Universidade Cidade de São Paulo (Unicid)

Linguagens de Programação

Professora: Vera Lúcia da Silva

vera@comp.ita.br

Conteúdo Programático:

Tópico Geral	***Conteúdo***
Conceitos Básicos de Linguagem de Programação	Linguagens de programação de alto e baixo nível; Linguagens de programação compilada e interpretada; e Interpretadores e Compiladores.
Introdução à Linguagem C++	Histórico; e Características.
Tipos Básicos de dados em C++	Definições de tipos de dados numéricos, lógicos e caracteres.
Identificadores	Definição de nomes de variáveis e constantes; e Declaração de variáveis e constantes em Linguagem C++.
Operadores lógicos, aritméticos e de atribuição	Definição de operadores em Linguagem C++; e Uso prático dos operadores lógicos, aritméticos, de atribuição, aritméticos de atribuição e unário.
Estrutura básica de programas em Linguagem C++	Programa principal; Comentários; Instruções do pré-processador; e Arquivos de cabeçalho.
Códigos especiais	Definição das funções dos códigos especiais da Linguagem C++.
Entrada e saída padrão C++	Comandos de Entrada/Saída padrão da Linguagem C++.
Comandos de controle de fluxo de execução	Estruturas condicionais – comandos *if*, *if/else* e *switch*; e Estruturas de repetição – comandos: for, *while* e do *while*.
Funções	Definição de funções em Linguagem C++; Protótipo da função; Passagem de parâmetros para função; Valores de retorno das funções; e Funções Recursivas.
Vetores e matrizes em linguagem C++	Definição e uso prático de Vetores e Matrizes.
Arquivos	Definição de arquivos; e Comandos de manipulação de arquivos.
Alocação dinâmica em Linguagem C++	Definição e manipulação de Ponteiros.
Programação Orientada a Objetos	Definição do paradigma da orientação a objetos; Conceitos de classes, objetos, instancia, polimorfismo, herança, composição, generalização, especialização, métodos e atributos; e Introdução a UML – Diagramas de classes.
Introdução a Linguagem Java	Histórico; Tipos básicos de dados em linguagem Java; e Comparação das características das Linguagens C++ e Java.

Programação em Linguagem Java	Ambiente de programação; Compilador e máquina virtual Java; e Desenvolvimento de programas em Linguagem Java, baseados nos conceitos de orientação a objetos.
Criação de *Applets* com Java	Desenvolvimento de aplicativos para Internet; Definição dos métodos básicos para o desenvolvimento de um aplicativo *Applet*; e Comandos da Linguagem HTML para execução de *Applets*.

Bibliografia

Bibliografia Básica:

- MIZRAHI, Victorine Viviane. ***Treinamento em Linguagem C++***. São Paulo: Makron Books, 1995.
- DEITEL, H. M., DEITEL, P.J. **JAVA Como Programar**. Porto Alegre: Bookman, 2001.
- DEITEL, H. M., DEITEL, P.J. **C++ Como Programar**. Porto Alegre: Bookman, 2001. Tradução por: Carlos Arthur Lang Lisbôa e Maria Lúcia Lang Lisbôa.
- HORSTMANN , C. S. **Core Java 2**. São Paulo: Makron Books, 2000

Bibliografia Complementar:

- FURGERI, Sérgio. **Java 2: Ensino Didático: Desenvolvendo e Implementando Aplicações.** São Paulo: Érica, 2002.
- DAMASCENO JÚNIOR, Américo. **Aprendendo Java: Programação na Internet**. 3º ed. São Paulo: Érica, 1996.
- TITTEL,ED. **Internet World 60 Minutos Para Aprender Java .** São Paulo: Berkeley Brasil Editora Ltda, 1997.
- GOODWILl, James. ***Developing Java Servlets***. Indiana: Sams Publishing, 1999.
- HOLZNER, Steven. **Borland C++ Programação for Windows**. São Paulo:Makron Books, 1995.
- FURLAN , José Davi. Modelagem de objetos através da UML . São Paulo: Makron Books, 1998.
- NAUGHTON, Patrick. **Dominando o Java**. São Paulo:Makron Books, 1996.
- BARKAKATI, Nabajyoti. **Visual C++ developer´s guide**. USA: Sams Publishing, 1993.
- SILVA, José Carlos G. da, ASSIS, Fidelis S. G. de. **Linguagens de Programação: Conceitos e Avaliações**. São Paulo:McGraw-Hill, 1998.
- BARKAKATI, Nabajyoti. **Visual C++ Guia de Desenvolvimento Avançado**. Rio de Janeiro:Berkeley, 1994. Tradução por: Luís Gustavo Neves da Silva.
- YAO, Paul. **Borland C++ 4.0: Programação for Windows**. São Paulo: Makron Books, 1995.
- JAMSA, Kris. **Sucesso com C++.** São Paulo: Érica, 1995.
- DEITEL, H. M., DEITEL, P.J. **C++ How to program**. New Jersey: Prentice-Hall, 1998.
- HOFF, Arthur Van. **Ligando em Java**. São Paulo:Makron Books, 1996.
- LALANI, Suleiman Sam. **Java:Biblioteca do programador**. São Paulo: Makron Books, 1997.
- Newman, Alexander. ***Usando Java***. Tradução de: *Using Java*. Rio de Janeiro: Campus, 1997.
- LINDEN, Peter van der. **Just Java**. Traduzido por: Betina de Oliveira. São Paulo: Makron Books, 1997.
- LEMAY,Laura. **Aprenda Em 21 Dias Java** : Editora Campus Ltda, 1999.
- CORNELL. **Core Java 2, 1.** Editora: Makron Books Do Brasil, 2000.

