Alisson dos Santos Nascimento - 509538
Johnatas Félix Zuza - 422160

ANÁLISE HISTÓRICA: C++

Desenvolvido por Bjarne Stroustrup em 1980 nos Laboratórios Bell de New Jersey, o C++ é uma linguagem de programação de nível médio baseada na linguagem C. C++ é uma extensão da linguagem de programação C, por conta disso inicialmente a linguagem era chamada de "C com classes". A motivação para o desenvolvimento do C++ foi a complexidade de grandes sistemas implementados com a linguagem C, eles eram de difícil entendimento e controle, portanto o C++ surgiu para permitir que esta barreira fosse quebrada, possibilitando que programadores pudessem gerenciar e compreender programas maiores e mais complexos.

O objetivo do desenvolvimento desta linguagem era melhorar uma versão do núcleo Unix(sistema operativo portável, multitarefa e multiutilizador)que era escrito em C. Alguns dos desafios incluíam simular a infraestrutura da comunicação entre processos num sistema distribuído ou de memória compartilhada e escrever drivers para tal sistema. Stroustrup observou algumas linguagens que serviram de base para a implementação da linguagem C , o que viria a gerar o C++, como o Simula 67 que possuía características bastante úteis para o desenvolvimento de software e a linguagem BCPL que era rápida, mas possuía demasiado baixo nível, dificultando sua utilização no desenvolvimento de aplicações. Logo, Stroustrup começou a acrescentar elementos do Simula 67 no C e para desenvolver a linguagem foram também acrescentados elementos de outras linguagens (ALGOL 68, Ada, CLU e ML) de vários níveis, na tentativa de criar uma linguagem com elementos novos e que mante-se a compatibilidade. No início do desenvolvimento, a linguagem usava um pré-processador, mas Stroustrup criou um compilador próprio, com novas características.

Ainda em 1983 o nome da linguagem foi alterado de C com Classes para C++ já possuindo seu compilador próprio, tendo também novas características adicionadas, como funções virtuais, sobrecarga de operadores e funções, referências, constantes, gerenciamento manual de memória, melhorias na verificação de tipo de dado e estilo de comentário de código de uma linha . A primeira versão oficial do C++ apareceu em 1985, e em 1989 foi lançada uma segunda versão da linguagem, com acréscimo das características: Herança múltipla, classes abstratas, métodos estáticos, métodos constantes e membros protegidos, incrementando também um suporte de orientação a objeto.Por fim no ano de 1990 foi adicionado na linguagem gabaritos, tratamento de exceções, espaço de nomes, conversão segura de tipo de dado e o tipo booleano.

Embora C++ fosse inicialmente projeto para ajudar na gerência de programas muito grandes, seu uso não se limita apenas a estes casos. Na verdade, os atributos orientados a objetos de C++ podem ser efetivamente aplicados a praticamente qualquer tarefa de programação. Desde os anos 1990 é uma das linguagens comerciais mais populares ,C++ já vem sendo utilizado em projetos tais como editores, bancos de dados, sistemas pessoais de arquivos e programas de comunicação. Com a mesma eficiência de C, C++ pode ser usado para construir

software de sistemas com ótimo desempenho, sendo a linguagem adotada pela maior parte de empresas e centros desenvolvendo software em grande escala. Alguns dos mais conhecidos programas são feitos em C++, como: Adobe Photoshop, MySQL, Mozilla Firefox, Internet Explorer, Microsoft Windows, entre vários outros

