Journal of Engineering and Technology for Industrial Applications, 2018. Edition. 16.Vol: 04
https://www.itegam-jetia.org
ISSN ONLINE: 2447-0228
DOI: https://dx.doi.org/10.5935/2447-0228.20180086

Paradigm of Object Orientation in the C ++ Programming Language

João Carlos Silva de Oliveira[1]

[1] Mestre em Engenharia de Processos. Docente do Centro Universitário do Norte – UNINORTE - *Laureate Universites*, Brasil, Rua Igarapé de Manaus, 211 – Centro, Manaus/AM.

Email: jcjunior182@gmail.com

Received: November 14th, 2018

Accepted: November 30th, 2018

Published: December 31th, 2018

ABSTRACT

The article aims to identify and describe in a summarized and analytical way the roots of the programming language currently denominated as C ++, so its main features, practical applications and factors relevant to its success. In order to respond to the research objectives, the option of methodological choice was an explanatory and descriptive research, approaching and exemplifying C ++ resources and evolutions in relation to C, being possible to attribute factors that influenced C ++ success trajectory, such as portability and the ability to solve problems at various levels of complexity.

Keywords: C ++, Programming Language, Object Oriented Programming

Paradigma da Orientação a Objetos na Linguagem de Programação C++

RESUMO

O artigo tem como objetivo identificar e descrever, de modo sintetizado e analítico, as raízes da linguagem de programação atualmente denominada como C++, assim como seus principais recursos, aplicações práticas e fatores relevantes para seu sucesso. Para responder aos objetivos da pesquisa, a opção de escolha metodológica foi de uma pesquisa explicativa e descritiva, abordando e exemplificando recursos e evoluções do C++ em relação ao C, sendo possível atribuir fatores que influenciaram a trajetória de sucesso do C++, tais como a portabilidade e a capacidade de resolver problemas em diversos níveis de complexidade.

Palavras-chave: C ++, Linguagem de Programação, Programação Orientada a Objetos

I. INTRODUÇÃO

Atualmente existem diversas linguagens de programação, cada uma adaptada a um determinado tipo de processo. A definição da linguagem de programação, na prática, deve estar condicionada ao tipo de solução que se deseja apresentar para um determinado problema. Para um programador é extremamente relevante compreender os fundamentos e técnicas da programação a serem utilizadas, seja ela de tecnologia web, mobile ou desktop.

Um dos objetivos de se criar uma linguagem de programação é que haja uma maior produtividade por parte dos programadores, possibilitando que suas intenções sejam expressas de formas mais fáceis [6].

Linguagem de programação pode ser definida como um método padronizado utilizado com o intuito de transmitir instruções para uma máquina que seja capaz de processá-las. Ou seja, é um tipo de linguagem utilizada pelo homem para se comunicar com a máquina, pois essa não reconhece e nem processa o dialeto normal do ser humano.

Para que essa relação pudesse então evoluir, tornou-se indispensável a criação de uma linguagem que tornasse a máquina operacional. A partir do desenvolvimento da linguagem de programação é que o homem passou a obter grandes resultados por meio da computação.

Linguagem de programação, pode ser entendido ainda como o conjunto de palavras, lexemas classificados em tokens, compostos de regras, que constituem o código fonte de um programa. Esse código é então traduzido e executado pelo microprocessador [4].

A linguagem C/C++, considerada de alto nível, é uma das mais bem sucedidas, considerada por muitos pesquisadores como uma das linguagens mais utilizadas de todos os tempos, por ser estruturalmente simples e de grande portabilidade.

Em pleno século XXI percebemos a necessidade em utilizar métodos que facilitem os processos de trabalho humano e a utilização dos sistemas de informação, em todas as áreas, nos direciona a minimizar falhas, produzindo soluções imediatas com sua eficiência e clareza de informações, sem o prejuízo da perda de dados [4].

Além de auxílio nos campos sociais e acadêmicos, a aplicação da Internet das Coisas (IoT) atrelado à domótica (automação residencial), como o uso da linguagem C/C++, tem auxiliado pessoas com necessidades especiais (PNEs). Essa união entre IoT e domótica gera inclusão digital e social, pois amplia cabalmente o potencial de auxílio e integração das PNEs com o meio em que vivem, a partir da interação com pessoas ou objetos através da internet.

A interatividade fomentada pelo avanço dos sistemas de informação, estimulou também o interesse na leitura e escrita. No campo social e acadêmico, essas novas formas de intercâmbios e inclusões, criaram níveis mais altos de comunicação, fortalecendo o desejo pela pesquisa e contribuindo para o estabelecimento e fortalecimento de relações interpessoais [2].

O interesse em desenvolver este artigo justifica-se pela necessidade de contribuir para a melhoria da obtenção de conhecimento e qualidade dos sistemas desenvolvidos em C/C++.

Por se tratar de uma linguagem que opera próximo ao hardware, existe uma dificuldade imensa em compreendê-la. Porém, em contrapartida, fornece um bom conhecimento de como o computador funciona, sendo extremamente relevante para se programar bem em qualquer linguagem.

Visando facilitar a obtenção do conhecimento científico e prático acerca desta tecnologia, de modo a se projetar novas estruturas de dados que atendam às demandas específicas, este artigo objetiva identificar e descrever, de modo sintetizado e analítico, as raízes da linguagem de programação atualmente denominada como C++, assim como seus principais recursos, aplicações práticas e fatores relevantes para seu sucesso.

II. REFERENCIAL TEÓRICO II.1

ORIGEM DA LINGUAGEM C++

A linguagem C++ é uma extensão de C, originada da linguagem B, escrita por Ken Thompson [5].

A linguagem C foi desenvolvida inicialmente nos laboratórios de Bell da American Telephone and Telegraph (AT &T) entre 1969 e 1973 [1].

Popularmente conhecido como o pai da linguagem C++, Bjarne Stroustrup, apresentou ao mundo seu grande projeto: uma linguagem forte capaz de resolver problemas complexos com seu vasto leque de recursos [1].

A ideia inicial de Stroustrup era implementar uma versão distribuída do Kernel da UNIX (Sistema Operacional Uniplexado), que era escrito a partir da linguagem C. Por isso, definiu C como base, de modo a evolui-lo, transformando-a numa linguagem numa mais completa e poderosa, o C++, que também englobou o ALGOL68(Algorithmic Language 19684), Ada, CLU e ML, em sua construção [3].

Ao longo dos anos, Stroustrup denominou a nova linguagem de "C com Classes", classes essas que, mais tarde, serviriam de arcabouço na programação orientada a objetos (POO). Em seguida, a linguagem foi denominada "C plus plus", atualmente sendo conhecida e utilizada como C++ [5].

Um pouco da complexidade do C++ foi herdada do C, como resultado do processo evolucionário, porém é complexa por que é poderosa e robusta, repleta de recursos fascinantes.

II.2 APLICAÇÕES EM C/C++

Dentre diversas aplicações da linguagem C/C++, talvez a mais interessante seja a sua utilização na construção de um dos Sistemas Operacionais mais conhecidos e utilizados em todo o mundo, o Microsoft Windows, que segundo informações da Statcounter, dominava mais de 80% do mercado, sendo superado em 2017 pelo Android [7].

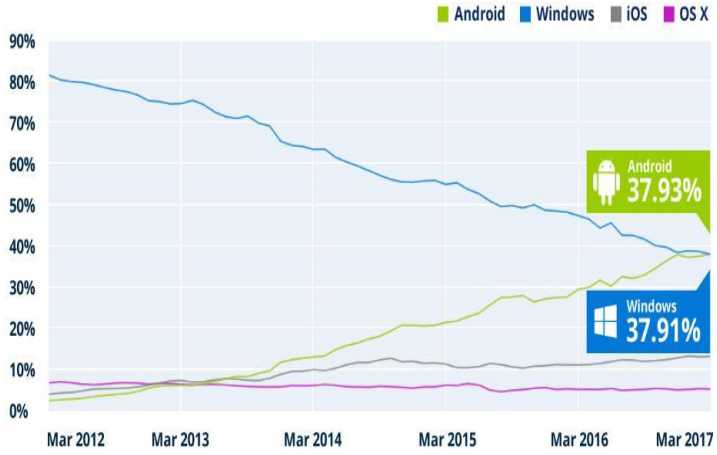

Figura 1: Sistemas operacionais mais utilizados.
Fonte: [7].

Dentre outros sistemas operacionais e ferramentas bastante conhecidas, que foram desenvolvidas utilizando a linguagem C/C++, podemos destacar: GNU-Linux, Mac OS-X, Adobe Photoshop, MySQL, Mozilla Firefox e Internet Explorer [10].

Alguns jogos e aplicações bastante conhecidas, tais como Doom 3, Half-Life, Counter-Strike, pacotes do Microsoft Office, CorelDRAW, sistemas integrados para satélites da NASA (National Aeronautics and Space Administration), assim como no Facebook (componentes de alta performance e credibilidade), foram criados com a utilização da linguagem C/C++ [1].

Uma das vantagens mais relevantes da linguagem C/C++ é a flexibilidade da equipe de desenvolvimento em adaptar o código-fonte sem ter que reeditá-lo.