Lista de Exercícios

1-) Faça o algoritmo e o fluxograma para resolver os seguintes problemas:

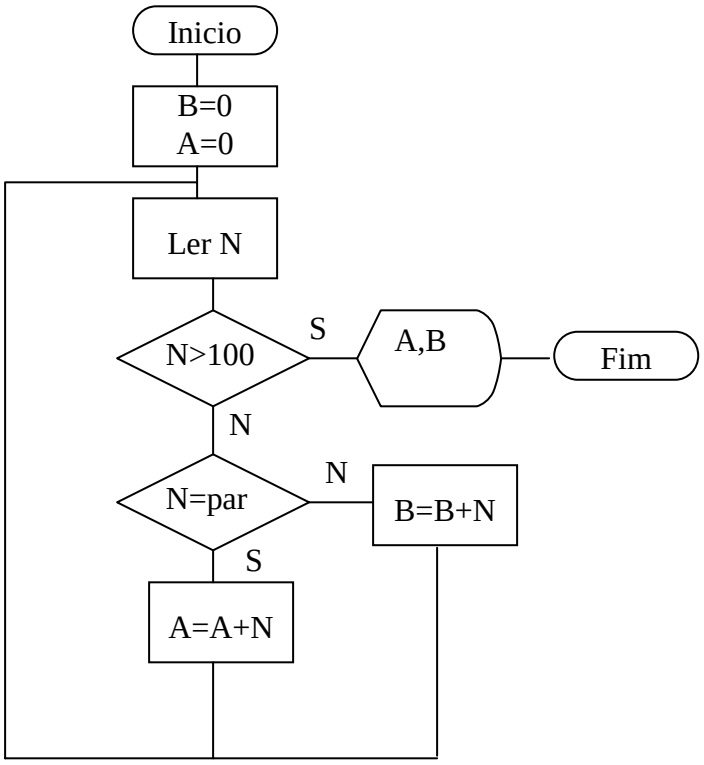
a) Somar os números pares de 0 a 100 e imprimir o resultado na tela.
b) Dada a entrada de 3 números imprimir o maior na tela.
c) Imprimir os número de 0 a 200 na tela.

2-) Três crianças moram na mesma rua. Observando as pistas abaixo, identifique o nome completo e a idade de cada uma.

- A menina de sobrenome Bento tem três anos a mais que Maria
- A criança cujo sobrenome é Silva tem 9 anos

	Bento	Castro	Silva	7	9	10
Ana						
João						
Maria						
7						
9						
10						

3-) Dado um arquivo contendo os números (N) : 1,5,10,100,15,101,20 respectivamente, dizer qual é o valor de A e B no final do fluxograma abaixo.



Conceitos Básicos de Linguagem de Programação

1. Linguagens de Programação - LPs

Linguagens de Programação são conjuntos de notações formais para descrever ações ou operações a serem realizadas por um computador. São ferramentas para o desenvolvimento de software.

> Software é um conjunto de programas, módulos, procedimentos, regras e quaisquer documentações associadas à operação de um sistema de processamento de dados.

Exemplos de Linguagem de programação: Cobol, Pascal, Fortran, Linguagem C, Java, entre outras.

Os componentes gerais de uma linguagem são a sua sintaxe e a sua semântica.

1.1. Sintaxe e Semântica das LPs

***Sintaxe*:** Conjunto de regras formais para a composição de um texto na linguagem (programa) a partir do agrupamento de letras, dígitos e/ou outros caracteres (alfabeto da linguagem). A sintaxe é um conjunto de regras formais para a escrita do programa.

> Um programa é uma coleção de objetos globais, que podem ser procedimentos, funções, protótipos de funções, declarações de variáveis - que são constituídos por agrupamentos de palavras reservadas (comandos e instruções), palavras criadas pelos usuários, letras e dígitos e/ou outros caracteres.

A sintaxe estuda a disposição destas palavras e símbolos nas linhas de programa e a disposição destas linhas, dentro do programa, como um todo, bem como a relação lógica das linhas do programa entre si. Está implícito ainda, na sintaxe, a corretude das palavras usadas, que caso contrário serão apontadas como erradas, visando a correção.

