INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO C++

Objetivos

- Apresentar a descrição da linguagem C++;
- Apresentar as estruturas básicas de controle em C++;
- Apresentar a forma de codificação em linguagem C++;
- Apresentar padrões de mapeamento para a linguagem C++.

Histórico

- 1972 – primeira versão de C , por Dennis Ritchie;
- 1979 – publicação do livro "The C Programming Language";
- 1980 – início dos trabalhos de Bjarne Stroustrup na AT&T;
- 1985 – divulgação do C++ fora da AT&T;
- 2000 – padronização ANSI.

Descrição da linguagem

- Alfabeto

  Um programa em C++ poderá conter os seguintes caracteres:

  - as vinte e seis (26) letras do alfabeto inglês:
    A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
    a b c d e f g h i j k l m n o p q r s t u v w x y z

  - os dez (10) algarismos:
    0 1 2 3 4 5 6 7 8 9

  - os símbolos:

<	menor	( )	parênteses
>	maior	[ ]	colchete
.	ponto	{ }	chaves
,	vírgula	+	soma
:	dois pontos	-	subtração
;	ponto-e-vírgula	*	asterisco
=	igualdade	/	barra
!	exclamação	#	sustenido
?	interrogação	“	aspas
&	ampersete ("e" comercial)	‘	apóstrofo
^	circunflexo	%	porcento
|	barra em pé	~	til

- Pontuação
  - Ponto-e-vírgula é usado para separar comandos, a menos que outro separador seja necessário;
  - Em alguns casos de operadores, convém o uso de espaços em branco antes, e depois.

- Observação:
  Em C++ utilizam-se, ***obrigatoriamente***, as letras minúsculas para os comandos próprios da linguagem.

Tipos de dados

- Tipos básicos

Algoritmo	C++
inteiro	**int**
real	**float**
caractere	**char**

- Outros tipos:

short	inteiros "*curtos*"
long	inteiros "*longos*": [-2.147.483.648,2.147.483.647]
unsigned	inteiros sem sinal: [0, 65535]
double	reais com dupla precisão

- Especificação de classe de armazenamento:

const	constante
auto	alocação na pilha até retorno de função
static	alocação em memória durante a execução
register	alocação em registrador (valores escalares)
extern	não aloca memória (declaração externa)

Exemplos:

```
int          y;
static int   x;
unsigned     z;
float        i, j, k;
double       a, b, c;
char         letra;
```

- Apontadores

Se o nome de uma variável é precedido por um asterisco (*), significa que o objeto será do tipo apontador.

Podem ser operados aritmeticamente, como inteiros, e comparados com outros apontadores, ou a uma constante deste tipo, ***NULL*** (que não aponta para nenhum objeto válido).

Exemplos:

```
int          *y;
float*       i, j, k;
char         *letra;
```

O endereço de um objeto pode ser manipulado diretamente precedendo o nome do objeto por um sinal & (ampersete).

Exemplos:

```
i      = &j;
nome = &endereco;
```

- Definição de novos tipos

  Forma geral:

  typedef <nome> <tipo>;

  Exemplo:

  ```
  typedef     REAL        float;
  typedef     INTEIRO     int   ;

  REAL        a, b, c;
  INTEIRO     *i, j;
  ```

- Constantes

  - Constante inteira

    Exemplos:

    ```
    10, 532, -10567
    067, 05421, 0724            (octal)
    0L, 130000000L, 3241L       (inteiro longo)
    0x41, 0xFFFF, 0xA0          (hexadecimal)
    ```

    Observação:
    Em geral, tem-se a faixa de -32768 a +32767, para 2 bytes de representação.

  - Constante real

    Exemplos:

    10.465 -5.61 +265. 0.0 .731 .37e+2 -3.e-1

    Observações:
    A vírgula decimal é representada por ponto decimal.
    Em geral, tem-se a faixa de 10^{-38} a 10^{+38}.

  - Constante literal

    Exemplos:

    ```
    Caractere   : '1', '  ', '*', 'A', 'a', '?'
    Cadeia      : "BRANCO", "CEU AZUL"
    ```

    Observações:
    O tamanho da cadeia é limitado.
    As cadeias são terminadas pelo símbolo especial ‘\0’.