II.3 PROPRIEDADES

O uso de classes, atrelado aos novos conceitos técnicos de programação orientada a objetos, foram as evoluções mais significativas da linguagem C++ [2].

A estrutura do C++ é composta por três principais pilares, sendo eles: polimorfismo, herança e encapsulamento [7].

II.3.1 POLIMORFISMO

Polimorfismo é a qualidade ou estado de ser capaz de assumir diferentes formas. Na POO, este princípio possibilita que referências de tipos de classes mais abstratas representem o comportamento das classes concretas as quais referenciam. Assim, é possível tratar vários tipos distintos de maneira homogênea [2].

O conceito de polimorfismo pode ser ilustrado através de animais, tais como cães e gatos. Embora sejam diferentes, compartilham algumas características: mamíferos, possuem quatro patas e pelos. Essas características também podem ser compartilhadas com porcos, cavalos e outros animais [7].

Logo, a essência do polimorfismo pode ser traduzida como algo com diferentes formas, possibilitando ações diferenciadas sobre o mesmo objeto [2].

II.3.2 HERANÇA

Herança é a capacidade da criação de classes com o intuito de se herdar estruturas de dados e funções definidas em outras classes, sendo possível adicionar ou redefinir novos elementos [6].

Além de possuir propriedades em comum, cada subclasse tem as suas especificações, tais como as superclasses de operações que são herdadas pelas subclasses e apresentam propriedade adicionais e peculiares [2].

II.3.3 ENCAPSULAMENTO

De forma resumida, o encapsulamento é uma propriedade que possibilita o agrupamento de dados num mesmo módulo, desde que haja uma relação. Ou seja, encapsular é capturar, na mesma classe, objetos que possuem relações de comportamento ou atributos genéricos [2].

É imprescindível entender o funcionamento do encapsulamento, que pode ser comparado a uma caixa preta. Uma vantagem interessante desta técnica é a segurança, pois protege os objetos de sofrerem modificações em seus atributos [8].

Desse modo, o encapsulamento torna-se uma das principais características de evolução da linguagem C++, que foi projetada para suportar a programação orientada a objetos com eficiência, portabilidade e flexibilidade [7].

III. METODOLOGIA

III.1 MATERIAIS

Para o desenvolvimento deste artigo utilizou-se a linguagem de programação C++.

Para que o código-fonte fosse escrito, editado e exibido através de figuras, utilizou-se o ambiente integrado de desenvolvimento (IDE): Dev C++, versão 5.11. Trata-se de uma IDE simples e intuitiva, que usa a porta Mingw do GCC (GNU Compiler Collection) como seu compilador, criando executáveis nativas no Win32, console ou GUI.

III.2 MÉTODOS

Para responder aos objetivos da pesquisa, a opção de escolha metodológica foi de uma pesquisa explicativa e descritiva.

III.2.1 PESQUISA EXPLICATIVA

A pesquisa explicativa tem a característica de registrar fatos, analisando-os, interpretando-os e identificando as suas causas. Esse modelo visa ampliar generalizações, estruturar e definir modelos teóricos, relacionando hipóteses numa visão mais unitária no âmbito produtivo, gerando hipóteses ou ideais por força de dedução lógica [4].

A pesquisa explicativa exige maior investimento em síntese, reflexão e teorização a partir do objeto de estudo. Objetiva identificar os fatores que contribuem para a ocorrência dos fenômenos ou variáveis que afetam o processo, ou seja, explica o porquê das coisas.

III.2.2 PESQUISA DESCRITIVA

Na pesquisa descritiva, realiza-se o estudo, análise, registro e interpretação dos fatos sem a interferência do pesquisador [10].

IV. RESULTADOS E DISCUSSÕES

IV.1 PRELÚDIO DA TECNOLOGIA

IV.1.1 CLASSE

Uma das primeiras evoluções da linguagem C++ em relação à C é a construção *Class*. Pode se utilizar a construção *Class* para definir novos tipos de dados. A construção *Class* combina membros de dados e funções de membros em uma única unidade.

IV.1.2 INSERÇÃO DE COMENTÁRIOS

Outro aprimoramento da tecnologia C++ está no método de delimitação de entrada de comentários. Além dos comentários no estilo C, delimitados por /* e */, o C++ fornece o estilo de comentários //. Todo o código inserido depois do //, na mesma linha, são ignoradas pelo compilador, como se fizessem parte de um comentário, conforme pode ser visto no código da figura 2:

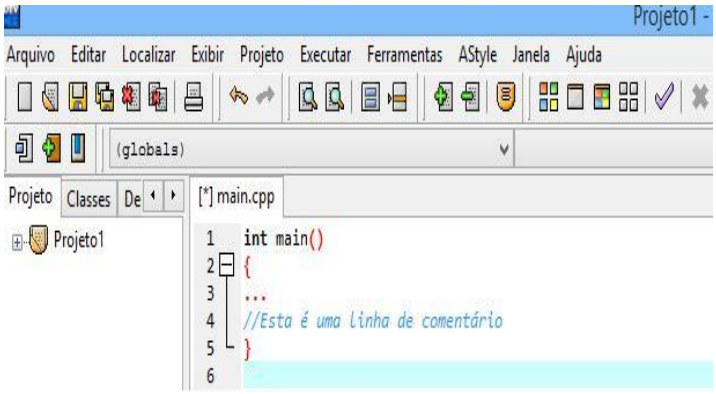

Figura 2: Delimitação de entrada de comentários.

O código C++ possui uma mistura dos dois tipos de comentários. Explicações em várias linhas são geralmente delimitadas por pares de comentários /* e */, enquanto os inseridos em linha única são delimitados por barras duplas //.

IV.1.3 OPERADOR DE RESOLUÇÃO DE ESCOPO

O C++ define o operador de resolução de escopo (::) para acessar uma variável oculta dentro de um escopo. Assim, cada varável definida em um programa tem um escopo distinto e visível apenas em seu próprio escopo. Conforme pode ser visto no código da figura 3, o código atribui o valor 1 à variável global "*a*" e o valor 2 à variável local "*a*".

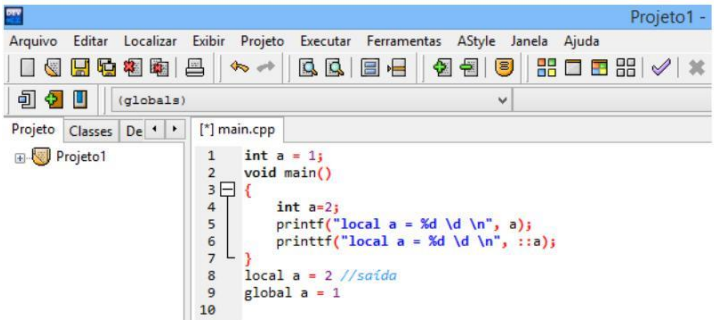

Figura 3: Operador de resolução de escopo.

IV.1.4 UTILIZAÇÃO DO MARCADOR STRUCT

Quando é impresso o valor de *a*, obtém-se valor distinto da variável *a*. Assim, o operador da resolução de escopo (::) recupera o valor global de *a*, que está oculto no escopo atual. Também pode ser utilizado o operador de resolução de escopo para declarar uma função de uma determinada classe. Por exemplo, a instrução *numlegs*(), implica uma função de membro da classe Animal: *Animal::numlegs*(), conforme pode ser visto no código da figura 4:

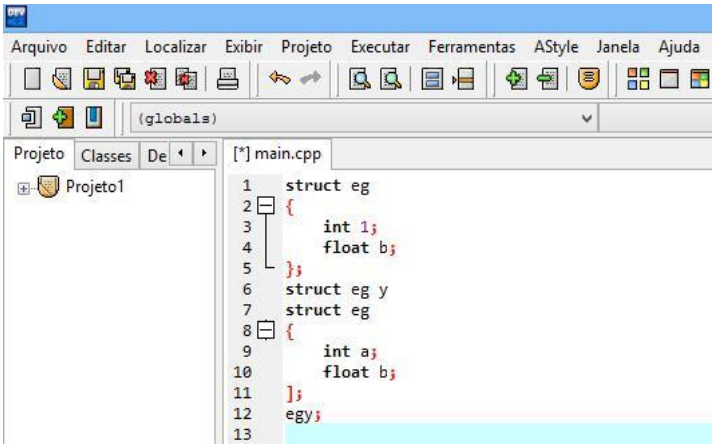

Figura 4: Recuperação de valor global da variável.

IV.1.5 USO DO ESPECIFICADOR CONST

Na linguagem de programação C++ pode-se utilizar *const* para declarar variáveis locais ou globais cujos valores permanecem fixos. O código da figura 5 exemplifica o uso de *const* em um programa C++:

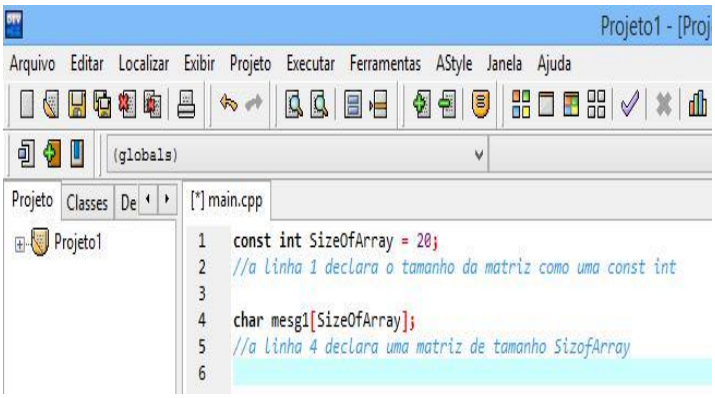

Figura 5: Uso do especificador const.