Semântica diz respeito à significação. A semântica é o estudo do sentido dos significantes. Em programação a semântica diz respeito ao significado do programa

sintaticamente válido. O que pode ocorrer é que um programa seja sintaticamente válido sem, no entanto, ter um significado lógico coerente. É parte do domínio da semântica verificar esta coerência em termos de significado lingüístico e não em termos de lógica de programa.

1.2. Nível das Linguagens de Programação

- **Linguagem de BAIXO NÍVEL:** é uma linguagem mais próxima da Linguagem de Máquina (L.M.), ou seja, mais próxima do *hardware*. Ex: *Assembler*

- **Linguagem de ALTO NÍVEL:** são linguagens, cada vez mais, afastadas da L.M., através do uso de abstrações cada vez mais complexas. Estas abstrações se dão em função da adoção de tipos de dados, palavras reservadas, funções, procedimentos automáticos etc. Ex: Cobol, Pascal, Java

- **A Linguagem C++**: é suficientemente próxima do hardware e ao mesmo tempo prevê a utilização de recursos de alto nível, por isso é classificada como ***linguagem de nível médio***.

1.3. Interpretação X Compilação

Nas ***linguagens interpretadas*** um programa é executado instrução a instrução, ou seja, cada comando é, primeiro traduzido para a linguagem de máquina, para somente em seguida, ser executado. Na Interpretação as ações resultantes de comandos da linguagem de alto nível são executadas diretamente, seguindo os passos:

- Obter próximo comando;
- Determinar as ações a executar;
- Realizar a ação.

Nas ***linguagens compiladas*** um programa é executado somente quando toda a tradução foi completada. A compilação de um programa fonte (texto escrito diretamente na linguagem de alto ou médio nível) prevê que o mesmo seja traduzido para a linguagem da máquina correspondente antes da execução. Esta tradução é geralmente

feita em diversas etapas:

ETAPAS	CRIA O	Extensão
[1a. etapa] depuração sintática	programa fonte	**.C ou .CPP**
[2a. etapa] geração de código de máquina relocável	programa objeto	**.OBJ**
[3a. etapa] *linkage* - liga códigos e funções de biblioteca	programa executável	**.EXE**

Programa Fonte: Declarações escritas em uma linguagem de programação.

Programa Objeto: Programa objeto ou código objeto é a representação de seu programa fonte em código de máquina, quando o mesmo é submetido a um compilador.

Erros:

– Nas linguagens compiladas os erros podem ser detectados em tempo de compilação, em tempo de *"linkagem*" (ligação), e mesmo, depurados todos os erros dessas etapas, podem ocorrer erros em tempo de execução.

– No caso das linguagens interpretadas os erros são detectados em tempo de execução.

Exemplos:

–Linguagem Compilada: linguagem C/C++

–Linguagem Interpretada: Linguagem *Basic*

1.3.1. Compiladores

Um compilador lê uma instrução do programa, checa sua sintaxe, traduz para a linguagem de máquina, passa para a próxima instrução até atingir a última instrução do programa.

Ao final deste processo o compilador constrói um arquivo de código objeto (.OBJ), porém para ser executado o código gerado deve ser submetido a um *linkeditor.*

A "linkedição" consiste em agregar ao programa .OBJ bibliotecas existentes na linguagem de programação utilizada e rotinas em linguagens de máquina que lhe permitirão sua execução, resultando um programa com extensão .EXE, que pode ser executado diretamente.

1.3.2. Interpretadores

Um interpretador lê uma instrução do programa, em seguida verifica a consistência de sua sintaxe, converte-a para linguagem de máquina e a executa.

Esse processo é repetido para todas as outras instruções do programa até a última instrução ser executada ou a consistência apontar algum erro. Caso exista uma rotina dentro do programa que necessite ser executada 100 vezes, o processo de tradução e checagem de sintaxe será executado 100 vezes para a mesma linha da rotina.

Principais Vantagens	
Compilador	**Interpretador**
Aumento da velocidade de execução do programa	Facilidade de desenvolvimento de um programa na fase de aprendizagem
Independência do Interpretador	Fácil depuração de erros
Proteção do código fonte – programas .EXE não podem ser alterados	

Bibliografia:

LEITE, Aury de Sá. **Introdução à Linguagem de Programação C++**. Notas de aulas. Não publicado, 1999.

MIZRAHI, Victorine Viviane. ***Treinamento em Linguagem C++***. São Paulo: Makron Books, 1995.

SILVA, José Carlos G. da, ASSIS, Fidelis S. G. de. **Linguagens de Programação: Conceitos e Avaliações**. São Paulo:McGraw-Hill, 1998.

