2.1 Programas:

2.2 O primeiro programa clássico:

* Para fazer um computador realizar alguma coisa, você (ou outra pessoa) tem que dizer para ele exatamente – com detalhes minuciosos – o que ele deve fazer. Tal descrição de “o que fazer” é chamada de programa;

programação é a atividade de escrever e testar tais programas.

* Em relação aos humanos, os computadores são estúpidos e precisam de detalhes minuciosos para executar alguma tarefa.
* Para poder descrever precisamente “coisas” para um computador, precisamos de uma linguagem definida precisamente, com uma gramática específica e um vocabulário bem definido para os tipos de ações que queremos realizar. Tal linguagem é chamada de linguagem de programação e C++ é uma linguagem de programação projetada para uma gama de tarefas de programação.
* Um cursor é um pequeno caractere ou linha piscante que mostra onde você pode digitar o próximo caractere.
* Em C++, cadeias de caracteres literais são delimitadas por aspas (");
* O \n é um “caractere especial” que indica uma nova linha.
* O nome cout se refere ao fluxo de saída padrão. Os caracteres “colocados no cout” usando o operador de saída << irão aparecer na tela;
* O nome cout é pronunciado “see-out” e é uma abreviatura de “character output stream”.
* Abreviaturas são comuns em programação, pois são essenciais para manter o texto do programa curto e fácil de administrar.
* Qualquer coisa escrita após o símbolo // (que é o caractere /, chamado de ”barra”, duplicado) em uma linha é um comentário. Comentários são ignorados pelo compilador e escritos para auxiliar os programadores que lêem o código.
* Um programa é escrito para dois públicos. Naturalmente, escrevemos o código para ser executado por computadores e principalmente ser entendidos pelos leitores humanos. Não esqueça: código é para ser lido – faça tudo o que você puder para torná-lo legível. De qualquer modo, os comentários são somente para auxiliar os leitores humanos; o computador não examina o texto em comentários.
* #include "std\_lib\_facilities.h" é uma diretiva “#include”. Ela instrui o computador a tornar disponível (“incluir”) recursos de um arquivo chamado std\_lib\_facilities.h.
* Um arquivo incluído com #include geralmente tem o sufixo .h e é chamado de cabeçalho (header) ou arquivo de cabeçalho. Um cabeçalho contém definições de termos, tais como cout.
* Como um computador sabe onde iniciar a execução de um programa? Ele procura por uma função chamada main e inicia executando as instruções que nela encontra.
* Cada programa C++ deve ter uma função chamada main para indicar onde inicia a execução. Uma função é basicamente uma sequência de instruções com um nome, que o computador executa na ordem em que estão escritas. Uma função tem quatro partes:

• Um tipo de retorno, aqui int (significando “inteiro”), que especifica o tipo do resultado, se houver, que a função irá devolver para qualquer um que solicitar a sua execução. A palavra int é uma palavra reservada em C++ (uma palavra-chave) e portanto int não pode ser usada como nome de mais nada (ver §A.3.1).

• Um nome, aqui main.

• Uma lista de parâmetros entre parênteses (ver §8.2 e §8.6), aqui (); neste caso, a lista de parâmetros está vazia.

• Um corpo de função delimitado por um conjunto de “chaves” { }, que relaciona as ações (denominadas comandos) que a função deve executar. Portanto, o menor programa em C++ é simplesmente

int main() { }

Isso, porém, não tem muita serventia, porque não faz nada.

* Um zero (0) devolvido por main() indica que o programa terminou com sucesso.
* Uma parte de um programa C++ que especifica uma ação e não é uma diretiva #include (ou alguma outra diretiva de pré-processador; ver §4.4 e §A.17) é denominada de comando.

