INTRODUÇÃO

A Informática nasceu da idéia de auxiliar o homem nos trabalhos rotineiros e repetitivos, em geral de cálculo e gerenciamento. E hoje é conceituada como ciência que estuda o tratamento automático da informação.

Assim, o elemento físico utilizado para o tratamento de dados e obtenção de informação, é o computador. E o processo de avaliação, contagem, ordenação e classificação de dados e informações é conhecido por computação.

Para nos comunicarmos com o computador devemos utilizar uma linguagem de programação, isto é, uma linguagem que o computador entenda. Temos dois tipos de linguagens:

- LINGUAGEM DE MÁQUINA: O computador, internamente, possui uma linguagem característica, baseada em grandezas matemáticas do sistema binário (0 e 1). Programar em linguagem de máquina significa comunicar-se com uma linguagem bem próxima a que a máquina entende.
- LINGUAGEM DE ALTO NÍVEL: São instruções semelhantes a linguagem natural que ao serem passadas para o computador são transferidas para linguagem de máquina para que possam ser atendidas.

Os tradutores são programas capazes de entender os programas escritos em linguagem de alto nível. Temos dois tipos de tradutores:

- COMPILADORES: Realizam a tradução de um conjunto completo de instruções criando um novo conjunto, todo em linguagem de máquina, que será usado na execução do programa.
- INTERPRETADORES: Realizam a tradução e imediata execução de uma instrução não gerando um novo conjunto em linguagem de máquina.

Algoritmo

Algoritmo é o conjunto de ações primitivas (instruções) organizadas de forma lógica, estruturada e bem definida, expressa em linguagem natural, que tem por finalidade resolver um problema, ou seja, é a descrição em português das ordens que o programa deve realizar.

Exemplo: Algoritmo de um saque em um caixa eletrônico:

1. Início
2. Passe o cartão
3. Abra a porta
4. Entre no caixa
5. Feche a porta
6. Insira o cartão no local indicado
7. Digite sua senha
8. Escolha a opção saque
9. Escolha o valor
10. Apanhe o dinheiro
11. Apanhe o comprovante e o cartão
12. Abra a porta
13. Saia do caixa
14. Feche a porta
15. Fim

Um algoritmo está completo se os seus comandos (instruções) forem do entendimento do seu destinatário, caso contrário, o comando não entendido deverá ser desdobrado em novos comandos, até que todos os comandos sejam entendidos. Este ato de desdobrar os comandos é chamado refinamento sucessivo do comando inicial.

Exemplo: A sequência de Fibonacci se define como tendo os dois primeiros termos iguais a 1 e cada termo seguinte é igual a soma dos dois termos imediatamente anteriores. Fazer um algoritmo que escreva os termos de Fibonacci inferiores a L.

Início

1) Escreva os termos de Fibonacci inferiores a L

Fim

Refinando 1 temos

1 Ref) Escreva os termos de Fibonacci inferiores a L

Receba o valor de L

2) Processe os dois primeiros termos

3) Processe os termos restantes

Fim Ref (Estrutura Sequencial)

Refinando 2 temos

2 Ref) Processe os dois primeiros termos

Atribua o valor 1 ao primeiro termo

Se ele for menor que L então

escreva-o

Atribua o valor 1 ao segundo termo

Se ele for menor que L então

escreva-o

Fim Ref (Estrutura Condicional)

Refinando 3 temos

3 Ref) Processe os termos restantes

Repita

Calcule o novo termo somando os 2 anteriores

Escreva o novo termo

Até que o novo termo seja maior ou igual a L

Fim Ref (Estrutura de Repetição)

Supondo que o destinatário entenda os comandos de atribuição e adição de dois termos, o algoritmo está completo e tem a seguinte estrutura:

Início

{ Dado de Entrada }

Receba o valor de L

{ Processamento dos dois primeiros termos }

Atribua o valor 1 ao primeiro termo

Se ele for menor que L então

escreva-o

Atribua o valor 1 ao segundo termo

Se ele for menor que L então

escreva-o

{ Processamento dos termos restantes }

Repita

Calcule o novo termo somando os 2 anteriores

Escreva o novo termo

Até que o novo termo seja maior ou igual a L

Fim

Assim, um algoritmo e seus refinamentos são formados por comandos (determinam ações a serem executadas) e por estruturas de controle (determinam a ordem em que os comandos devem ser executados, se devem ou não ser executados e quando devem ser repetidos).