LINGUAGEM C/C++

1. Introdução a Linguagem C/C++

1.1. Histórico

A linguagem C foi desenvolvida por Dennis Ritchie, pesquisador dos Laboratórios Bell, no ano de 1972. Foi inicialmente criada para ser um "*ASSEMBLY* portátil". Rodava num computador PDP-11, usando o sistema operacional UNIX. Posteriormente foi utilizada para reescrever o UNIX. Somente em 1978 Brian W. Kernighan e Dennis Ritchie estabeleceram a sintaxe e a semântica da linguagem C.

O desenvolvimento de C foi fortemente influenciado pela linguagem B, desenvolvida por Ken Thompson, que por sua vez, teve suas origens na linguagem BCPL, desenvolvida por Martins Richards. Tais linguagens surgiram da evolução da linguagem Algol CPL, na tentativa de aproximar esta linguagem do hardware.

Com a saída do UNIX dos limites da Bell *Laboratories* em meados dos anos 70, a linguagem C se tornou mais popular e por volta de 1980 muitas versões de compiladores C apareceram, oferecidos por diversas empresas e compatíveis inclusive com outros sistemas operacionais.

Em maio de 1987 a Borland lançou o Turbo C, especialmente destinado aos IBM-PCs e compatíveis que utilizavam como sistema operacional o PC-DOS ou equivalentes.

Uma evolução natural da linguagem C, o C++ (C plus-plus), foi criada por um outro pesquisador dos Laboratórios Bell, Bjarne Stroustrup, em 1983, tendo sido lançada no mercado em 1985 pela AT&T. O C++, um superconjunto da linguagem C, utiliza todas as bibliotecas do C, possuindo ainda, algumas bibliotecas suplementares muito diferentes; que visam permitir a Programação Orientada a Objetos (POO). O C++ possui, ainda, um compilador mais eficiente que o da linguagem C, eficiência esta que é comprovada em termos de forte verificação dos tipos de dados envolvidos num programa, checagem de parâmetros passados para uma função, entre outras vantagens.

Família da Linguagem C++:

CPL (Combined Programming Language)

BCPL (Basic CPL)

B (Ken Thompson, Bell Labs, 1970)

C (Dennis Ritchie, Bell Labs, 1972).

C++ (Bjarne Stroustrup, Bell Labs, 1983)

1.2. Dados Técnicos Gerais

A linguagem C/C++ é modular: a principal característica das linguagens modulares é a utilização de blocos (utiliza uma grande variedade de unidades de programa compatíveis entre si) que estimulam o estilo de programação *TOP-DOWN* (de-cima-para-baixo, ou melhor, do maior-para-o-menor, ou ainda, estimula a divisão de problemas amplos, em pequenos sub-problemas). A Linguagem C/C++ permite a utilização de sub-rotinas compiladas em separado e permite a criação de bibliotecas de funções e sub-rotinas. As bibliotecas de funções e sub-rotinas podem ser muito amplas, pois a *linkagem* (ligação do programa objeto às funções de biblioteca para a geração do programa executável) se dá somente com relação a cada uma das funções utilizadas e não há a *linkagem* obrigatória de toda a biblioteca, como por exemplo, no Pascal.

A Linguagem C/C++ é estruturada: desencoraja a utilização dos *goto's* - desvios incondicionais -, que geram os chamados códigos "macarronada". O "*goto*" é substituído por diversos tipos de laços e desvios, tais como: *while, do-while, for; if-then-else, switch*, que permitem ao programador exercer um controle lógico mais eficaz sobre os códigos fontes de seus programas.

A linguagem C/C++ possui sub-rotinas com variáveis locais, isto é, funções cujas variáveis são visíveis apenas dentro desta função e somente no momento em que estas funções estejam sendo usadas. Assim as variáveis com mesmo nome, que pertençam a funções distintas, são protegidas dos efeitos colaterais ***(proteção de variáveis***), isto é, uma modificação em nível funcional não acarretará mudança na variável em nível global. Desta forma, mudanças de

valores de variáveis no corpo do programa principal não afetam as variáveis de funções e vice-versa, a não ser que o programador assim o queira.

Programa em Linguagem C/C++ tem **aspecto modular e funcional**; o próprio programa principal "*main()* " é uma função.

O C/C++ permite **chamadas recursivas de funções**, isto é, uma função pode chamar-se a si mesma repetidas vezes.

O C/C++ permite a alocação da memória disponível para armazenagem de dados em tempo de execução do programa, é a chamada **alocação dinâmica** de memória (possível através da utilização de ponteiros = *"pointers*").

A Linguagem C/C++ permite que o programador **acesse detalhes do hardware** da máquina, no entanto **provê recursos de alto nível** suficientes para que o programador implemente seus programas sem a necessidade de se preocupar com detalhes da máquina.

2. Tipos de Dados na Linguagem C/C++

Na Linguagem C/C++ as variáveis podem assumir os tipos: Inteiro, ponto flutuante ou real, caractere e ponteiros.