Pode-se definir um conjunto de constantes em um arquivo de cabeçalho e incluir esse arquivo em vários blocos do programa. Isso equivale a definir todas as constantes separadamente em cada arquivo de programa. O C++ trata uma definição *const* como uma definição estática por padrão. No segmente do código mostrado na figura 6, *NumOfDigits* vai reter seu valor entre chamadas de funções:

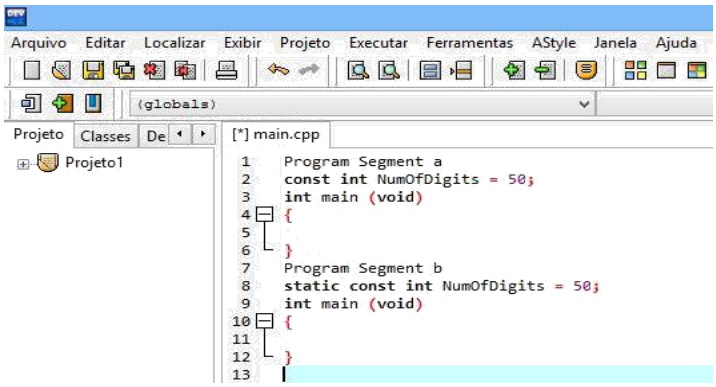

Figura 6: Uso do const como definição estática.

IV.1.6 ALOCAÇÃO DINÂMICA DA MEMÓRIA

Outra evolução do C++ em relação do C é a forma de como a memória é alocada às variáveis. No código em C, a alocação dinâmica de memória é tratada por funções de biblioteca, tais como *malloc*() e *free*(). Em contrapartida, o C++ usa operadores, ao invés de funções, para alocar dinamicamente memoria às variáveis.

Os operadores *new* e *delete* no C++ executam operações similares às funções *malloc*() e *free*(), respectivamente. Pode-se utilizar o operador *new* para alocar memória às variáveis e o *delete* para retirar a memória das variáveis.

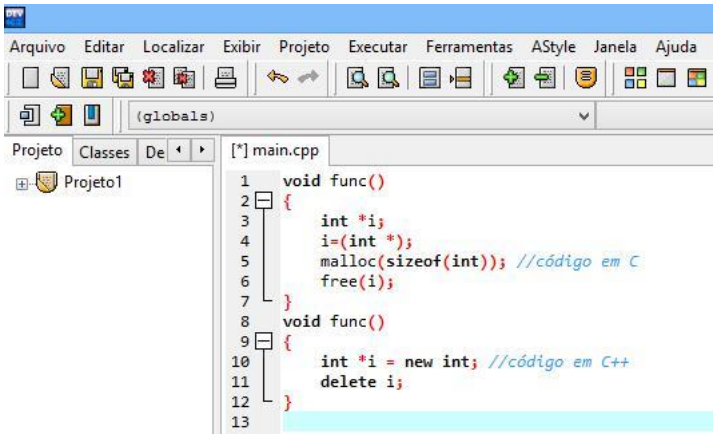

Figura 7: Alocação dinâmica de memória.

Definindo a alocação de memória, em termos de operadores em vez de funções, o C++ reduz a sobrecarga de chamadas de função e oferece um gerenciamento mais rápido de memória dinâmica.

IV.1.7 USO DE REFERÊNCIAS

O C++ utiliza referências para passar valores às funções. Uma referência é um nome alternativo de uma variável, sendo indicado através do uso do operador &, da mesma forma que um ponteiro é indicado usando-se o operador *. O trecho de código

abaixo, na figura 8, exemplifica como se pode declarar a variável *rodents* como uma referência para a variável *rats*.

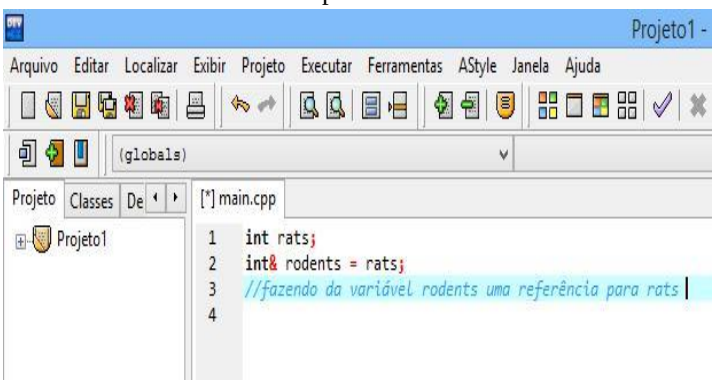

Figura 8: Uso de referências.

Utiliza-se uma referência na passagem de um argumento para uma função. Ao usar uma referência como um argumento, a função trabalha com os dados originais, em vez de uma cópia. Essa forma de passar argumentos para a função é conhecido como passagem por referência.

O C++ fornece uma biblioteca especial de classes, denominada biblioteca de fluxos, para tratar das operações de entrada e saída de programas C++. Essas classes são definidas no arquivo cabeçalho *iostream.h*.

Quando inclui-se o arquivo de cabeçalho *iostream.h* em um programa, o C++ cria de forma automática, quatro objetos para tratar das operações de entrada e saída. Os objetos *cin*, *cout*, *cerr* e *clog* são anexados aos dispositivos padrão de entrada e saída.

O C++ fornece os operadores para escrever dados no fluxo de saída e buscar dados no fluxo de entrada. O C++ fornece o operador de deslocamento à esquerda << para enviar caracteres para um fluxo de saída. O operador <<, quando usado para direcionar valores para a saída padrão, é conhecido como operador de inserção. O operador de inserção pode ser utilizado mais de uma vez em uma única instrução, para executar várias operações de saída. O exemplo pode ser observado na figura 9, quando tanto o nome da variável quanto o valor, são exibidos utilizando-se uma única instrução:

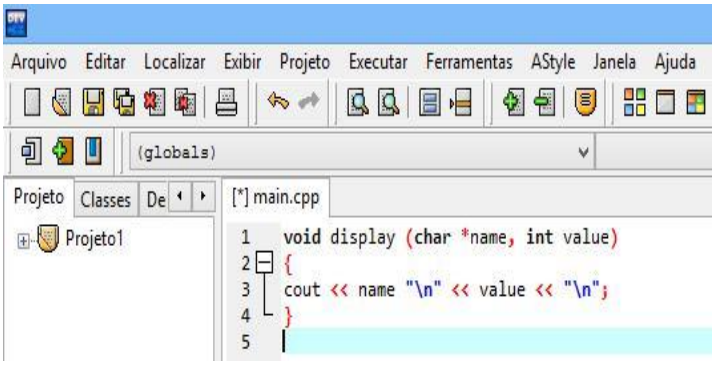

Figura 9: Uso de operadores.

O processo de busca de dados no fluxo de entrada é chamado de extração de fluxo. O C++ fornece o operador de deslocamento à direita >>, para buscar dados no fluxo de entrada.

O operador >>, quando usado para ler valores da entrada padrão, é conhecido como operador de extração. O trecho de código, conforme mostrado na figura 10, mostra o uso de >> com *cin*. O operador >> aceita uma cadeia de caracteres do teclado e armazena na variável nome.

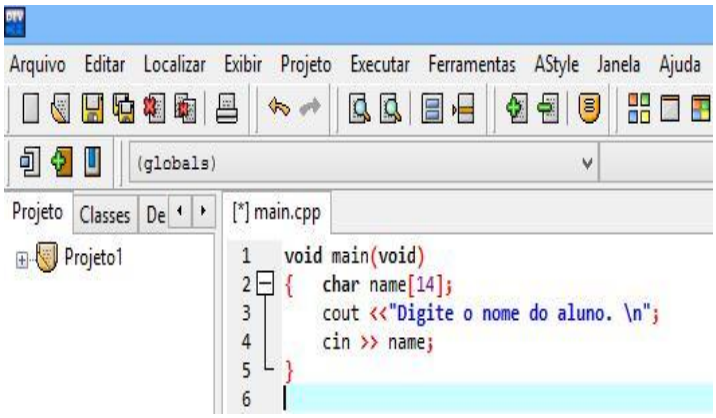

Figura 10: Busca de dados no fluxo de entrada.

IV.1.8 CONVERSÕES DE TIPOS

A conversão de tipo envolve a transmutação de uma variável de um tipo específico de dado para outro. Por exemplo, na linguagem C, pode-se atribuir um valor *int* a uma variável *long* ou adicionar um valor *long* a uma variável *float*.