Linguagem C

Um programa escrito na linguagem de programação C/C++ consiste em uma coleção de funções, sendo que a main() é a primeira função a ser executada. Além disso, C/C++ é case-sensitive, ou seja, diferencia letras maiúsculas de letras minúsculas (por exemplo, int Int). Um programa em C/C++ mínimo consiste em:

```
main () {

}
```

Este programa define a função main, que não possui argumentos e não faz nada. As chaves, { }, são usadas para expressar agrupamentos. No caso do exemplo, elas indicam o início e o fim do corpo da função main, que está vazia, isto é, não faz nada. Cada programa em C/C++ deve ter uma função main.

Assim, a estrutura genérica de um programa em C/C++ é a que se apresenta a seguir, podendo alguns dos elementos não existir:

Comandos do pré-processador.

Definições de tipos.

Protótipos de funções: declaração dos tipos de retorno e dos tipos dos parâmetros das funções.

Declaração de variáveis globais.

Um bloco de instruções principais: main.

Blocos de funções.

Documentação do programa: comentários.

Em C/C++, existem comandos que são processados durante a compilação do programa (pré-processador). Estes comandos são genericamente chamados de diretivas de compilação. Tais comandos informam ao compilador do C/C++ basicamente quais são as constantes simbólicas usadas no programa e quais bibliotecas devem ser anexadas ao programa executável.

A diretiva #include <filename> diz ao compilador para incluir na compilação do programa o arquivo filename. Geralmente estes arquivos contêm bibliotecas de funções ou rotinas do usuário. Bibliotecas são arquivos contendo várias funções que podem ser incorporadas aos programas. Os arquivos terminados em “.h” são chamados headers (ou cabeçalhos).

Estrutura Básica:

```
#include <nome_da_biblioteca>

void main( ) {

        bloco de comandos;

}
```

Identificadores

Em C/C++, os nomes de variáveis, funções, rótulos e vários outros objetos definidos pelo usuário são chamados de identificadores. A escolha dos nomes desses identificadores deve ser feita seguindo as regras:

Um identificador deve iniciar por uma letra ou por um "_" (underscore).

A partir do segundo caractere pode conter letras, números e underscore.

Deve-se usar nomes significativos dentro do contexto do programa.

Lembrar que C/C++ é uma linguagem case-sensitive (letras maiúsculas são diferentes de letras minúsculas).

Costuma-se usar maiúsculas e minúsculas para separar palavras.

Deve ser diferente das palavras reservadas.

Exemplos de identificadores:

Correto	**Incorreto**
Cont	1cont
Teste23	Oi!
usuario_1	usuario...1
RaioDoCirculo	Raio Do Circulo

As palavras reservadas não podem ser utilizadas como identificadores, pois são de uso restrito da linguagem C/C++ (comandos, estruturas, declarações, etc.). O conjunto de palavras reservadas do C/C++ padrão é:

auto, double, if, static, break, else, int, struct, case, entry, long, switch, char, extern, register, typedef, continue, float, return, union, default, for, sizeof, unsigned, do, goto, short, while.

Tipos Básicos de Dados

Cada linguagem de programação possui tipos de dados pré-definidos. Assim para desenvolvermos algoritmos utilizaremos os seguintes tipos de dados:

INTEIRO = armazena qualquer valor pertencente ao conjunto dos números inteiros.

REAL = armazena qualquer valor pertencente ao conjunto dos números reais.

CARACTERE = armazena apenas 1 elemento pertencente a um dos conjuntos abaixo:

{A, ..., Z}, {0, ..., 9}, {*, ?, &, =, +, $, -, /, ', ",}

LITERAL = Armazena 1 sequência de caracteres.

BOOLEANO = Armazena apenas as constantes FALSO ou VERDADEIRO.

Em C/C++ existem cinco tipos básicos de dados: caractere, inteiro, ponto flutuante, ponto flutuante de precisão dupla e sem valor (char, int, float, double e void, respectivamente). O tipo void declara explicitamente uma função que não retorna valor algum ou cria ponteiros genéricos. Valores do tipo char são normalmente usados para conter valores definidos pelo conjunto de caracteres ASCII.

Esta linguagem não possui o tipo de dado booleano, pois considera qualquer valor diferente de zero como sendo verdadeiro. C não possui também um tipo especial para armazenar cadeias de caracteres (literais, strings). Deve-se, quando necessário, utilizar um vetor contendo vários elementos do tipo char.