> "Variável em C++ é um espaço de memória reservado para armazenar um certo tipo de dado, tendo um nome para referenciar o seu conteúdo".

Um tipo de variável define um tamanho e uma forma de armazenamento para a variável definida.

Os tipos apresentados podem ser agrupados como:

- Real (float, double)
- Inteiro (short, int, long)
- Caractere (char)
- Ponteiro – definido por um " ***** " na frente do nome da variável com o tipo para o qual o ponteiro apontará. Este tipo de variável contém o endereço de outra variável.

As variáveis em C/C++ podem ser dos seguintes tipos:

Tipo	Definição	Byte
void	não devolve nenhum valor, não ocupa memória	0
char	variáveis que contém os caracteres ASCII abrange os códigos dos caracteres **de -128 a 127**	1
Unsigned char	variáveis que contém os caracteres ASCII abrange os códigos caracteres **de 0 a 255**	1
short	valores numéricos inteiros range: **de -32768 a 32767**	2
Unsigned short	valores numéricos inteiros não negativos range: **de 0 a 65535**	2
int	valores numéricos inteiros range: **de -32768 a 32767**	2
Unsigned int	valores numéricos inteiros não negativos range: **de 0 a 65535**	2
Long int	valores numéricos inteiros longos range: **de –2147483648 a 2147483647**	4
Unsigned long int	valores inteiros longos não negativos range: **de 0 a 4294967265**	4
float	números reais em ponto flutuante range: **de (±) 3.4E-38 a (±) 3.4E+38**	4
double	valores numéricos reais em ponto flutuante range: **de (±) 1.7E-308 a (±) 1.7E+308.**	8
Long double	valores numéricos reais em ponto flutuante range: **de (±) 3.4E-4932 a (±) 1.1E+4932**	10

Modificadores de tipos:

Com exceção do tipo *void*, os tipos de dados básicos da linguagem C/C++ podem ser acompanhados pelos modificadores: *long, short* e *unsigned*, como mostra a tabela acima.

O modificador *unsigned* é usado para considerar os tipos *char, short, int* e *long* só com valores positivo, aumentando um bit o tamanho do número a ser armazenado.

Observação: ao definir um tipo de dado em seu programa observe que o range, ao ser ultrapassado ou antepassado, virá a provocará erro. Se o seu ambiente de programação C/C++ possuir dispositivo para *tratamento* desse tipo *de erro*, um aviso será emitido ou o programa será interrompido, caso contrário, ao haver um estouro de range, o próprio ambiente "cuida do caso a seu modo", isto é, o range é visto como uma fita contínua, ligada pelas extremidades do range, onde ao maior elemento sucede o menor deles. Por exemplo, se estivermos usando um tipo int, a soma 32767+1 resultará -32768; a soma 32767+2 terá como resultado -32767, etc.

3. Operadores da Linguagem C/C++

3.1. Operadores Aritméticos

adição: **+**

subtração: **-**

multiplicação: *****

divisão: **/**

Resto de uma divisão de inteiros (**módulo):** % (este é um símbolo *sobrecarregado*)

Observação:

Um símbolo é sobrecarregado quando é utilizado de diversas formas ou com diversas funções, o que é o caso do símbolo **%**.

3.2. Operadores de Decremento/Incremento

Decremento de 1 unidade (pós ou prefixado): **--** ; exemplo x-- ou --x

Incremento de 1 unidade (pós ou prefixado): **++** ; exemplo x++ ou ++x

Prefixado: incrementa/subtrai a variável de 1 (um), e usa o novo valor da variável na expressão que reside.

Pós-fixado: usa o valor corrente da variável na expressão na qual reside para avaliação e depois incrementa/subtrai a variável de 1(um).

3.3 Operadores Relacionais

maior:	**>**
maior ou igual:	**>=**
menor:	**<**
menor ou igual:	**<=**
igual (verificador de igualdade):	**==**
desigual:	**!=**

3.4. Operadores Lógicos

and (e):	**&&**
or (ou):	**||** (no teclado elas aparecem como barras interrompidas)
not (Não):	**!**

3.5. Operadores de Atribuição

O **símbolo " ="** é chamado comando de **atribuição** quando ligar duas variáveis ou uma variável a esquerda e um valor numérico à direita, assim, "x = n" deve ser lido a variável x recebe o valor n, ou o valor da variável n.