ANALISADOR LÉXICO/SINTÁTICO E ÁRVORE PARSER C++:

```
/* front.c - um analisador léxico e analisador sintático
 simples para expressões aritméticas simples */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

/* Declarações globais */
/* Variáveis */
int charClass;
char lexeme [100];
char nextChar;
int lexLen;
int token;
int nextToken;
FILE *in_fp, *fopen();

/* Declarações de Funções */
void addChar();
void getChar();
void getString();
void getNonBlank();
int lex();
void getString();
void error();
void expr(int level);
void term(int level);
void factor(int level);
void operator(int level);

/* Classes de caracteres */
#define LETTER 0
#define DIGIT 1
```

```
#define QUOTE 2
#define STD_RESV 3
#define COUT_RESV 4
#define LIST_CHAR 5
#define UNKNOWN 99
/* Códigos de tokens */
#define INT_LIT 10
#define IDENT 11
#define ASSIGN_OP 20
#define ADD_OP 21
#define SUB_OP 22
#define MULT_OP 23
#define DIV_OP 24
#define LEFT_PAREN 25
#define RIGHT_PAREN 26
#define STRING_LIT 28
#define OUTPUT_OP 29
#define SEMI_COLON 30

/*****************************************************/
/* função principal */
int main() {

	char resposta[10];
	int i = 1;
	int *pi;
	pi = &i;
	int level = 0;

	while (i) {
		printf("Escolha uma opcao:\n");
		printf("1. Abrir arquivo de Exemplo.\n");
		printf("2. Escrever expressao.\n");
		printf("0. Sair\n");
		printf("Opcao: ");

		fgets(resposta, sizeof(resposta), stdin);

		// Remover o caractere de nova linha da resposta
		resposta[strcspn(resposta, "\n")] = '\0';

		// Verificar a opção escolhida
		if (strcmp(resposta, "1") == 0) {
			printf("Opcao selecionada: Abrir arquivo de Exemplo.\n");
```

```
                if ((in_fp = fopen("exemplo.txt", "r")) == NULL)
                        printf("ERROR - cannot open front.in \n");
                else {
                        getChar();
                        do {
                                lex();
                                expr(level);
                        } while (nextToken != EOF);
                }
        } else if (strcmp(resposta, "2") == 0) {
                printf("Opcao selecionada: Escrever expressao.\n");
                char input[100];
                printf("Digite uma expressao: ");
                scanf("%s", input);
                getchar();
                in_fp = fopen("temp.txt", "w+");
                fprintf(in_fp, "%s", input);
                fclose(in_fp);

                /* Abrir o arquivo e dados de entrada e processar seu
                conteudo */
                if ((in_fp = fopen("temp.txt", "r")) == NULL)
                        printf("ERROR - cannot open temp.txt \n");
                else {
                        getChar();
                        do {
                                lex();
                                expr(level);
                        } while (nextToken != EOF);
                }
        } else if (strcmp(resposta, "0") == 0) {
                printf("Saindo...\n");
                *pi = 0;
        } else {
                printf("Opcao invalida! Tente novamente.\n");
        }
    }

    return 0;
}
/*****************************************************/
/* lookup - uma função para processar operadores e parênteses
 e retornar o token */
int lookup(char ch) {
```

```
        switch (ch) {
                case '(':
                        addChar();
                        nextToken = LEFT_PAREN;
                        break;
                case ')':

                        addChar();
                        nextToken = RIGHT_PAREN;
                        break;
                case '+':
                        addChar();
                        nextToken = ADD_OP;
                        break;
                case '-':
                        addChar();
                        nextToken = SUB_OP;
                        break;
                case '*':
                        addChar();
                        nextToken = MULT_OP;
                        break;
                case '/':
                        addChar();
                        nextToken = DIV_OP;
                        break;
                case ';':
                        addChar();
                        nextToken = SEMI_COLON;
                        break;
                default:
                        addChar();
                        nextToken = EOF;
                        break;
        }

        return nextToken;
}
/*****************************************************/
/* addChar - uma função para adicionar nextChar ao
vetor lexeme */
void addChar() {
        if (lexLen <= 98) {
                lexeme[lexLen++] = nextChar;
```

```
                lexeme[lexLen] = 0;
        } else
                printf("Error - lexeme is too long \n");
}
/*****************************************************/
/* getChar - uma função para obter o próximo caractere da entrada e determinar sua
classe de caracteres */
void getChar() {
        if ((nextChar = getc(in_fp)) != EOF) {
                if (isalpha(nextChar))
                        charClass = LETTER;

                else if (isdigit(nextChar))
                        charClass = DIGIT;
                else if (nextChar == "")
                        charClass = QUOTE;
                else if (nextChar == ':') {
                        charClass = STD_RESV;
                } else if (nextChar == '<') {
                        charClass = COUT_RESV;
                } else
                        charClass = UNKNOWN;
        } else
                charClass = EOF;
}

/*****************************************************************/
/* getString - uma função para obter o próximo char da entrada
para aceitar qualquer caractere string, com exceção de " e quebra de linha */
void getString () {
        if ((nextChar = getc(in_fp)) != EOF) {
                if (nextChar == "") {
                        charClass = QUOTE;
                } else if (nextChar == '\n') {
                        error();
                } else
                        charClass = LIST_CHAR;
        } else {
                charClass = EOF;
        }
}

/*****************************************************/
/* getNonBlank - uma função para chamar getChar até que ela
```

```
retorne um caractere diferente de espaço em
branco */