Na linguagem C++, existem duas formas de conversão de tipos: implícita e explícita. As conversões implícitas são implementadas pelo compilador sem a intervenção do programador, em detrimento das explícitas, que dependem do programador.

Uma conversão implícita acontece quando tipos de dados distintos são misturados. Por exemplo, quando você atribui um valor *int* a uma variável *long*, o compilador faz uma conversão implícita. A figura 11 ilustra a conversão de tipo de dado *float* para o tipo *int*, assim como a saída do segmento. A variável *pi* foi declarada como um número inteiro. Atribuir um valor *float* à pi resulta na conversão de *float* para *int*.

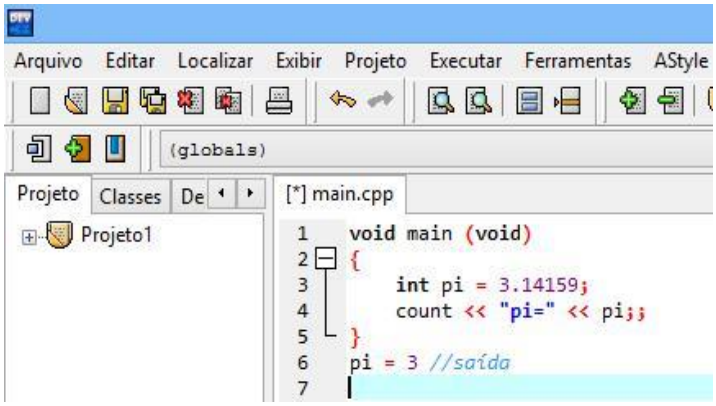

Figura 11: Conversões de tipos.

Uma conversão implícita também é possível de ser realizada nas classes, sendo obtida através de um construtor de conversão, que é uma função utilizada para inicializar elementos de dados de uma classe com valores padrão. Utilizando o construtor de conversão, o tipo de argumento é convertido para o tipo do objeto da classe. O trecho de código mostrado abaixo, exemplifica a conversão do um valor *int* em um valor do tipo *Complex*, definido pelo programador:

```
Complex::Complex (int i)
{
        real = (double) i;
```

```
    imag = 0.0;
}
```

Geralmente, qualquer construtor que leva um único argumento atua como um modelo para a conversão de um valor daquele tipo de argumento para o tipo da classe.

Por exemplo, o construtor *Stonew(int lbs)* atua como um modelo de conversão int para *Stownet*. Similarmente, *Stownet(double lbs)* atua como um modelo de conversão de *double* para *Stownet*, conforme pode ser visto no trecho de código da figura 12:

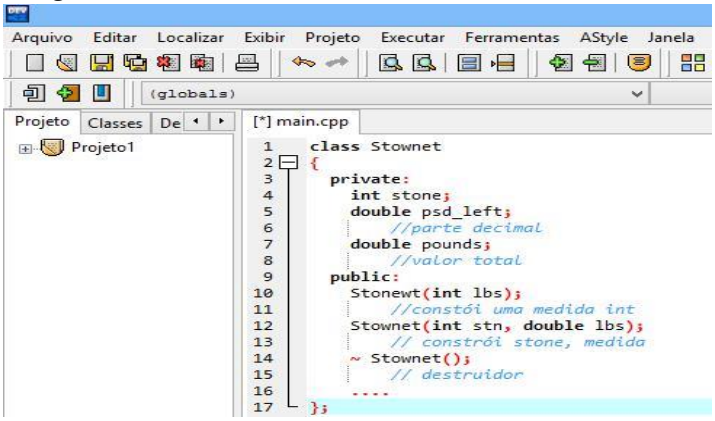

Figura 12: Conversões implícitas.

A conversão explícita é a transmutação forçada pelo programador, de um determinado tipo de dado para outro. A conversão explícita também é possível para classes, após a definição de uma função de conversão.

IV.2 RECURSOS DA LINGUAGEM C++

IV.2.1 STRUCT

O corpo da classe, na linguagem C++, é cercado por um par de chaves seguidas de um ponto e vírgula ou de uma lista de declarações terminada por ponto e vírgula. O método de definição de uma classe em C++ é diferente de struct em C. Antagonicamente à aplicação do struct em C, pode-se incluir funções na classe. A código da figura 13 ilustra a diferença do uso de struct em C e da classe em C++.

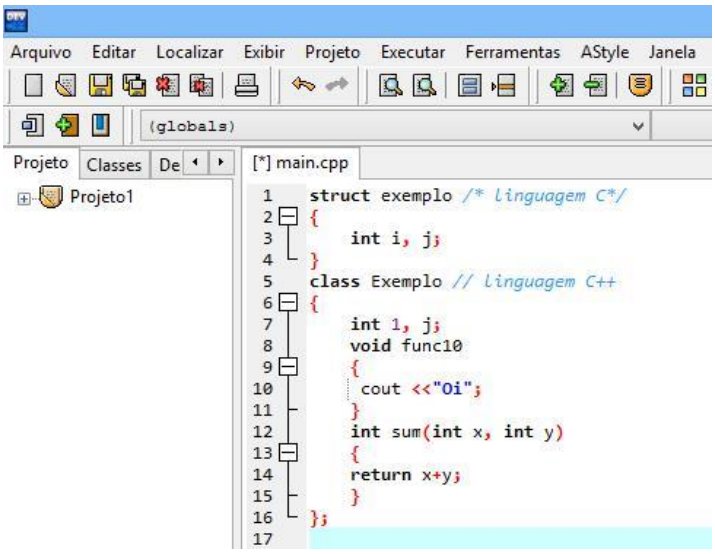

Figura 13: Uso de struct.

IV.2.2 CONSTRUTOR

No C++, quando um objeto é criado, os membros de dados precisam ser inicializados. A tecnologia permite que sejam ativados membros de dados de uma classe definindo-se um construtor. O construtor de uma classe é uma função de membro com o mesmo nome de uma classe.

O código da figura 14 ilustra a utilização do construtor para a classe *Circulo*. Quando se declara um objeto da classe *Circulo*, fornecem-se três valores para inicializar as coordenadas e raio do objetivo.

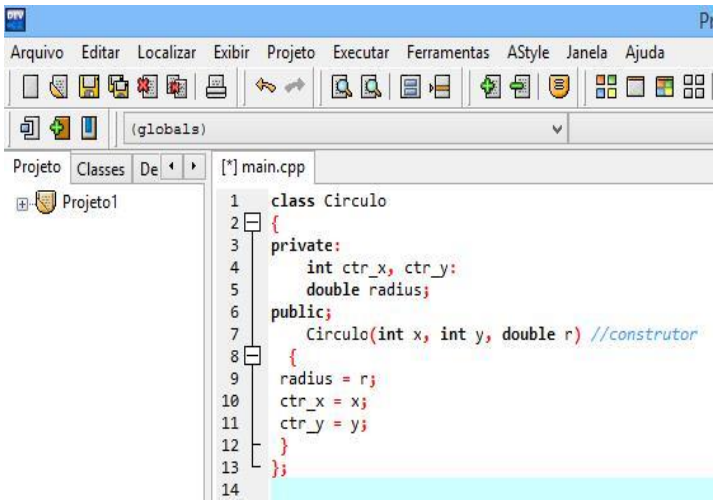

Figura 14: Uso de construtor.

Um construtor é declarado dentro da definição de classe. No entanto, o corpo do construtor pode ocorrer fora da definição de classe. Uma classe pode ter mais de um construtor.

Um construtor de classe é chamado quando um programa cria um objeto dessa classe. Uma classe derivada, quando não possuir membros de dados adicionais, terá um construtor de classe derivada com um corpo vazio. O compilador C++ gera um construtor padrão se não for definido construtor para a classe. Pode-se definir um construtor como uma função de membro pública, protegida ou privada de uma classe.

O C++ fornece duas formas de inicializar um objeto usando construtores: explícita e implícita. Quando um objeto é criado, o armazenamento necessário para conter os membros de dados definidos para a classe é alocado de forma automática. Os construtores servem apenas para inicializar o armazenamento recém alocado associado a um objeto de classe.

IV.2.3 DESTRUIDOR

Para excluir um objeto é necessário desativá-lo, podendo ser executado a partir de uma função especial chamada destruidor.

Podem-se utilizar destruidores para executar qualquer operação que o programador desejar implementar, como o último uso do objeto, que pode ser de salvar o conteúdo dele em disco.

Para declarar um destruidor, atribui-se o mesmo da classe a adiciona um *til* (~) ao nome. Um destruidor não aceita argumento, pois ele desaloca a memória do objeto inteiro. O código na figura 15 ilustra *~Boneco*() como destruidor de um objeto da classe *Boneco*.