2.3 Compilação:

* C++ é uma linguagem compilada. Isto significa que para obter um programa executável, você deve primeiro traduzi-lo de uma forma legível por seres humanos para algo que a máquina possa “entender.” Esta tradução é feita por um programa denominado compilador.
  + Aquilo que você lê e escreve é denominado código fonte ou texto do programa, e aquilo que o computador executa é denominado executável, código objeto ou código de máquina. Normalmente os nomes dos arquivos com código fonte de C++ recebem o sufixo .cpp ou .h, ao passo que os arquivos de código objeto recebem o sufixo .obj (em Windows) ou .o (Unix).
  + A simples palavra código é, portanto, ambígua e pode causar confusão; use-a com cuidado apenas onde é óbvio o que ela significa. A menos que especificado de outra maneira, usamos código para designar o “código fonte” ou até mesmo “o código fonte exceto os comentários”, porque comentários na verdade são apenas para seres humanos e não são examinados pelo compilador ao gerar o código objeto.
* Omitir qualquer detalhe em nosso programa, tal como um arquivo #include, um ponto e vírgula ou as chaves, irá causar erros. De modo similar, o compilador possui tolerância zero para erros de grafia.
* Note que muitos comandos de C++ terminam com um ponto e vírgula. O compilador precisa destes pontos e vírgulas para saber onde um comando termina e o seguinte inicia. Não existe uma maneira curta, correta e não técnica de resumir onde é necessário pôr ponto e vírgula.
  + “Coloque um ponto e vírgula depois de cada expressão que não termine com uma chave direita (})”.
* Lembre-se: o compilador é seu amigo; possivelmente, o compilador é o melhor amigo que você tem quando programa.

2.4 Ligação:

* Um programa geralmente consiste em diversas partes separadas, muitas vezes desenvolvidas por pessoas diferentes.
* O código objeto e os executáveis não são portáveis entre sistemas. Por exemplo, quando você compila para executar em uma máquina Windows, você obtém código objeto para Windows que não vai poder ser executado em uma máquina Linux.
* Erros encontrados pelo compilador são chamados de erros de compilação, erros encontrados pelo ligador são chamados de erros de ligação e erros não encontrados até que o programa seja executado são chamados de erros de execução ou erros de lógica. Geralmente, erros de compilação são mais fáceis de entender e corrigir do que erros de ligação, e erros de ligação são quase sempre mais fáceis de encontrar e corrigir do que erros de execução e erros de lógica.

2.5 Ambientes de programação:

* Para programar, usamos uma linguagem de programação. Também usamos um compilador, para traduzir nosso código fonte para código objeto, e um ligador para ligar nosso código objeto a fim de obtermos um programa executável. Além disso, usamos algum programa para digitar o texto do código fonte no computador e editá-lo. Estas são apenas as primeiras e mais críticas ferramentas que constituem o conjunto de ferramentas do programador ou “ambiente de desenvolvimento de programas.”
* IDEs geralmente incluem um editor com recursos úteis, tais como colorir o código para ajudar a distinção entre comentários, palavras-chave e outras partes do código fonte de seu programa; há ainda outros recursos que ajudam a depurar, compilar e executar o código. A depuração é a atividade de encontrar erros em um programa e removê-los;

Cap3

* Para ler algo, precisamos de algum lugar para onde enviar a leitura; ou seja, algum lugar na memória do computador para colocar o que lemos. Chamamos um “lugar” assim de objeto.

Um objeto é uma região da memória com um tipo que especifica a espécie de informação que pode ser colocada nela.

* Um objeto com um nome é chamado de variável.
* Prompt - É uma mensagem que lembra o usuário de executar uma ação.
* Definição – É um comando que introduz um novo nome em um programa e reserva memória para uma variável.
* cin (de “character input”) se refere ao fluxo padrão de entrada definido na biblioteca padrão. O segundo operando do operador >> (“obter de”) especifica para onde a entrada vai.
* A nova linha é necessária para chamar a atenção da máquina. Até que uma nova linha seja inserida (pressionando a tecla Enter), o computador simplesmente coleta o tipo de dado inserido.
* Usamos aspas quando queremos uma cadeia de caracteres literal. Quando não usamos aspas, usamos um nome para nos referirmos ao valor de algo.
* Os “lugares” nos quais armazenamos dados são chamados de objetos.
* Para acessar um objeto necessitamos de um nome. Um objeto com um nome é chamado de variável e possui um tipo específico que determina o que pode ser colocado no objeto e que operações podem ser aplicadas a ele (por ex., podemos multiplicar ints usando o operador \* e comparar strings usando o operador <=).
* Os itens de dados que colocamos em variáveis são chamados de valores.
* Um comando que define o que uma variável é denomina-se de definição e a definição pode (e geralmente deve) fornecer um valor inicial.
* double é uma abreviatura para “double precision floating point” (ponto flutuante com precisão dupla). O ponto flutuante é uma aproximação do computador para o conceito matemático de número real.

A.2

* Literais representam valores de tipos diversos. Por exemplo, o literal 12 representa o valor inteiro “doze”, "manhã" representa o valor do string de caracteres manhã e true representa o valor booleano true.