void getNonBlank() {
        while (isspace(nextChar))
                getChar();
}
/*****************************************************/
/* lex - um analisador léxico simples para expressões
aritméticas */
int lex() {
        lexLen = 0;
        getNonBlank();
        switch (charClass) {
                        /* Reconhecer identificadores */
                case LETTER:
                        addChar();
                        getChar();
                        while (charClass == LETTER || charClass == DIGIT) {
                                addChar();
                                getChar();
                        }
                        if (strcmp(lexeme, "std::cout") == 0) { // Verificar se é "std::cout"
                                nextToken = COUT_RESV; // Atribuir token
correspondente
                        } else {
                                nextToken = IDENT;
                        }
                        break;
                        /* Reconhecer literais inteiros */
                case DIGIT:
                        addChar();
                        getChar();
                        while (charClass == DIGIT) {
                                addChar();
                                getChar();
                        }
                        nextToken = INT_LIT;
                        break;
                        /* Caracteres de expressões*/
                case QUOTE:
                        addChar();
                        getString();
                        while(charClass == LIST_CHAR) {
                                addChar();
```

```
                        getString();
                }
                if(charClass == QUOTE) {
                        addChar();
                        getChar();
                        nextToken = STRING_LIT;
                } else {
                        error();
                }
                break;
        case UNKNOWN:
                lookup(nextChar);
                getChar();
                break;
                /* Fim do arquivo */
        case EOF:
                nextToken = EOF;
                lexeme[0] = 'E';
                lexeme[1] = 'O';
                lexeme[2] = 'F';
                lexeme[3] = 0;
                break;
    } /* Fim do switch */
    return nextToken;
} /* Fim da função lex */

//Imprime o nome do Token ao inves de seu codigo
void imprimirNomeToken(int token) {
    switch (token) {
        case INT_LIT:
                printf("INT_LIT");
                break;
        case IDENT:
                printf("IDENT");
                break;
        case ASSIGN_OP:
                printf("ASSIGN_OP");
                break;
        case ADD_OP:
                printf("ADD_OP");
                break;
        case SUB_OP:
                printf("SUB_OP");
                break;
```

```
                case MULT_OP:
                        printf("MULT_OP");
                        break;
                case DIV_OP:
                        printf("DIV_OP");
                        break;
                case LEFT_PAREN:
                        printf("LEFT_PAREN");
                        break;
                case RIGHT_PAREN:
                        printf("RIGHT_PAREN");
                        break;
/* tokens das palavras reservadas de C++*/
                case STD_RESV:
                        printf("STD_RESV");
                        break;
                case COUT_RESV:
                        printf("COUT_RESV");
                        break;
                case STRING_LIT:
                        printf("STRING_LIT");
                        break;
                case OUTPUT_OP:
                        printf("OUTPUT_OP");
                        break;
                default:
                        printf("Token desconhecido");
                        break;
        }
}
/*mensagem de Erro*/
void error() {
        printf("Error - ocorreu um erro na analise sintatica\n");
        exit(1);
}

/* expr
 Analisa sintaticamente cadeias na linguagem gerada pela
regra:
 <expr> -> <term> {(+ | -) <term>}
 */
// Funcao <statement> -> <command> | <string> | <expr>
// Funcao <statement> -> <command> | <string> | <expr>
void expr(int level) {
```

```
        printf("%*sEnter <expr>\n", level, "");
        level = level + 5;
        term(level);

        while (nextToken == ADD_OP || nextToken == SUB_OP) {
                operator(level);
                lex();
                term(level);
        }


        printf("%*sExit <expr>\n", level - 5, "");
}

/* term
 Analisa sintaticamente cadeias na linguagem gerada pela
regra:
 <term> -> <factor> {(* | /) <factor>)
 */
void term(int level) {
        printf("%*sEnter <term>\n", level, "");
        level = level + 5;

        factor(level);

        while (nextToken == MULT_OP || nextToken == DIV_OP) {
                operator(level);
                lex();
                factor(level);
        }

        printf("%*sExit <term>\n", level - 5, "");

}

/* factor
 Analisa sintaticamente cadeias na linguagem gerada pela
regra:
 <factor> -> id | int_constant | (<expr)
 */
/* factor
 Analisa sintaticamente cadeias na linguagem gerada pela
regra:
 <factor> -> id | int_constant | (<expr)
```

```
 */
void factor(int level) {
	printf("%*sEnter <factor>\n", level, "");

	/* Determina qual RHS */
	if (nextToken == IDENT || nextToken == INT_LIT) {
		printf("%*s", level, "");
		imprimirNomeToken(nextToken);
		printf(" --> %s\n", lexeme);
		/* Obtém o proximo token */
		lex();
	}

	/* Se a RHS é (<expr>), chame lex para passar o parêntese
	esquerdo, chame expr e verifique pelo parêntese
	direito */
	else {
		if (nextToken == LEFT_PAREN) {
			printf("%*s", level, "");
			imprimirNomeToken(nextToken);
			printf(" --> %s\n", lexeme);
			lex();
			expr(level);
			if (nextToken == RIGHT_PAREN) {
				printf("%*s", level, "");
				imprimirNomeToken(nextToken);
				printf(" --> %s\n", lexeme);
				lex();
			} else
				error();
		}
		/* Não era um identificador, um literal inteiro ou um
		parêntese esquerdo */
		else
			error();
	}

	printf("%*sExit <factor>\n", level, "");
}
/*função para imprimir os operadores*/
void operator (int level) {
	printf("%*s", level, "");
	imprimirNomeToken(nextToken);
	printf(" --> %s\n", lexeme);
```