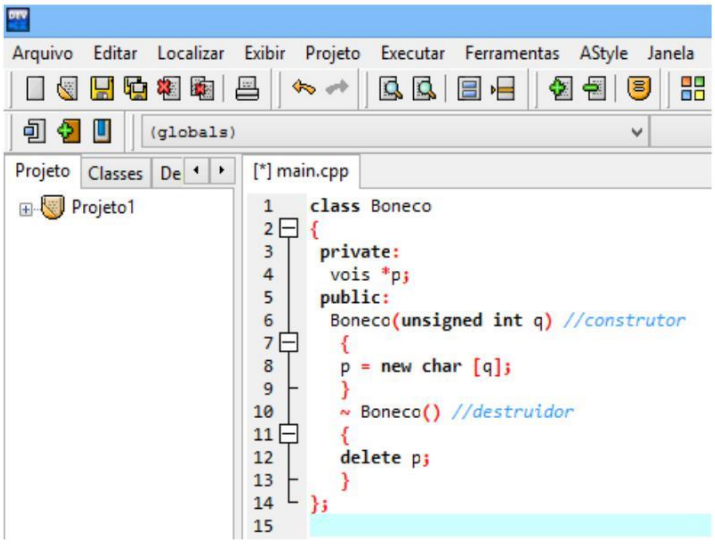

Figura 15: Uso de destruidor.

Uma classe pode ter apenas um destruidor. Assim como um construtor, um destruidor não tem tipo de retorno.

Quando um objeto é criado na memória, os membros de dados são inicializados pelo construtor. Cada objeto mantém seu próprio conjunto de membros de dados. No entanto, as instâncias de objetos não carregam uma cópia de todas as funções de membro dentro dela. Existe apenas uma cópia das funções de membro na memória.

IV.2.4 PONTEIRO THIS

Quando uma função de membro é invocada, todos os objetos chamam a mesma cópia da função de membro. A função de membro usa o ponteiro *this* para identificar o objeto que está invocando a função. O ponteiro *this* é criado automaticamente pelo compilador, não sendo necessário declarar o ponteiro *this* explicitamente.

O ponteiro *this* ajuda o compilador na identificação do objeto ao qual um membro de dado específico está associado. Quando você cria os objetos de classe *eg1* e *eg2*, um ponteiro *this* separado fica associado aos membros de ambos os objetos. Quando você atribui um valor à variável *i*, o ponteiro *this* transmite para o compilador que esse *i* é um membro do objeto *eg1* (ou *eg2*) da classe *Exemplo*.

O compilador C++ gera de forma automática um ponteiro *this* para cada função de membro de uma classe. O ponteiro *this* é declarado implicitamente para todas as classes, conforme pode ser visto abaixo:

*<nome da classe> *this;*

IV.2.5 ESPECIFICADORES DE ACESSO

Especificadores de acesso controlam o acesso aos membros de dados às funções de membro de uma classe. No C++ existem três tipos de especificadores de acesso:

- Public;
- Private;
- Protected.

Os membros de dados e as funções de membros listados na seção public de uma classe podem ser acessados por todas as classes do programa. No trecho de código da figura 16, os membros de dados *legs* e *color* são declarados *public*, podendo ser acessados por todas as classes do programa.

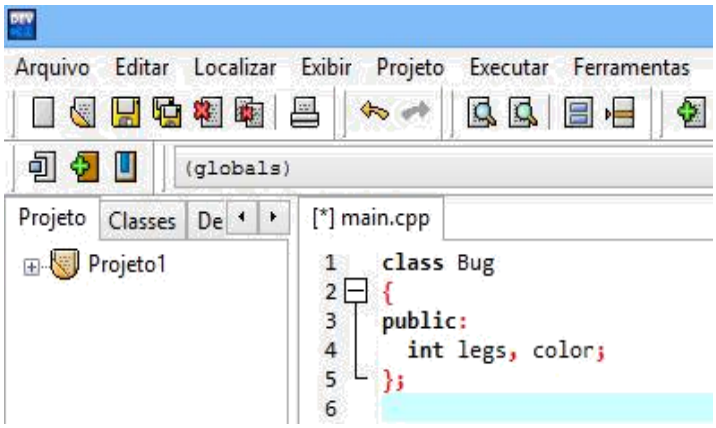

Figura 16: Uso de especificadores de acesso.

Os membros de dados e as funções de membros listados na seção *private* de uma classe podem ser acessados apenas pela própria classe. O membros de dados e as funções de membros listadas na seção *protect* de uma classe podem ser acessados apenas pela classe e por suas derivadas.

IV.2.6 DEFINIÇÃO DE ESCOPO

Escopo é a parte de um programa onde uma determinada variável torna-se acessível. Uma variável tem um escopo associado que, juntamente com o nome da variável, a identifica de forma única.

O escopo de uma variável se estende entre o ponto de declaração da variável e a chave de fechamento mais próxima dentro da qual a variável é declarada.

No C++ é possível definir três tipos de escopo: local, arquivo e classe.

- Escopo local

Variáveis definidas dentro de uma função ou um bloco de {} tornam-se acessíveis dentro desta função ou bloco, tendo um escopo local. Variáveis quem têm escopo local são também chamadas de variáveis automáticas porque são criadas automaticamente dentro do escopo e deixam de existir quando o escopo termina. Variáveis locais são armazenamentos alocados no momento em que uma função é invocada.

Para reter o valor de uma variável local, pode-se definir a variável como estática, concedendo à mesma um valor inicial. Quando se declara uma variável como estática, os dados são inicializados antes da execução do programa e o valor é retido entre as chamadas de funções.

- Escopo de Arquivo

O segundo tipo de escopo, em C++, é denominado escopo de arquivo. Pode-se acessar variáveis com escopo de arquivo em todo o código do programa. Elas são chamadas de variáveis globais. Quando uma variável local possui o mesmo nome de uma global, a global torna-se oculta pela local.

Para acessar o valor da variável oculta no escopo atual, utiliza-se o operador de resolução de escopo. O código da figura 17 ilustra como o operador de resolução de escopo ajuda a acessar o valor global de *r*.

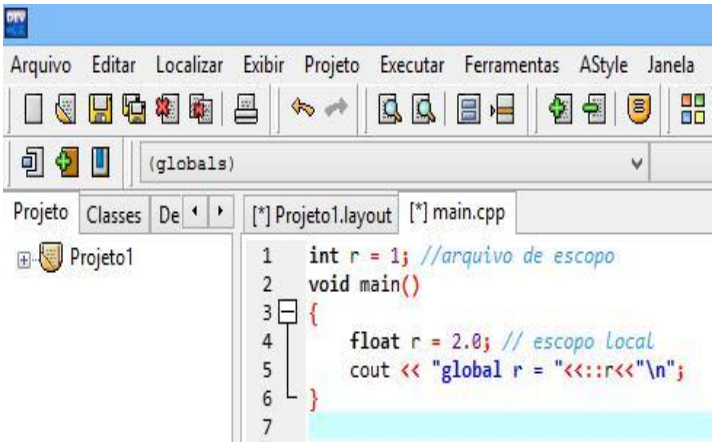

Figura 17: Escopo de arquivo.

- Escopo de Classe

As variáveis quem têm escopo de classe são visíveis dentro da classe, mas não fora. Pode-se usar membros de classe com o mesmo nome em outras classes sem que haja conflito. Variáveis definidas em uma classe têm um escopo de classe, a menos que os membros sejam declarados como *public*.

Se uma variável com escopo de arquivo tiver o mesmo nome que um membro de classe, então a variável com escopo de arquivo não será acessível dentro da classe.

O trecho de código da figura 18 mostra a variável de classe *height* sendo declarada depois de estar sendo utilizada. O construtor inicializa a variável *height* declarada para a classe e não a variável global *height*:

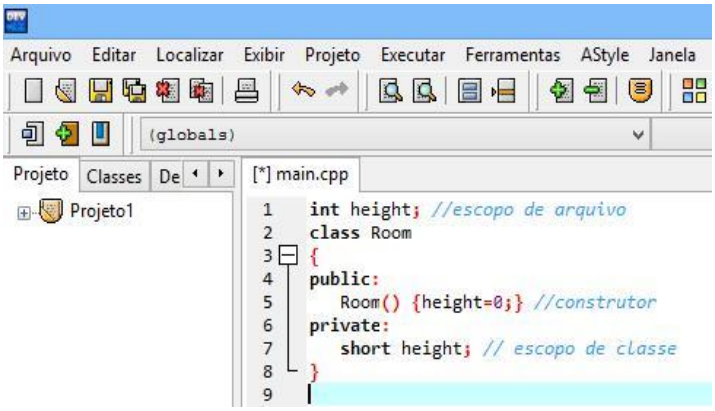

Figura 18: Escopo de Classe.

IV.3 FUNÇÕES ESPECIAIS E SOBRECARGAS

Funções especiais podem auxiliar no desenvolvimento de um programa, tornando o código mais funcional e flexível.