Operadores de Atribuição Aritméticos: +=, - =,*=,/=,%=

Como regra geral, se **x** é uma variável, **exp** uma expressão e **op** um operador aritmético, então:
x **op**= **exp** equivale a x = x **op** (**exp**)

Exemplos:

1) i +=2; equivale a i =i + 2;
2) x *= y+1; equivale a x = x * (y +1)
3) Seja d =12
   d %= 9 equivale a d = d % 9 => d = 3

3.6. Operador Ternário: Operador “?:”

expressão 1 ? expressão 2 : expressão 3;

Se a expressão 1 for verdadeira, a expressão 2 é avaliada e seu valor passa a ser o resultado, caso contrário a expressão 3 é avaliada e seu valor é o resultado, por exemplo:

" x > y ? x = x-1 : x = x+ 1; " : que significa se x é maior que y então x é decrementado de uma unidade, caso x seja menor ou igual a y, o x será incrementado de uma unidade. Note que esta sentença poderia ser escrita como: **"x>y ? x-- : x++; ".**

Operadores

Aritméticos	Atribuição	Atribuição Aritméticos	Incremento Decremento	Relacionais	Lógicos	Ternário
+ adição	=	+=	++	>	&& (e)	?:
- subtração		-=	--	>=	|| (ou)	
* multiplicação		*=		<	! (não)	
/ divisão		/=		<=		
% módulo		%=		==		
				!=		

Precedência e Associatividade dos operadores:

A precedência define a ordem de avaliação dos operadores dentro de uma expressão. A tabela abaixo apresenta a ordem de prioridade na avaliação dos operadores de forma decrescente, do maior para o menor. A associatividade define o sentido da avaliação. Na maioria dos casos a avaliação é realizada da esquerda para a direita, mas existem algumas exceções como mostra a tabela abaixo:

Operadores	**Associatividade**	**Tipo**
( )	Esquerda para Direita	parêntese
++ -- - !	Direita para Esquerda	unário
* / %	Esquerda para Direita	Aritméticos
+ -	Esquerda para Direita	Aritméticos
< <= > >=	Esquerda para Direita	Relacional
= = !=	Esquerda para Direita	Relacional
&&	Esquerda para Direita	E lógico
||	Esquerda para Direita	OU lógico
?:	Direita para Esquerda	condicional
= += -= *= /= %=	Direita para Esquerda	Atribuição

Tabela Verdade: && e ||

Expressão 1	**Expressão 2**	**Expressão 1 && Expressão 2**	**Expressão 1 || Expressão 2**
Falso	Falso	Falso	falso
Falso	Verdadeiro	Falso	Verdadeiro
Verdadeiro	Falso	Falso	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro

Tabela Verdade: !

Expressão	**!** Expressão
Falso	Verdadeiro
Verdadeiro	Falso

4. A Estrutura Básica de um Programa em C++

Um programa em C/C++ é uma coleção de objetos globais que podem ser variáveis e funções. As unidades funcionais de um programa em C++ são as funções.

Um programa C++ consiste em um conjunto de funções ou de uma única função com os seguintes elementos básicos:

<tipo> <nome> (<parâmetro>) **{** **<instrução 1>;** **<instrução 2>;** **.** **.** **.** **<instrução n>;** **}**	**<tipo>** - o tipo de dados que a função irá retornar. (*void* se a função não tiver valor de retorno). **<nome>** O nome da função – Seguir as regras de identificação de variáveis. **{ }** - Indicador de início e término da função. **;** - Todo comando em C++ deve finalizar com ; (ponto-e-vírgula).

- A comunicação entre as funções é feita pela passagem de parâmetros (por valor ou por referência), por retorno de valores funcionais à função chamadora, ou através de variáveis globais, que são visíveis (têm validade) em todo o programa;
- As funções podem ser definidas em qualquer ordem dentro de um programa fonte, sendo que o texto do programa fonte pode estar subdividido em vários outros arquivos-fonte que poderão ser compilados em separado e depois ligados (*linkage*);
- Espaços em branco, tabulações e mudanças de linha são ignorados pelo compilador. Portanto, torna-se possível escrever várias instruções em uma única linha, inserir espaços em branco, tabulações e pular linhas à vontade. Porém existem algumas exceções:
    - Cadeias de caracteres constantes como “O primeiro programa” não podem ser separadas em diversas linhas;
    - Nome de funções, operadores e comandos da linguagem não podem ser separados;
    - Diretivas do pré-processador como **#include** não podem ser escritas em várias linhas;

- A função **main()** (principal) é única para cada programa. Esta função marca o ponto de partida e término de um programa e se este for constituído de uma única função, esta será a principal. A função **main()** não precisa estar no início do programa.

4.1. Um programa em C++

Um simples programa em C++.

```
1  // Primeiro programa em C++
2  #include <iostream.h>
3  void main()
4  {
5    cout << “Primeiro programa”;
6  }
```

Analisando o primeiro programa:

- A **linha 1** trata de um comentário.

Comentário: Comentários são utilizados para documentar o código fonte. Podem ser colocados em qualquer lugar do programa e são tratados pelo compilador como espaços em branco. Existem duas maneiras de colocar comentários em um programa C++.

1) Duas barras **(//)** – comentários de linha: Este estilo de comentário começa com duas barras e termina com o final da linha. Tudo que estiver escrito após as duas barras será ignorado pelo compilador.