- Caracteres predefinidos:

  ```
  '\0'      nulo (fim de cadeia de caracteres)
  '\n'      passa para a próxima linha
  '\t'      passa para a próxima coluna de tabulação (9,17, ... )
  '\b'      retorna o cursor uma coluna
  '\r'      posiciona o cursor no inicio da linha
  '\f'      limpa a tela ou passa para a próxima página
  '\\'      barra invertida
  '\''      apóstrofo
  '\nnn'    representação de um byte, em octal
  '\xnn'    representação de um byte, em hexadecimal
  ```

- Definição de constantes

  Formas gerais:

  #**define** <NOME1> <valor1>
  #**define** <NOME2> (<valor2>)

  const <tipo> <NOME> = <valor>;

  Exemplos:
  #**define** TRUE 1
  #**define** FALSE 0

  const float PI = 3.1415926;

- Variáveis

  - Nome de variável

    a) O nome de uma variável tem tamanho determinado;
    b) O primeiro caractere é uma letra ou travessão (_);
    c) Outros caracteres podem ser letra, algarismo ou travessão (_).

    Exemplos:
    Nomes válidos : I, a, de, V9a, Lista_Notas

    Nomes inválidos: x+, t.6, 43x, so 5

  - Declaração de variáveis

    - Variáveis simples

      Forma geral:

      <tipo 1> <lista de nomes 1>;
      <tipo 2> <lista de nomes 2>;
      ...
      <tipo N> <lista de nomes N>;

      Exemplos:
      char fruta;
      int i, j, k;
      float p, DELTA;

      A declaração de variáveis pode ser usada também para a atribuição de valores iniciais.

      Exemplos:
      int x = 10,
      y = 20;

- Variáveis agrupadas
  - Homogêneas

    Forma geral:

    ```
    <tipo 1> <lista de nomes 1> [índice];
    <tipo 2> <lista de nomes 2> [índice 1] [índice 2];
      ...
    <tipo N> <lista de nomes N> [índice] ... ;
    ```

    Exemplos:

    ```
    char  frutas   [10],
    char  letras   [3] = {'a','b','c'},
    char  nomes [3][10] =
          {
            "Alfredo",
            "Jose",
            "Mario"
          };
    int   v[5] = {1,2,3,4,5}, j[4][4], k;
    ```

    Observação:

    O primeiro elemento tem índice igual a zero.

  - Heterogêneas

    Forma geral:

    enum {<lista de valores>} <declaração de nomes>;

    struct <nome> {<campos>} <lista de nomes>;

    union <nome> {<lista>} <lista de nomes>;

    Exemplos:

    ```
    enum {banana,laranja,abacaxi} fruta = banana;

    struct pessoa
    {
      char nome, endereco;
      int   rg, cpf, titulo_eleitoral;
    };

    struct pessoa funcionario, operário;
    ```

    Observação:

    O acesso aos campos de uma estrutura pode ser feito por

    nome.membro ou apontador -> membro

- Tipos de operadores
  - Aritméticos

Algoritmo	C++
* / ***mod***	* / %
+ -	+ -

Observações:

O operador ***div*** (divisão inteira) é a própria barra (/), quando os operandos forem inteiros.

Existem formas compactas para incremento e decremento:

<variável inteira>++	pós-incremento
++<variável inteira>	pré-incremento
<variável inteira>--	pós-decremento
--<variável inteira>	pré-decremento

  - Relacionais

Algoritmo	C++
< ≤ > ≥	< <= > >=
= ≠	== !=

Observação:

O resultado de uma comparação de dois valores pode ser 0 (falso) ou 1 (verdadeiro).

  - Lógicos (bit a bit)

Algoritmo	C++
complemento de um	~
e	&
ou-exclusivo	^
ou	|
deslocamento à direita	>>
deslocamento à esquerda	<<

Observação:
O resultado de uma operação lógica é um valor cujos bits são operados um a um de acordo com a álgebra de proposições.

  - Conectivos lógicos

Algoritmo	C++
não	!
e	&&
ou	| |

- Prioridade de operadores

Operador	Associação
( ) [ ] { }	à esquerda
! ~ ++ -- + - (tipo) * & **sizeof**	à direita
* / %	à esquerda
+ -	à esquerda
>> <<	à esquerda
< <= >= >	à esquerda
== !=	à esquerda
&	à esquerda
^	à esquerda
| |	à esquerda
&&	à esquerda
|	à esquerda
?:	à direita
= += -= *= /= %=	à direita
>>= <<= &= |= ^=	à direita
,	à esquerda

- Funções intrínsecas

As regras usadas na formação dos nomes dessas funções intrínsecas são as mesmas utilizadas para os nomes das variáveis.

