1][2], ..., xy[1][14]
...
xy[9][0], xy[9][1], xy[9][2], ..., xy[9][14]

Quando se declara uma matriz, por exemplo do tipo float, está se declarando que cada elemento da matriz é do tipo float e não a matriz como um único elemento do tipo float;

Quando devemos usar matrizes?

Imagine a seguinte situação: você precisa elaborar um programa para calcular e armazenar a média de 1000 alunos de uma escola. Como você faria isto sem as matrizes?

Provavelmente você declararia 1000 variáveis para armazenar as primeiras notas, depois mais 1000 variáveis para as segundas notas e finalmente mais 1000 variáveis para armazenar as médias. Imagine você declarando uma a uma estas 3000 variáveis!!!

Mas, ainda não acabou. Você precisa escrever 1000 vezes a equação para calcular a média, já que para cada aluno as variáveis são diferentes. E ainda 1000 couts para imprimir estas médias. Ufa!!. Convenci você que as matrizes são importantes?? Ainda não?? É claro, ainda não mostramos como elas podem resolver este pequeno probleminha.

Uma das vantagens em se utilizar uma matriz é que com apenas uma declaração podemos declarar quantas variáveis forem necessárias para as partes do problema. No exemplo das médias, poderíamos declarar uma matriz para as primeiras notas, outra para as segundas notas e ainda outra para as médias. Observe, seriam no máximo três declarações, enquanto que sem as matrizes teríamos 3000 declarações.

Outra vantagem é que é possível colocar-se variáveis no lugar dos índices. Opa!! Então, o problema das médias foi resolvido. Visualizou a solução? É claro, podemos colocar a leitura das notas, o cálculo das médias e a impressão dentro de um loop for e indexar a matriz utilizando a variável contadora do loop for. Genial! Com algumas linhas de programa teríamos resolvido o problema. Então vamos lá.

```
//        EXEMPLO 043
//        Calculo de 20 médias utilizando matrizes
#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;
main()
{
float p1[20],p2[20],med[20];
int i;

          system("CLS");
          for(i=0;i<20;i++)     // loop de leitura das notas e calculo da media
          {
                    cout << "Digite as notas p1 e p2 do "<< i+1 << "o aluno: ";
                    cin >> p1[i] >> p2[i];
                    med[i] = (p1[i]+p2[1])/2;
          }

          system("CLS");
          for(i=0;i<20;i++)    // loop de impressao das medias
          {
                    cout << "A media do aluno numero   "<<i+1 << "e" << med[i];
          }
          system("PAUSE");
}
```

Matrizes como parâmetros de função

Quando se deseja passar uma matriz como parâmetro para uma função é claro que se deseja passar a matriz como um todo. Portanto, existe uma sintaxe adequada para isto.

```
//      EXEMPLO 044
//      Calculo de medias usando funcoes e matrizes
#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;
void calculo_das_medias(float,float);

void main()
{
float p1[20],p2[20];
int i;

        system("CLS");
        for(i=0;i<20;i++)       // loop de leitura das notas e calculo da media
        {
                cout << "Digite as notas p1 e p2 do "<< i+1 << "o aluno: ";
                cin >> p1[i] >> p2[i];
        }
        calculo_das_medias(p1,p2);
        system("PAUSE");
}

calculo_das_medias(float p1[], float p2[])
{
float med[20];
int i;
        for(i=0;i<20;i++)
        {
                med[i] = p1[i]+p2[i]/2;
                cout << "A media do aluno numero "<< i+1 << "e "<< méd[i];
        }
}
```

Exercício 11