}

DEBULHANDO A LINGUAGEM: C++

A linguagem C++, não possui um limite estrito para o número de caracteres em um programa. Os nomes ou identificadores em C++ podem ser compostos por letras maiúsculas e minúsculas, dígitos numéricos e o caractere de sublinhado (_), entretanto, o primeiro caractere não pode ser um dígito numérico. Além disso, a sensibilidade à capitalização é uma característica da linguagem, isso significa que letras maiúsculas e minúsculas são distintas e fazem diferença nos nomes.

Ex.:

SOL Sol sol

Independente de possuírem o mesmo nome, são identificadores diferentes.

Palavras-chave e palavras reservadas são essencialmente a mesma coisa em C++, ambos os termos são usados para descrever palavras que têm significados especiais e são reservadas para uso específico na linguagem. C++ apresenta um grande acervo dessas palavras, abaixo estão algumas das mais conhecidas:

- ***Auto ,bool ,char ,case ,const ,short ,typename ,typedef ,struct ,nullptr ,enum.***

Em C++, é possível definir apelidos para tipos de dados existentes usando o recurso de "*typedef*" ou "*using*". Em C++, o tipo de uma variável ou expressão é especificado durante a declaração da mesma, existem três tipos de vinculação em C++: vinculação estática, vinculação dinâmica e vinculação automática. A vinculação estática é o método principal sendo esse tratado como o mais relevante método de vinculação em C++. Em C++, as declarações de variáveis podem ser feitas tanto de forma explícita quanto de forma implícita, depende da forma como se deseja definir o tipo da variável, por conta disso, o tipo de uma variável não pode ser necessariamente determinado pelo formato de seu nome.

Os tipos são vinculados estaticamente nessa linguagem, portanto não é possível escrever um programa em C++ para processar dados sem conhecer os tipos desses dados. A linguagem C++ é uma linguagem de programação multiparadígma, contudo ela foi criada para dar suporte principalmente à programação imperativa.O C++ permite aos programadores incluírem o especificador "*static*" a uma definição de variável, possibilitando as variáveis definidas serem estáticas, com isso, em vez de ser de instância a variável é de

classe. C++ permite declarações de variáveis em qualquer lugar onde uma sentença poderia ocorrer tendo todas as variáveis declaradas em uma função ou método, sendo as variáveis definidas em métodos, por padrão dinâmicas da pilha. A linguagem possibilita a alocação de memória dinâmica por meio dos comandos "*new"* e "*delete", possibilitando o* aloque e liberação de memória explicitamente durante a execução do programa.