Exemplo:

a = sin (b)

a - nome da variável que receberá o resultado da função;
sin - função (seno) predefinida do C++ ;
b - nome da variável que vai ser o argumento da função.

Nome (argumento)	Tipo de argumento	Descrição
sin (X)	**double**	seno (em radianos)
cos (X)	**double**	cosseno (em radianos)
atan(X)	**double**	arco tangente
sqrt(X)	**double**	raiz quadrada
exp (X)	**double**	exponencial de "e"
abs (X)	**int**	valor absoluto inteiro
fabs(X)	**double**	valor absoluto real
log (X)	**double**	logaritmo neperiano
log10 (X)	**double**	logaritmo neperiano
pow(X,Y)	**double, double**	elevar X a Y

A linguagem C++ dispõe de uma significativa biblioteca básica, com diversas funções além das aritméticas citadas acima.

- Expressões
  - Aritmética

    Exemplos:

Algoritmo	C++
10 + 15	10 + 15
543.15/3	543.15/3
(x + y + z)*a/z	((x + y + z) * a)/z

  - Lógica

    Exemplos:

Algoritmo	C++
A = 0	A == 0
$a \neq 1$	a != 1
$(A \geq 0)$ & $(a \leq 1)$	(A >= 0) && (a <= 1)

    Observação:
    Para efeito de clareza, ou para mudar a precedência de operadores, pode-se separar as proposições por parênteses.

- Estrutura de programa

```
// definições para pré-processamento

// definições globais

// definições de funções e procedimentos
  <tipo> <nome> (<lista de parâmetros>)
  {
    // definições locais

    // comandos
  }

// parte principal
   int main (<lista de parâmetros>)
  {
    // definições locais

    // comandos
    return 0;          // ou return EXIT_SUCCESS;
  }

// definições de funções e procedimentos (OPCIONAL)
  <tipo> <nome> (<lista de parâmetros>)
  {
    // definições locais

    // comandos
  }
```

- Comentários

  Comentários são precedidos pelos sinais // , ou /* */ envolvendo o texto.

  Exemplo:

```
int main ( void )
{
  // Este programa nao faz nada - comentario

  /*
    que também pode ser colocado assim
  */
  return 0;
}
```

- Atribuição

  - Atribuição simples

    Forma geral:

    <variável> = <expressão>;

    Exemplo:

```
x       =   0   ;
a       = 1.57;
letra   =  'A'  ;
```

  - Atribuição múltipla

    Forma geral:

    <variável 1> = <variável 2> = <expressão>;

    Exemplo:

```
x = y = 0;
```

    Observação:

    A execução inicia-se pela direita.

  - Atribuição composta

    Forma geral:

    <variável> <operador> = <expressão>;

    Exemplo:

```
i += 1    ou    i = i + 1
```

    Observação:

    Operadores permitidos: + - * / % >> << | & ^

- Atribuição condicional

  Forma geral:

  <variável> = <teste> ? <expressão 1>: <expressão 2>;

  Exemplo:
  x = (a < b) ? a: b;

- Descrição de entrada e saída

  - Entrada/Saída (padrão C++):

    Forma geral:

    cin >> <variável> ;
    cout << <expressão>;

    Observação:

    É necessário usar a definição abaixo (ou similar):

    #**include** <iostream>
    using namespace std;

  - Entrada/Saída formatada (padrão C):

    Forma geral:

    scanf (<formato>,<lista de apontadores>);
    printf (<formato>, <lista de itens>);

    Observação:

    É necessário usar a definição abaixo (ou similar):

    #**include** <stdio.h>

  - Especificação de formatos:

    Forma geral:

    %<sinais><<0><largura>>< . ><precisão><conversão>

    onde:

<sinais>		podem ser:
	'-'	- alinha a esquerda a saída
	'+'	- conversão de sinal (+ ou -)
	' '	- conversão de sinal (" " ou -)
	'#'	- começo com (0, 0x onde apropriado)
	<0>	- preenchimento com zeros
<largura>		- largura mínima do campo
<precisão>		- número máximo de caracteres - ou número de dígitos fracionários (neste caso é precedida por < . >)

<conversão> - pode ser:

caractere	argumento	conversão
d	**int**	para decimal
o	**int**	para octal
x	**int**	para hexadecimal
u	**int**	para decimal, sem sinal
c	**char/int**	para um caractere
s	***char**	para cadeia de caracteres
e	**float**	para real com expoente
f	**float**	para real sem expoente
g	**float**	para real
%		sinal %
ld	**long**	para decimal
lo	**long**	para octal
lx	**long**	para hexadecimal
le	**double**	para real com expoente
lf	**double**	para real sem expoente

Exemplos:

%5c - X do tipo caractere e com valor igual a 'A'

				A

%5d - X do tipo inteiro e com valor igual a 100

		1	0	0

%5.2f - X do tipo real e com valor igual a -1

-	1	.	0	0

Observação:

Se a largura (no exemplo, 5) não for suficiente para conter o número na sua forma de representação interna, o tamanho padrão para cada tipo será usado.