- Função Friend

A função *friend* é utilizada para acessar todos os membros da classe à qual a função tenha sido declarada como amiga. Em C++, uma função pode acessar as duas classes, mesmo que ela não seja membro. Isso ocorre quando se declara tal função como amiga das classes as quais ela deseja acessar. A função faculdade pode acessar todo os membros da classe *Estudante*, conforme pode ser visto no trecho de código da figura 18:

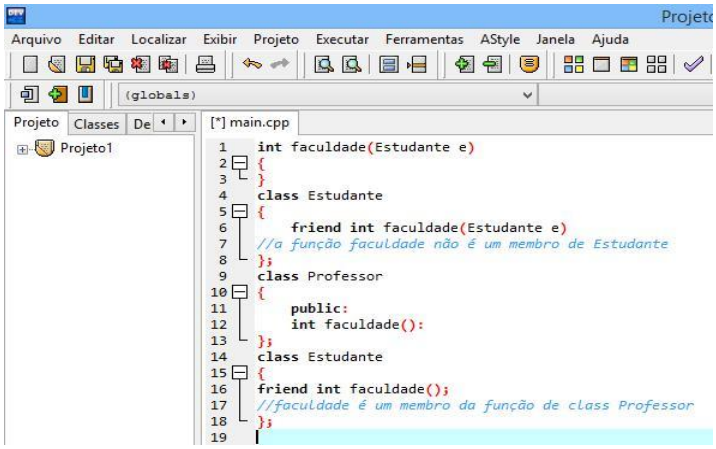

Figura 18: Função friend.

Quando o programa alcança a instrução de chamada de função, o programa segue o fluxo exibido na figura 19:

Figura 19: Fluxo do programa utilizando funções especiais.

- Função Inline

Outra função especial fornecida pelo C++ é a função inline, que é comumente utilizada para reduzir a sobrecarga das chamadas de funções. Uma chamada de função envolve uma sobrecarga em termos do tempo levado para se executar as etapas adicionais listadas acima.

Quando se declara uma função inline, o corpo da função é expandido no ponto em que é invocada. Ela não é compilada como um pedaço solto do código, mas haverá uma inserção quando houver uma chamada dessa função.

O código da figura 20, ilustra o exemplo da utilização da função inline. Neste programa, *adic*() é declarado como uma função inline que está sendo chamada em main():

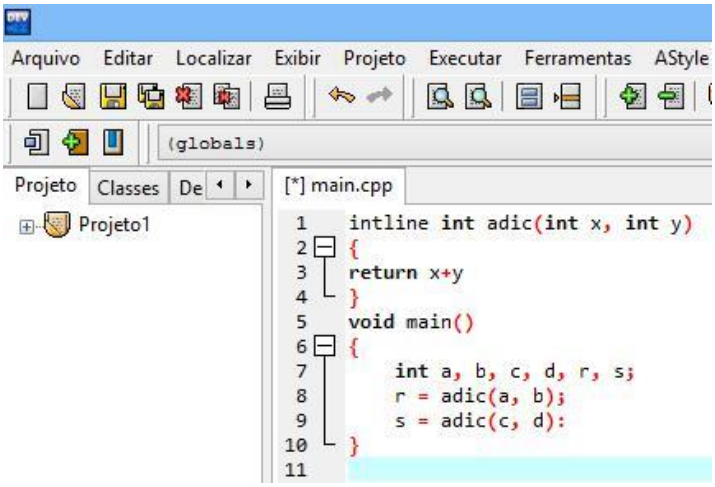

Figura 20: Função inline.

IV.4 SOBRECARGA

O C++ permite dois tipo de sobrecarga: de operador e de função.

IV.4.1 SOBRECARGA DE OPERADOR

A sobrecarga de operador é a atribuição de vários significados a um operador. Por exemplo, o operador + tem a capacidade de adicionar dois valores de qualquer tipo numérico padrão *(int, float* ou *double*).

O operador + não permite que sejam adicionados dois números complexos, pois *Complex* é um tipo de dado definido pelo programador. Para adicionar dois números complexos, pode-se sobrecarregar o operador +. Quando se é utilizado o mesmo operador para diferentes tipos de dados, então esse operador está sobrecarregado.

O código da figura 21, ilustra o operador +(*Complex* c1, *Complex* c2) definindo a nova versão do operador + para adicionar dois operandos da classe *Complex*. A declaração da classe *Complex*, evidenciada na figura, contém declarações para sobrecarga dos operadores + e =:

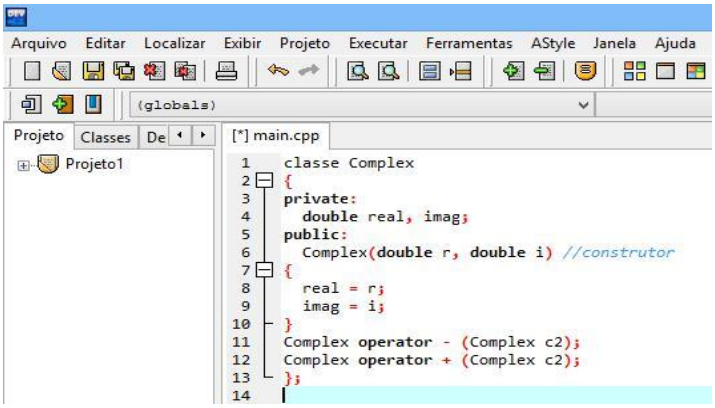

Figura 21: Sobrecarga de operadores.

O C++ impõe algumas restrições na sobrecarga do operador definido pelo usuário. Uma das restrições impostas pelo C++ é que o operadores não podem ser sobrecarregados para tipos de dados padrão. Pode-se sobrecarregar operadores apenas para tipos de dados definidos pelo programador, não podendo redefinir o operador + para retornar o produto de duas variáveis do tipo int, por exemplo.

IV.4.2 SOBRECARGA DE FUNÇÃO

Sobrecarga de funções é a criação de funções como o mesmo nome, mas com listas de argumentos distintos. Se existem funções que executam tarefas semelhantes, é mais eficaz atribuir o mesmo nome a tais. O tipo mais comum de sobrecarga de função é observador nos construtores de classe.

No C++ utiliza-se mais de um construtor de classe para lidar com diferentes tipo de inicialização, conforme pode ser observado no trecho de código da figura 22:

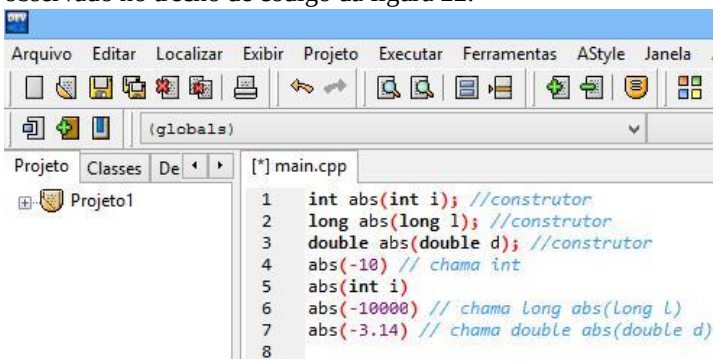

Figura 22: Sobrecarga de função.

Pode-se sobrecarregar funções nas duas listas de argumentos e não no tipo de retorno. No código a função *artigo*() não está sobrecarregada porque a lista de argumentos é do mesmo tipo em ambos os casos, mesmo que os tipos de retorno sejam diferentes. Quando se compila o fragmento de código mostrado abaixo, o compilador retornará um erro:

int artigo(int j);
void artigo(int j);

IV.5 BIBLIOTECA E FUNÇÕES DE ENTRADA E SAÍDA

IV.5.1 BIBLIOTECA DE E/S

A biblioteca de E/S do C++ oferece funções para ler e exibir caracteres. As funções de E/S que são comumente utilizadas para ler e exibir caracteres estão listadas:

- *put*() - exibe caracteres, aceitando um argumento do tipo de dados char;
- *get*() - extrai e retornar um único valor do fluxo de entrada, não extraindo caractere delimitador;
- *write*() - grava no arquivo indicado pelo descritor as informações obtidas do endereço fornecido;
- *read*() – realiza a leitura de dados do arquivo para a memória;
- *getline*() - extrai um bloco de caracteres do buffer até encontrar o limite numérico especificado;
- *peek*() - retorna o próximo caractere da entrada sem extraí-lo do fluxo de entrada;
- *putback*() - insere um caractere de volta no fluxo de entrada;
- *ignore*() - pula um número especificado de caracteres de entrada.

IV.5.2 FORMATAÇÃO DE INFORMAÇÕES

O C++ oferece as funções mostradas na figura para formatação de E/S. Essas funções de membros dos objetos de fluxo são: *cin e cout*. Para se utilizar essas funções de E/S é necessário incluir o arquivo de cabeçalho iostream.h no programa.

A função *widht*() é utilizada para definir o número máximo de caracteres armazenados em um buffer. Essa mesma função pode ser usada para fluxos de saída.

A função *fill*() é utilizada para preencher espaços extras com um caracter especificado. Por padrão, essa função utiliza espaços como caracteres de preenchimento. A função é invocada somente se o valor definido pela função *widht*() for maior que o tamanho do valor inserido.

Outra função de formatação do C++ é a *setw*(), que é um manipulador que define o comprimento de uma variável. Os manipuladores permitem que se alterem as características de um fluxo. A função *setw*() define o número máximo de caracteres que podem ser inseridos ou extraídos de cada vez. A função setw() não tem um valor padrão. Ela perde seu valor depois de ser utilizada uma vez. Torna-se necessário incluir o arquivo de cabeçalho *iomanip.h* para utilizá-lo.