2) Comentários usando os símbolos **/*** e ***/ -** Tudo que estiver escrito entre os símbolos /* (início) e */ (término) será ignorado pelo compilador. O conteúdo entre estes símbolos pode estar escrito em várias linhas, numa única linha ou na mesma linha de uma instrução C++. Exemplos:

```
/* Declaração de variáveis
   Inteiras e reais
*/
```

```
/* Programas em C++ */
```

- A **linha 2** do programa apresenta uma diretiva do pré-processador C++.

  Pré-Processador C++ - O Pré-Processador C++ é um programa que examina o programa-fonte em C++ e executa nele certas modificações com base em instruções chamadas **diretivas**.

  Diretivas - Toda diretiva é iniciada pelo símbolo # e seu texto deve ser escrito em uma única linha. Caso ocorra a necessidade de quebra de linha, pode-se terminar a linha com a barra invertida "\" e continuar em outra linha.

  Diretiva #include - Esta diretiva inclui no programa fonte o arquivo indicado antes do programa ser compilado. A linha 2 do programa solicita ao compilador incluir o arquivo **iostream.h** no programa fonte manipulado. A busca pelo programa indicado pode ocorrer de duas maneiras:

  - Ao utilizar a diretiva #include com os símbolos **<> (maior e menor)** indica que a biblioteca ou arquivo especificado deve ser procurado no diretório INCLUDE.
  - Ao utilizar a diretiva #include com os símbolos **" " (aspa duplas)** indica que a biblioteca ou arquivo especificado deve ser procurado primeiramente no diretório atual e depois, caso não seja encontrado, no diretório INCLUDE. Exemplo: #include "iostream.h".

  Arquivos de inclusão – ou arquivos de cabeçalho contêm definições e declarações necessárias para que o compilador reconheça vários identificadores da linguagem C++. Geralmente possuem extensão .h (Header ou cabeçalho). O arquivo **iostream.h** contém declarações necessárias ao uso dos objetos **cout** e **cin** e dos operadores de **inserção <<** e **extração >>** e outras definições básicas de entrada e saída padrão.

  - A **linha 3** apresenta a declaração da função principal, **main()**;
  - As **linhas 4 e 6** apresentam os caracteres de início e término de função;
  - A **linha 5** apresenta uma instrução de impressão da frase entre aspas duplas na tela.

  cout – é um objeto da classe de I/O predefinida em C++. Sua função está associada à saída padrão. O operador <<, conecta a mensagem a ser impressa a **cout**.

5. Identificadores em C++

Para definir o nome de variáveis em C++, faz-se necessário seguir algumas regras básicas para evitar confusões:

1. Letras maiúsculas e minúsculas são diferentes. Considerando esta regra, os seguintes nomes de variáveis são diferentes: PESO, Peso, peso, pESO, peSO, ...
2. A quantidade de caracteres significativos: somente os 32 primeiros caracteres que formam o nome da variável são considerados.
3. Uma variável em C++ não pode ter o mesmo nome que uma palavra-chave (palavra reservada da linguagem). A seguir temos uma tabela de palavras-chave em C++:

Palavra chave em C e C++				
auto	*break*	*case*	*char*	*const*
continue	*default*	*do*	*double*	*else*
enum	*extern*	*float*	*for*	*goto*
If	*int*	*long*	*register*	*return*
short	*signed*	*sizeof*	*static*	*struct*
switch	*typedef*	*union*	*unsigned*	*void*
volatile	*while*			
Palavra chave somente em C++				
Asm	*bool*	*catch*	*class*	*const_cast*
delete	*dynamic_cast*	*explicit*	*false*	*friend*
inline	*mutable*	*namespace*	*new*	*operator*
private	*protected*	*public*	*reinterpret_cast*	*static_cast*
template	*this*	*throw*	*true*	*try*
typied	*typename*	*using*	*virtual*	*wchar_t*

4. O nome de uma variável em C++ pode conter letras, números e o caractere ***underscore*** "_", sendo obrigatório que o início do nome de uma variável contenha uma letra ou o caractere de sublinhado (***underscore***).

5.1. Declaração de Variáveis

Em C++ todas as variáveis devem ser declaradas. A declaração de uma variável deve ocorrer antes do uso da variável pelo programa. Uma variável pode ser declarada em qualquer lugar do programa.

Uma declaração de variável consiste no nome de um tipo e nome da variável, seguido de ponto-e-vírgula. Exemplos:

```
// Exemplo01.cpp
#include <iostream.h>
void main()
{
  int num1;
  num1=44;
  cout << “\nO Primeiro numero e “ << num1;
  int num2;
  num2=88;
  cout << “\nO Primeiro numero e “ << num2;
}
```

A declaração de variável pode ser combinada com o operador de atribuição, assim fornecendo a variável um valor inicial.

```
//Exemplo02.cpp
#include <iostream.h>
void main()
{
  int evento = 5;
  char corrida = ‘C’;
  float tempo = 27.25;
  cout << “\nO tempo vitorioso na eliminatória “
       << corrida << “\nda competição “ << evento
       << “foi “ << tempo <<”. “;
}
```

5.2. Constantes

Uma constante tem valor fixo e inalterável.