- Caracteres com funções especiais em formatos:

caractere	função
\0	fim da cadeia de caracteres
\n	fim de linha (LF)
\t	tabulação
\b	retrocesso (BS)
\r	retorno de carro (CR)
\f	avanço de carro (FF)
\\	barra invertida
\'	apóstrofo
\nnn	representação em octal
\xnn	representação em hexadecimal

Exemplo completo de programa:

```
#include <stdio.h>
#include <iostream>
using namespace std;

int main ( void )
{
  int i, j;

  cout << "Exemplo: ";
  cout << "\n";
  cout << "Digite um numero inteiro: ";
  cin   >>   j;
   i   =   j * 2 + 10;
  printf ( "%s %d\n","O resultado e\' igual a ", i );
  cout << "\nPRESSIONAR <Enter> para terminar.";
  getchar ( );
  return EXIT_SUCCESS;
}
```

Se fornecido o valor 5 para a variável ***j*** , o resultado será:

```
O resultado e' igual a 20
```

- Estruturas de controle
  - Sequência simples

    Forma geral:

Algoritmo	C++
<comando>	<comando> ;
<comando>	<comando> ;

    Observação:
    Em C++ todos os comandos são separados por ponto-e-vírgula.

- Estrutura alternativa
  - Alternativa simples

    Forma geral:

Algoritmo	C++

se <condição>
então
  <comandos>
fim se
	if (<condição>)
	{
	  <comandos> ;
	}

  - Alternativa dupla

    Forma geral:

Algoritmo	C++
se <condição> então <comandos 1> senão <comandos 2> fim se	**if** (<condição>) { <comandos 1> ; } **else** { <comandos 2> ; }

  - Alternativa múltipla

    Forma geral:

Algoritmo	C++

escolher <valor>
 	
<opção 1>:
  <comandos 1>
 	
<opção 2>:
  <comandos 2>
 	
. . .
<opção n-1>:
  <comandos N-1>
 	
senão
  <comandos N>
fim escolher
	switch <valor>
	{
	  case 1:
	  <comandos 1> ;
	  break;
	  case 2:
	  <comandos 2> ;
	  break;
	. . .
	  case (n-1):
	  <comandos N-1> ;
	  break;
	default:
	  <comandos N>
	}

    Observações:

    A variável de decisão deve ser de tipo escalar.

    Se o comando ***break*** for omitido, os comandos da próxima opção também serão executados.
    A indicação ***default*** é opcional.

- Estrutura repetitiva

  - Repetição com teste no início

    Forma geral:

Algoritmo	C++
repetir enquanto <condição>	**while** (<condição>)
	{
<comandos>	<comandos> ;
fim repetir	}

    Observação:
    A condição para execução é sempre verdadeira.

  - Repetição com teste no início e variação

    Forma geral:

Algoritmo	C++
repetir para	**for**(
<variável> = <valor inicial>	<valor inicial> ;
: <fim>	<teste de fim> ;
: <variação>	<variação>)
	{
<comandos>	<comandos> ;
fim repetir	}

    Observações:
    A condição para execução é sempre verdadeira.
    Em C++ , qualquer um dos elementos, ou mesmo todos, podem ser omitidos. Entretanto, se tal for preciso, recomenda-se o uso de outra estrutura mais apropriada.

- Repetição com teste no fim

    Forma geral:

Algoritmo	C++
repetir até <condição>	**do**
	{
<comandos>	<comandos> ;
fim repetir	} **while** (<condição>) ;

    Observação:
    A condição para execução é sempre verdadeira.

- Interrupções

  Em C++ , as repetições podem ser interrompidas, em sua sequência normal de execução através dos comandos:

  break; e **continue**;

  O comando ***break*** serve para interromper completamente uma repetição, passando o controle ao próximo comando após a estrutura repetitiva.
  O comando ***continue*** interrompe uma iteração, voltando ao início.