O C++ dispõe ainda de outra função de formatação, denominada *precision*(), que é utilizada para definir número de dígitos a serem exibidos após o ponto decimal em vários tipos de dados *double* ou *float*. Por padrão, o número de dígitos exibidos pela função *precision*() é 6. A função permanece em vigor até ser redefinida para um novo valor de um programa.

IV.5.3 FUNÇÕES DE E/S

Para que se possa lidar com entrada e saída de arquivo, o C++ define três classes: *ifstream, ofstream, fstream*. As definições das três classes são armazenadas no arquivo de cabeçalho *fstream.h*:

- *ifstream* - derivada da classe istream, conecta um arquivo ao programa para entrada;
- *ofstream* - derivada da classe ostream, conecta um arquivo ao programa de saída;
- *fstream* - derivada da classe iostream, conecta um arquivo ao programa para entrada e saída.

As funções de entrada e saída de arquivo utilizadas no C++ são: *open*() e *closed*(). A função *open*() abre um arquivo num modo especificado. Pode-se então conectar o arquivo ao objetivo de classe usando a função *open*(). A sintaxe da função *open*() é: *open(nomedoarquivo, mode)*. Pode-se ainda abrir um arquivo para entrada e saída, conforme exibido no código da figura 23.

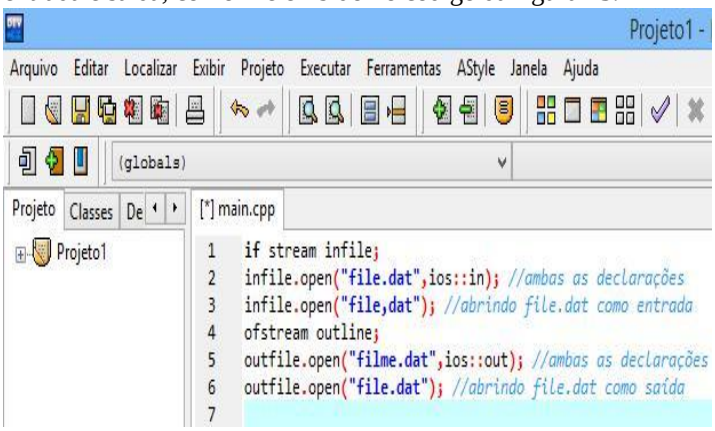

Figura 23: Funções de entrada e saída.

Outra função muito utilizada no C++ é a *close*(), utilizada para desconectar explicitamente um arquivo de um programa. Ao fechar uma conexão, não se elimina o fluxo, apenas desconecta-se o fluxo do arquivo de entrada ou de saída. O código abaixo exemplifica o fechamento de conexão:

1. *oufile.close*() //fecha a conexão para o arquivo de saída;
2. *infile.close*() //fecha a conexão para o arquivo de entrada.

Em C++ um arquivo também pode ser acessado randomicamente, utilizando-se funções próprias para tal finalidade. As funções utilizadas são:

- *seekg*() - move o ponteiro do arquivo para um endereço absouto dentro do arquivo;
- *seekp*() - move o ponteiro do arquivo para um determinado local no arquivo;
- *tellg*() - verifica a posição atual de um ponteiro de arquivo para fluxo de entrada;
- *tellp*() - verifica a posição atual de um ponteiro de arquivo para fluxo de saída.

O C++ dispõe ainda de algumas funções de verificação de fluxo para verificação de erros. O objeto da biblioteca *iostream* mantém um conjunto de sinalizadores de condição, através dos quais o progresso do fluxo pode ser monitorado por meio das seguintes funções:

- *eof*() - retorna um valor diferente de zero se o fluxo encontrar o fim do arquivo;
- *bad*() - retorna um valor diferente de zero se houver tentativa de realizar uma operação inválida, como buscar uma posição além do fim do arquivo;
- *fail*() - retorna um valor diferente de zero se uma operação for mal sucedida ou caso haja tentativa de realização de uma operação inválida;
- *good*() - retorna um valor verdadeiro se *eof*(), *bad*() e *fail*() retornarem falso.

Existem ainda, no C++, funções do buffer. Quando um fluxo é armazenado em buffer, as inserções ou extrações não têm operações de E/S correspondente para realizarem a gravação ou leitura de dados fisicamente em um dispositivo. Assim, todas as inserções e extrações são armazenadas em um buffer, de onde os dados são gravados ou lidos em porções. As funções de buffer estão listadas abaixo:

- *flush*() - permite que limpe-se o buffer de saída fazendo com que os dados contidos nele sejam gravados em um arquivo, garantindo que tudo que estiver armazenado seja exibido;

- *clear*() - redefine o estado de fluxo e desliga o sinalizador de fim de arquivo.

IV.6 HERANÇA

No C++, a classe derivada herda membros de dados e funções de membro de sua classe-base. A classe derivada pode ainda ter seus próprios membros e funções. A herança possibilita a reutilização de código. Quando uma classe é definida, pode ser

utilizada para criar classes derivadas. A reutilização de classes existentes poupa tempo e trabalho.

A sintaxe utilizada para derivar uma classe é: "*class Derived : Base"*. Pode-se criar uma classe derivada usando herança única ou múltipla.

IV.6.1 HERANÇA ÚNICA

- Pública

Na herança única pública, membros públicos e protegidos da classe-base tornam-se membros públicos e protegidos da classe derivada. O código da figura 24, ilusta um exemplo. A classe *Bug* tem dois elementos de dados privados: *legs* e *color*. Possui ainda duas funções de membros públicas: *Bug*() e *draw*():

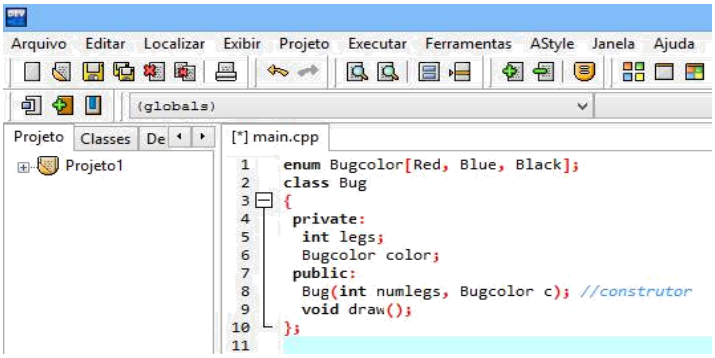

Figura 24: Herança única pública.

- Privada

Na herança única privada, os membros públicos e protegidos da classe-base, tornam-se membros privados da classe derivada. Uma classe derivada não herda membros privados da classe-base. Na definição da classe exibida no código da figura 25, o qualificador private indica que a classe *Humbug* é derivada da classe *Bug* através de herança privada:

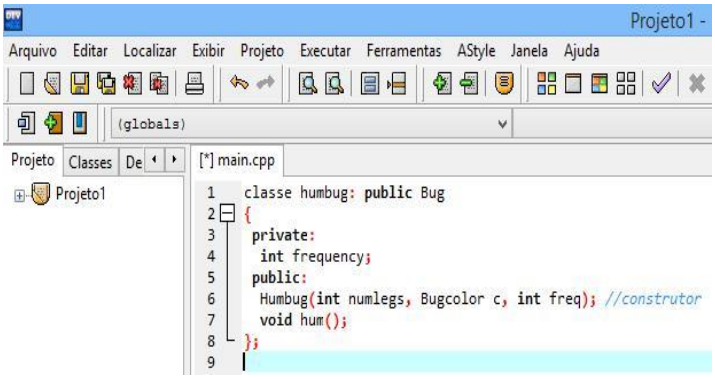

Figura 25: Herança única privada.

IV.6.2 HERANÇA MÚLTIPLA

Na herança múltipla, uma classe é derivada de mais de uma classe-base. Uma classe derivada também pode herdar funções de membro do mesmo nome de mais de uma classe-base.

Para especificar a que função *set*() está se referindo, usa-se o operador de resolução de escopo, conforme exibido no código da figura 26:

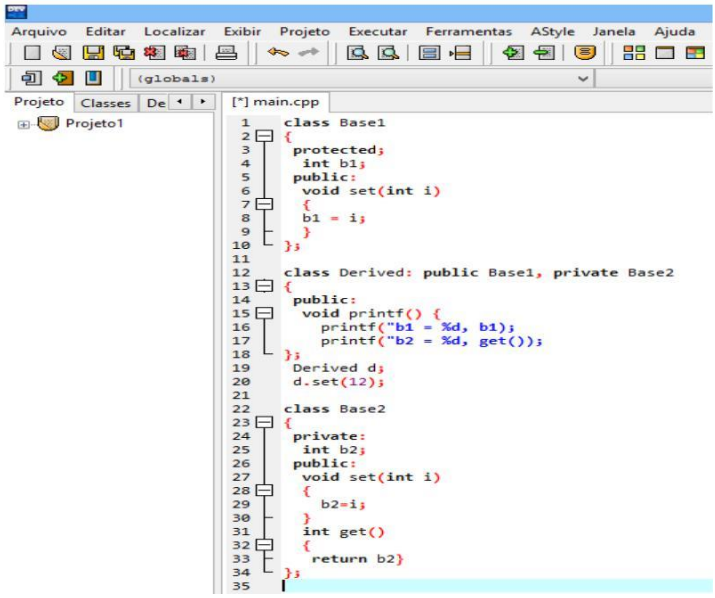

Figura 26: Herança múltipla.