A.2.1 Literais inteiros

* Literais inteiros vêm em três variedades:

• Decimal: uma série de dígitos decimais

Dígitos decimais: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9

• Octal: uma série de dígitos octais iniciando com 0

Dígitos octais: 0, 1, 2, 3, 4, 5, 6 e 7

• Hexadecimal: uma série de dígitos hexadecimais iniciando com 0x ou 0X

Dígitos hexadecimais: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, A, B, C, D, E e F

* Um sufixo u ou U torna um literal unsigned (sem sinal) (§25.5.3) e um sufixo l ou L torna-o long; por exemplo, 10u e 123456UL.

A.2.2 Literais de ponto flutuante

* Um literal de ponto flutuante contém um ponto decimal (.), um expoente (por ex., e3) ou um sufixo de ponto flutuante (d ou f ). Por exemplo:
  + 123 // int (sem ponto decimal, sufixo ou expoente)
  + 123. // double: 123.0
  + 123.0 // double
  + .123 // double: 0.123
  + 0.123 // double
  + 1.23e3 // double: 1230.0
  + 1.23e–3 // double: 0.00123
  + 1.23e+3 // double: 1230.0
* Literais de ponto flutuante têm o tipo double, a menos que o sufixo indique outra coisa. Por exemplo:
  + 1.23 // double
  + 1.23f // float
  + 1.23L // long double

A.2.3 Literais booleanos

* Os literais do tipo bool são true e false. O valor inteiro de true é 1 e o valor inteiro de false é 0.

A.2.4 Literais de caracteres

* Um literal de caractere é um caractere entre apóstrofes como, por exemplo, 'a' e '@'. Além disso, existem alguns “caracteres especiais”:

VEJA OS CARACTERES ESPECIAIS NO BLOCO DE NOTAS “meuGlossário”.

* Um caractere especial é representado por seu “nome C++” entre apóstrofes, como, por exemplo, '\n' (nova linha) e '\t' (tabulação).

A.2.5 Literais string

* Um literal string é uma série de caracteres entre aspas, como, por exemplo, "Knuth" e "King Canute". Uma nova linha não pode ser parte de um string; em vez disso, use o caractere especial \n para representar nova linha em um string.

3.3 Entrada e tipo

* A operação de entrada >> (“obter de”) é sensível ao tipo; isto é, ela lê dados de acordo com o tipo de variável para a qual está sendo feita a leitura.
* Por convenção, a leitura de strings é encerrada pelo assim chamado espaço em branco, isto é, caracteres de espaço, nova linha e tabulação. Além disso, o operador >>, por default (padrão), ignora espaços em branco.
* Uma leitura de string usando >> é (por default) terminada por um espaço em branco; isto é, faz-se a leitura de uma única palavra.
* Simplesmente usamos >> duas vezes, uma para cada variável de leitura. Quando quisermos escrever os nomes, devemos inserir um espaço entre eles.

3.4 Operações e operadores

* Por “erro” queremos dizer que o compilador irá rejeitar um programa que tenta subtrair strings.
* O compilador sabe exatamente quais as operações que podem ser aplicadas a cada variável e pode, portanto, evitar muitos enganos. Entretanto, o compilador não sabe quais operações sobre quais valores fazem sentido para você, de modo que ele vai aceitar com alegria operações válidas que produzam resultados que podem parecer absurdos para você. Por exemplo:

int idade = –100;

* Para strings, o + significa concatenação; isto é, quando s1 e s2 são strings, s1+s2 é um string onde os caracteres de s1 são seguidos pelos caracteres de s2.

3.5 Atribuição e inicialização

* O operador de atribuição, representado por ’ = ’ atribui um novo valor a uma variável.

• Inicialização (dar a uma variável seu valor inicial)

• Atribuição (dar a uma variável um novo valor)

Exemplo:

int y = 8; // inicializar y com 8

x = 9; // atribui 9 a x

string t = "Alô!"; // inicializa t com "Alô!"

s = "Bom dia"; // atribui "Bom dia" para s

- Entretanto, atribuição e inicialização são logicamente diferentes.

- Em princípio, uma inicialização sempre encontra a variável vazia. Por outro lado, uma atribuição (em princípio) deve limpar o valor antigo da variável antes de colocar o novo valor.

OBS: - Você pode pensar sobre uma variável como uma espécie de pequena caixa e o valor como uma coisa concreta, como uma moeda, que você coloca dentro dela. Antes da inicialização, a caixa está vazia, mas depois da inicialização ela sempre conterá uma moeda, de modo que para colocar uma nova moeda nela (isto é, o operador de atribuição), você primeiro terá que remover a antiga (“destruir o valor antigo”) – e você também não poderá deixar a caixa vazia.