Os escopos presentes na linguagem C++ são: bloco, função, classe e namespace. Uma das características do C++ é que ele permite que as declarações de variáveis apareçam em qualquer lugar onde uma sentença poderia aparecer na unidade de programa. C++ têm tanto declarações quanto definições de dados globais, permitindo uma estrutura de programa onde as definições de variáveis podem aparecer fora das funções, nessa linguagem uma variável global oculta pode ser acessada por meio do operador de escopo (::). C++ possui três principais durações de tempo de vida: duração automática, duração estática e duração dinâmica, e mesmo sendo diferentes, escopo e tempo de vida, elas estão diretamente relacionadas. Por exemplo, uma variável declarada em "*static*" é estaticamente vinculada ao escopo dessa função, mas seu tempo de vida se estende pela execução completa do programa do qual ela pertence. C++ permite a criação de constantes nomeadas por meio da palavra reservada "*const*".

Ex.:

***Const int** MAX* = 10*;*

Assim "MAX" é declarado como um inteiro e possui o valor fixo de 10.

C++ possui os tamanhos de inteiro com sinal:

"int", "long int", "unsigned int"

Contudo a biblioteca padrão da linguagem também fornece tipos inteiros com tamanhos fixos e garantidos:

"int8_t", "int16_t", "int32_t", "int64_t"

C++ ainda inclui tipos inteiros sem sinal, que são geralmente usados para dados binários. Essa linguagem também possui três tipos de ponto flutuante:

*"**float", "double", "long double"***

A escolha do tipo de ponto flutuante depende da precisão desejada para o seu programa.

C++ suporta também tipos de dados complexos, por meio da biblioteca padrão “*complex”* é possível trabalhar com números complexos. C++ não apresenta apesar de tudo um tipo de dado decimal na sua biblioteca padrão, sendo essa linguagem baseada principalmente em pontos flutuantes e números inteiros. Além de possuir um tipo booleano, a linguagem C++ também é uma das poucas que permite expressões numéricas sendo usadas como se fossem booleanas. A base do conjunto de caracteres da linguagem é a “*ASCII*” contudo ela também possui suporte para o “*UNICODE”,* com isso a linguagem possui recursos para casamento de padrões. A linguagem C++ usa vetores de caracteres “*char*” para armazenar cadeias de caracteres, diferente de outras linguagens que possuem comandos como “String”. C++ utiliza de cadeias de tamanho dinâmico na sua biblioteca padrão, requerendo a alocação e a liberação de armazenamento dinâmico, além de suportar as cadeias de tamanho estático e dinâmico limitado também.

C++, não possui tipos de dados específicos para representar valores ordinais primitivos, contudo pode-se usar dados numéricos para representar valores ordinais, como “*int*”, “*char*”,”*enum*”. C++ possui os tipos de enumeração da linguagem C, contudo C++ também permite que as constantes de enumeração sejam atribuídas a variáveis de qualquer tipo numérico, podendo as constantes de enumeração aparecer em apenas um tipo enumeração que seja no mesmo ambiente de referenciamento. C++ utiliza três tipos de enumeração: enumeração simples, enumeração com escopo e enumeração com tipo subjacente.

Referências:

https://pt.wikipedia.org/wiki/C%2B%2B
https://www.dca.fee.unicamp.br/cursos/POOCPP/node18.html#:~:text=C%2B%2B%20Sum%C3%A1rio-,Origens%20de%20C%2B%2B,para%20C%20%2B%2B%20em%201983.
https://www.infoescola.com/informatica/cpp/
Documentação do C++ – introdução, tutoriais, referência. | Microsoft Learn

Escopo (C++) | Microsoft Learn

Programação funcional versus programação imperativa − LINQ to XML | Microsoft Learn

Livro - Conceitos de Linguagens de Programação Robert W.Sebesta