- Números constantes em C++ podem ser escritos nas seguintes bases:

Base	**Representação**	**Exemplos**
Decimal	Normal	6766, 10, 34,...
Hexadecimal	Com ‘0x’ à frente	0x72, 0xFFA, ...
Octal	Com ‘0’ à frente	07, 045, 034,...

- Caracteres constantes são escritos entre aspas simples (‘). Exemplo: ‘5’, ‘a’,’Q’,...
- Cadeia de caracteres constante é reconhecida quando escrita entre aspas duplas (“). Exemplo: “Primeiro Programa”

6. Códigos Especiais

O “*pintf*()” (no C) ou o “*cout*<<” (no C++) pode conter dentro das aspas (" ") <u>códigos de escapes:</u>

Quando trabalhamos com a impressão de caracteres em tela ou impressora, alguns códigos são necessários em C++. Esses códigos são representados pela barra invertida (\) acompanhada de outros caracteres indicadores.

\b	retrocesso
\f	*form feed* = alimentação de formulário – salta página de formulário
\n	*line feed* (CR + LF) = mudança de linha
\r	retorno de carro, sem mudar de linha
\t	tabulação horizontal
\"	aspas duplas
\'	*plic* (ou aspas simples) - apóstrofo
\0	zero, *NULL*, ou final de *string*
\\	imprime uma barra: “\”
\v	tabulação vertical
\a	alerta (*beep*)
\xdd	constante hexadecimal

Exemplo:

```
// Este programa salta uma linha antes de imprimir
#include <iostream.h>
void main()
{ cout << "\nPulando linha"; }
```

7. Imprimindo dados usando o *cout*

C++ fornece uma biblioteca de funções e classes conhecida como "*streams*". Ela fornece os elementos necessários para a execução de operações de leitura e impressão (I/O). O objeto *cout* está associado à saída padrão.

O objeto *cout* utiliza o operador <<, denominado "operador de inserção", para conectar a mensagem a ser impressa.

O objeto *cout* permite imprimir valores numéricos em diferentes bases:

dec – A impressão do campo é apresentada em decimal (modo *default*).

hex – A impressão do campo é apresentada em hexadecimal.

oct – A impressão do campo é apresentada em octal.

Exemplo:

```
// Este programa imprime o valor 65 em diferentes bases
#include <iostream.h>
void main()
{ int n=65;
 cout << '\n' << "hexadecimal " << hex << n;
 cout << '\n' << "Octal       " << oct << n;
 cout << '\n' << "Decimal     " << dec << n;
}
```

Para imprimir símbolos entre 128 a 255 (símbolos de língua estrangeira e caracteres gráficos), o objeto *cout* utiliza a representação **\xdd**, onde **dd** representa o código do caractere na base hexadecimal.

Bibliografia:

MIZRAHI, Victorine Viviane. ***Treinamento em Linguagem C++***. São Paulo: Makron Books, 1995.

LEITE, Aury de Sá. **Introdução à Linguagem de Programação C++**. Notas de aulas. Não publicado, 1999.

Programas exemplos:

```
// Venus.cpp
#include <iostream.h>
void main()
{
cout <<"\n VENUS ESTA A " <<67 <<" MILHOES DE MILHAS " <<'\n' <<" DO SOL.";
}
```

```
// jota.cpp
#include <iostream.h>
void main()
{
 cout << "\n A letra " << 'j';
 cout << " pronuncia-se " << "jota" << '.';
}
```

Imprimindo caracteres Gráficos

É possível imprimir qualquer caractere da tabela ASCII, inclusive códigos de 128 a 255, que consistem em símbolos de línguas estrangeiras e caracteres gráficos. Para imprimir caracteres acima de 127 é utilizada a notação **\xdd**, onde **dd** representa o código do caractere na base hexadecimal. Veja os exemplos abaixo:

```
// Carros.cpp
#include <iostream.h>
void main()
{
 cout << "\n\n";
 cout << "\n \xDC\xDC\xDB\xDB\xDB\xDB\xDC\xDC";
 cout << "\n \xDFO\xDF\xDF\xDF\xDFO\xDF";
 cout << "\n\n";
 cout << "\n \xDC\xDC\xDB \xDB\xDB\xDB\xDB\xDB\xDB";
 cout << "\n \xDFO\xDF\xDF\xDF\xDF\xDFOO\xDF";
 cout << "\n\n";
}
```

```
// Quadrado.cpp
#include <iostream.h>
void main()
{
 cout << "\n\n\n\xC9\xCD\xBB\n\xBA\x20\xBA\n\xC8\xCD\xBC";
}
```