* Lembre-se de que, para um string, o operador >> lê palavras separadas por espaços em branco. Você termina este laço fornecendo ao programa um caractere de fim de entrada (geralmente chamado de fim de arquivo). Em uma máquina Windows, o fim de entrada é Ctrl+Z (Control e Z pressionados juntos) seguido de um Enter (return). Em uma máquina Unix ou Linux é Ctrl+D (Control e D pressionados juntos).

3.6 Operadores de atribuição compostos

* Incrementar uma variável (isto é, adicionar 1 a ela) é tão comum em programas, que C++ possui uma sintaxe especial para isso. Por exemplo:

++contador

significa

contador = contador + 1

* Podemos querer adicionar 7 a ela, subtrair 9 ou multiplicá-la por 2. Tais operações são diretamente suportadas por C++. Por exemplo:

a += 7; // significa a = a+7

b –= 9; // significa b = b–9

c \*= 2; // significa c = c\*2

* Em geral, para qualquer operador binário op, a op= b significa a = a op b (§A.5).
* Para iniciantes, esta regra nos dá os operadores +=, –=, \*=, /= e %=. Isto fornece uma notação agradavelmente compacta que reflete diretamente as nossas ideias. Por exemplo, em muitos domínios de aplicação /= e %= são referidos como “escalonar”.

3.7 Nomes

* Damos nomes às nossas variáveis para que possamos lembrar-nos delas e a elas nos referir em outras partes de um mesmo programa.
* O que pode ser um nome em C++? Em um programa C++, um nome inicia por uma letra e contém somente letras, dígitos e o caractere sublinhado.

Os exemplos seguintes não são nomes:

* + 2x // um nome deve iniciar por uma letra
  + tempo$de$vender // $ não é letra, dígito nem sublinhado
  + Menu Iniciar // espaço não é letra, dígito nem sublinhado
* Quando dizemos “não são nomes”, isto significa que um compilador C++ não vai aceitá-los como nomes.
* Evite sublinhados no início pois você nunca vai ter seus nomes coincidindo com algum nome que a implementação gerou.
* Nomes são sensíveis a maiúsculas e minúsculas; isto é, letras maiúsculas e minúsculas são consideradas distintas, de modo que x e X são nomes diferentes.
* Ao escolher nomes para suas variáveis, funções, tipos, etc., escolha nomes significativos, nomes que ajudem as pessoas a entender o seu programa.
* Abreviaturas e siglas podem confundir as pessoas, portanto, use-as com parcimônia.
* Nomes curtos, tais como x e i, são significativos quando usados convencionalmente; isto é, x deveria ser uma variável local ou parâmetro (ver §4.5 e §8.4) e i deveria ser um índice de laço (ver §4.4.2.3).
* Não use nomes longos demais; eles são difíceis de digitar, tornam as linhas tão grandes que não cabem na tela e são difíceis de ler rapidamente.
* Nosso “estilo tradicional” usa sublinhado para separar as palavras em um identificador, tal como conta\_elementos, em vez de alternativas como contaElementos e Conta\_Elementos. Nunca usamos nomes com todas as letras maiúsculas, como TODAS\_AS\_LETRAS\_MAIUSCULAS, por que isto é convencionalmente reservado para uso em macros (§27.8 e §A.17.2), as quais evitamos.
* A linguagem C++ e a biblioteca padrão não usam letras iniciais maiúsculas, de modo que é int em vez de Int e string em vez de String.
* Evite nomes que são fáceis de errar na digitação, na leitura ou confundir. Por exemplo: Nome nomes nomeS

foo f00 fl

f1 fI fi

Os caracteres 0, o, O, 1, l e I são particularmente propensos a causar problemas.

3.8 Tipos e objetos

• Um tipo define um conjunto de valores possíveis e um conjunto de operações (para um objeto).

• Um objeto é um espaço de memória que contém um valor de um determinado tipo.

• Um valor é um conjunto de bits interpretados de acordo com um tipo.

• Uma variável é um objeto com um nome.

• Uma declaração é um comando que dá um nome a um objeto.

• Uma definição é uma declaração que reserva memória para um objeto.