Do mesmo modo, quando se cria uma classe derivada a partir de mais de uma classe-base, precisa-se também especificar que construtor está sendo invocado pela classe derivada. Para diferenciar os construtores das classes-base, deve-se especificar os argumentos no construtor da classe derivada.

IV.7 POLIMORFISMO

Polimorfismo, em C++, é a propriedade que permite que funções de mesmo nome tenha interpretações diferentes em uma classe-base e em suas classes derivadas. A mesma mensagem é enviada a dois objetos diferentes e eles produzem dois conjuntos diferentes de ações.

No trecho do código exibido na figura 27, as classes derivadas como *Square* e *Circle* herdaram a função *draw*() da classe-base Figure. Quando essa função é invocada, a classe *Square* desenha um quadrado e a classe *Circle* desenha um círculo.

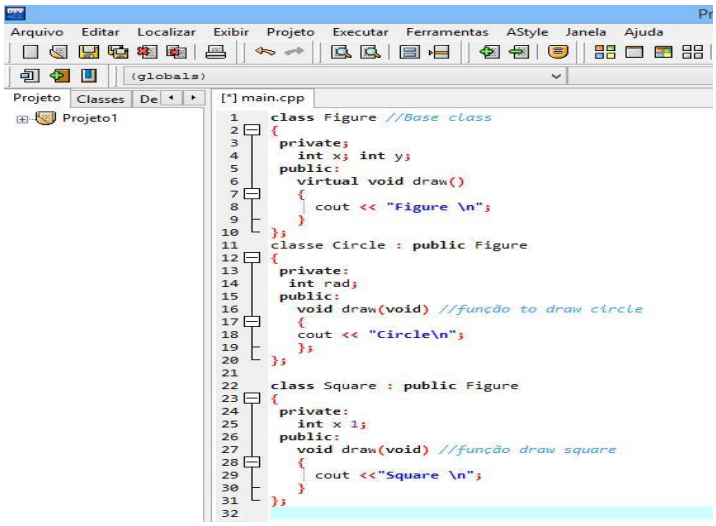

Figura 27: Polimorfismo.

O compilador do C++ associa a função a uma classe identificando o tipo do objetivo ou do ponteiro usado para invocá-lo. Esse processo de associar uma função a uma chamada, é conhecido como vinculação, e possui dois tipos:

- Estática: resolve a chamada de função em tempo de compilação;
- Dinâmica: resolve a chamada de função em tempo de execução.

Funções polimórficas usam a vinculação dinâmica para resolver chamadas de função porque o compilador não pode determinar que definição de função de membro chamar, uma vez que as funções de membro possuem o mesmo nome.

Outro importante recurso do C++, dentro do polimorfismo, são as funções virtuais. Para se declarar uma função virtual, deve-se adicionar a palavra virtual antes do nome da função. Isso indica, para o compilador do C++, que a função opera como um espaço reservado no programa.

Quando um programa é executado, a função virtual é substituída pela função real, conforme exibido no trecho de código da figura 28:

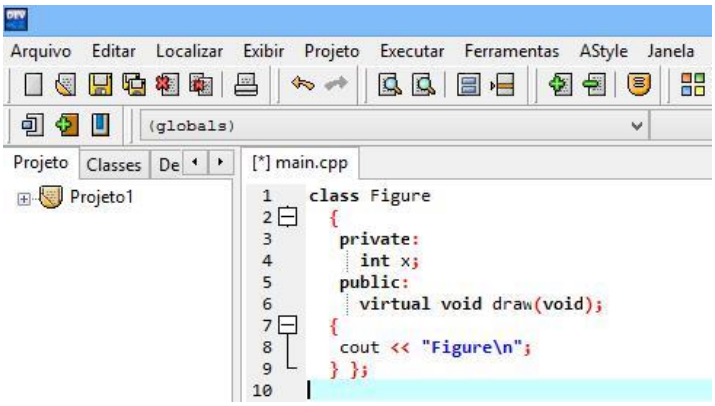

Figura 28: Funções Virtuais.

É necessário declarar uma função como virtual na classe-base. É opcional a declaração de uma função virtual da classe-base como virtual nas classes derivadas. Assim, o C++ cria um membro de dados oculto para cada função virtual definida em uma classe. A palavra-chave virtual informa ao compilador que o ponteiro ou referência a um objeto deve invocar a função de membro da classe derivada em vez da função de membro da classe-base. Como resultado, obtém-se a saída desejada independentemente de uma função estar sendo chamada como instância da classe-base ou derivada.

V. CONCLUSÃO

A linguagem de programação C/C++ tornou-se uma das mais utilizadas e estudadas por acadêmicos e pesquisadores em todo o mundo, devido a sua portabilidade, modularidade, recursos de baixo nível, imperatividade e simplicidade, embora atualmente existam diversas outras linguagens. Internalizar o conhecimento sobre esta tecnologia traz grandes vantagens ao programador que deseja aplicar a cognição, sobretudo, para o desenvolvimento de softwares voltados para microprocessadores e microcontroladores, tais como o Arduino ou memórias voláteis, além de dispositivos que contam com a Internet das Coisas (IoT) e a domótica. O desenvolvimento e a evolução da linguagem C++ é fruto do trabalho de milhares de acadêmicos, profissionais, pesquisadores e cientistas da indústria de software, que utilizam a linguagem em seu ensino, na construção de bibliotecas de rotinas ou participando de comitês internacionais de padronização, entre outras atividades. O C/C++ é uma excelente linguagem também para o desenvolvimento de novos sistema operacionais, pois além de eficiente, dispõe de inúmeros recursos para controlar a memória da máquina, acessando o microcontrolador e permitindo chamadas de rotinas em Assembly. Além disso, sua portabilidade faz com que o código possa ser compilado em diversas arquiteturas de hardware ou software, podendo ser utilizada nos mais variados sistemas operacionais, dentre os quais podemos destacar o MacOS, Linux e Windows. A modularidade, outra característica revelante e peculiar desta tecnologia, permite que o código seja dividido em vários blocos de programação distintos, portanto, no momento em que a função é fechada, o que foi escrito não interfere nos blocos seguintes.

A sintaxe do C++ é simples, embora cheia de recursos. Desde que as regras sejam seguidas, o programador dificilmente cometerá erros, pois o compilador enviará avisos e alertas sobre eventuais problemas de sintaxe, fornecendo dicas para correção imediata do erro. Por fim, este artigo permitiu elucidar, de modo analítico e substanciado, as raízes da linguagem de programação C/C++, assim como aplicações e fatores que a tornaram uma das linguagens mais estudadas, utilizadas e exploradas em todo o mundo. Os códigos exibidos através das figuras, em todo o artigo, exemplificaram o uso de recursos e evoluções do C++ em relação ao C, identificando e elucidando fatores que influenciaram a trajetória de sucesso do C++. Obviamente, embora não seja a melhor linguagem para a solução de todos os problemas, o seu conjunto de atributos a capacita para resolver problemas em diversos níveis de complexidade, podendo-se fazer qualquer coisa com esta poderosa linguagem de programação.

VI. AGRADECIMENTOS

Ao Instituto de Tecnologia e educação Galileo da Amazônia (ITEGAM) e ao Centro Universitário do Norte (UNINORTE), pelo apoio à pesquisa.

VII. REFERÊNCIAS

[1] Aguilar, J. L. **C + +**. 1. ed. São Paulo:McGraw Hill, 2008.

[2] Aho, A. V.; Jeffrey; Sethi, R.; Lam, M. S. (2008). **Compiladores. Princípios, técnicas e ferramentas.** São Paulo: Addison-Wesley, Pearson. p. 3-5.

[3] Barth, A. C. **Tabela herança**. Ibirama, Universidade do Estado de Santa Catarina – UDESC, Centro de Desenvolvimento do Alto Vale do Itajaí – CEAVI.

[4] Brain, M. **Entendendo C++**. Interface Technologies, 1998.

[5] Feliciano. P.; Lamego, C. G. **Linguagem de Programação C++**. Disponível em:<http://www.ceavi.udesc.br/arquivos/id_submenu/387/patricia_feliciano___guilherme_cesar_lamego.pdf>. Acesso em 21/11/2018.

[7] Liberty, J. **C + + de A a Z**. ed. Campus, 1999.

[8] Marconi, A. M.; Lakatos, M. E. **Metodologia do trabalho científico: procedimentos básicos, pesquisa bibliográfica,**

projeto e relatório, publicações e trabalhos científicos. 7. ed. – 6. reimpr. São Paulo: Atlas: 2011.

[9] Melo, A. C. V.; Silva, F. S. C. (2003). **Princípios de Linguagens de Programação**. São Paulo: Edgard Blücher Ltda. p. 7-11. 211.

[10] Trevelin, E. C. **Apostila de C++**. 2007. Faculdade de Engenharia de Ilha Solteira. Disponível em: <http://www.dee.feis.unesp.br/graduacao/disciplinas/langcpp/index.php?pagina=modulo01>. Acesso em 20/11/2018.