* Uma caixa string pode conter valores de cadeias de caracteres, como "Interoperabilidade", "tokens: !@#$%^&\*" e "Quem tem medo do lobo mau?".
* A representação de um string é um pouco mais complicada do que a de um int, porque um string contabiliza o número de caracteres que armazena.
* As apóstrofes e aspas usadas para caracteres e cadeias de caracteres literais não são armazenadas.
* Todos os int possuem o mesmo tamanho; isto é, o compilador reserva a mesma quantidade fixa de memória para cada int. Em um computador de mesa típico, esta quantidade é 4 bytes (32 bits). De modo similar, bools, chars e doubles possuem tamanho fixo. Você vai descobrir que geralmente um computador de mesa usa um byte (8 bits) para um bool ou um char e 8 bytes para um double.
* Pense desta maneira: a memória do computador não sabe nada sobre tipos; é apenas memória. Os bits de memória ganham significado somente quando decidimos como a memória deve ser interpretada. Isto é similar ao que fazemos quando usamos números. O que significa 12,5? Não sabemos. Poderia ser R$ 12,50 ou 12,5 cm ou 12,5 litros. Somente quando fornecemos a unidade é que a notação 12,5 passa a significar algo.
* Um bit é uma unidade de memória de computador que pode conter um valor 0 ou 1. Para o significado de números binários, veja §A.2.1.1.

3.9 Segurança quanto a tipo

* Cada objeto, assim que é definido, é associado a um tipo. Um programa – ou uma parte de um programa – é seguro no que diz respeito aos tipos quando os objetos são usados somente de acordo com as regras de seu tipo.
* Infelizmente, existem maneiras de realizar operações que não são seguras quanto a tipo. Por exemplo, usar uma variável antes de ela ter sido inicializada não é considerado seguro quanto a tipo.
* Uma implementação pode até mesmo provocar um erro de hardware quando o x não inicializado é usado. Sempre inicialize suas variáveis!
* A segurança total dos tipos é o ideal e, portanto, a regra geral para a linguagem.
* Infelizmente, um compilador C++ não pode garantir a segurança total dos tipos, mas podemos evitar violações de segurança de tipos por meio de boas práticas de codificação e verificação durante a execução do programa. O ideal é nunca usar características da linguagem cuja segurança o compilador não possa assegurar: segurança estática de tipos.
* A solução óbvia, que seria o compilador gerar implicitamente código para verificar violações de segurança de tipos, está além de C++. Quando decidimos fazer coisas que não são seguras (quanto a tipo), nós mesmos devemos fazer a verificação.

3.9.1 Conversões seguras

* Quando necessário, um char é convertido para um int e um int é convertido para um double. Por exemplo:
  + char c = 'x';
  + int i1 = c;
  + int i2 = 'x';

- Aqui, tanto i1 quanto i2 ficam com o valor 120, que é o valor inteiro do caractere 'x' no conjunto mais popular de caracteres de 8 bits, o ASCII. Esta é uma maneira simples e segura de obter a representação numérica de um caractere.

- Chamamos isso de conversão segura de char para int porque nenhuma informação é perdida; isto é, podemos copiar o int resultante de volta para um char e obter o valor original

* Assim, tendo em conta que um valor é sempre convertido para um valor igual ou (para doubles) para a melhor aproximação de um valor igual – essas conversões são seguras:
  + bool para char
  + bool para int
  + bool para double
  + char para int
  + char para double
  + int para double
* Quando se trata de um int realmente grande, podemos (em alguns computadores) sofrer uma perda de precisão ao converter para double. Este é um problema raro.

3.9.2 Conversões inseguras

* Infelizmente, C++ também permite conversões inseguras (implícitas). Por insegura, queremos dizer que um valor pode ser implicitamente convertido para um valor de outro tipo que não é igual ao valor original.
  + double para int
  + double para char
  + double para bool
  + int para char
  + int para bool
  + char para bool

- são aceitas pelo compilador, mesmo que sejam inseguras. Elas são inseguras no sentido de que o valor armazenado pode diferir do valor atribuído. Por que isso pode ser um problema? Porque muitas vezes não suspeitamos que uma conversão insegura esteja ocorrendo.

* Conversões double para int truncam (sempre arredondam para baixo; em direção a zero) em vez de usar o arredondamento 4/5 convencional.
* Conversões de int para char não têm problemas com truncamento – nem int nem char podem representar uma fração de um inteiro. Entretanto, um char pode armazenar somente valores pequenos.
* Em um PC, um char é 1 byte, enquanto que um int ocupa 4 bytes.

Portanto, não podemos colocar um número grande, como 1000, em um char sem perda de informação: o valor é “estreitado”. Por exemplo:

int a = 1000;

char b = a; // b se torna –24 (em algumas máquinas)