A partir dos cinco tipos básicos podemos definir outros tipos. A tabela a seguir apresenta os tipos de dados definidos no padrão ANSI.

TIPO	FAIXA DE VALORES	TAMANHO EM BYTES
char	-127 a 127	1
unsigned char	0 a 255	1
signed char	O mesmo que char	1
int	-32.767 a 32767	2
unsigned int	0 a 65.535	2
signed int	O mesmo que int	2
short int	O mesmo que int	2
unsigned short int	O mesmo que unsigned int	2
signed short int	O mesmo que short int e int	2
long int	-2.147.483.648 a 2.147.483.647	4
unsigned long int	0 a 4.294.967.295	4
signed long int	O mesmo que long int	4
float	3,4 E-38 a 3,4E+38	4
double	1,7 E-308 a 1,7E+308	8
long double	3,4 E-4.932 a 1,1E+4.932	10

Exceto o void, os tipos de dados básicos (como mostra a tabela) podem ter vários modificadores precedendo-os. Um modificador é utilizado para alterar o significado de um tipo básico para adaptá-lo mais precisamente às necessidades de diversas situações. A lista de modificadores é composta por: signed, unsigned, long e short. Todos estes podem ser aplicados aos tipos char e int, e o long também pode ser aplicado ao double.

O padrão ANSI elimina o long float porque é igual a double. O uso de signed com int é permitido, mas redundante, pois a declaração de um int assume um número com sinal. O uso mais importante de signed é para modificar o char.

Variáveis

Uma variável é uma posição de memória que pode ser identificada através de um nome, e é usada para guardar um valor. O conteúdo de uma variável pode ser alterado através de um comando de atribuição, ou seja, após uma atribuição a variável muda de valor. É interessante comentar que não há uma inicialização implícita na declaração, mas a variável pode ser inicializada na declaração. Todas as variáveis em C/C++ devem ser declaradas antes de serem usadas. A forma geral de uma declaração é:

tipo lista_de_variáveis;

sendo:

tipo = tipo de dado válido em C/C++.

lista_de_variáveis = pode consistir em um ou mais nomes de identificadores separados por vírgula.

A declaração de variáveis pode ser feita em três lugares básicos: dentro de funções (variáveis locais), na definição dos parâmetros das funções (parâmetros formais) e fora de todas as funções (variáveis globais).

Exemplos:

Algoritmo	**C**
x: real	float x;
y, z: real	float y, z;
SomaGeral: inteiro	int SomaGeral;
sexo: caractere	char sexo;
nome: literal [40]	char nome[40];

Note que em C/C++ após cada comando deve-se colocar ; (ponto e vírgula).

Constantes

São identificadores que não podem ter seus valores alterados durante a execução do programa. As constantes em C/C++ podem ser de qualquer um dos cinco tipos de dados básicos. Porém, a maneira como cada constante é representada depende do seu tipo. As constantes de caracteres são envolvidas por apóstrofos (‘) e, as cadeias de caracteres são representadas entre aspas (“).

Para criar uma constante existe o comando #define que, em geral é colocado no início do programa fonte, ou seja, no seu cabeçalho.

Exemplos:

Algoritmo:	**C**:
LARGURA_MÁXIMA = 50	#define LARGURA_MÁXIMA 50
FALSO = 0	#define FALSO 0
VERDADEIRO = 1	#define VERDADEIRO 1
CURSO = “CC”	#define CURSO “CC”
TERMINO = ‘T’	#define TERMINO ‘T’
VALOR_DE_PI = 3,1415	#define VALOR_DE_PI 3.1415

Observe que não se coloca ponto e vírgula após o valor.

Comandos de Entrada e Saída

Os principais comandos (funções) de entrada e saída são utilizados para ler do teclado e escrever no vídeo, respectivamente, o valor de variáveis. Para que o programa possa fazer uso desses comandos é necessário utilizar a biblioteca <stdio.h>.

O comando de entrada é utilizando para receber dados digitados pelo usuário. Os dados recebidos são armazenados em variáveis. Os comandos de entrada mais utilizados são:

scanf: lê todos os tipos de dados.

cin >>: lê todos os tipos de dados.

gets: lê uma sequência de caracteres (string) até que seja pressionado enter.

Apesar do comando scanf poder ser utilizado para qualquer tipo de dados, seu uso para char ou string pode resultar em problemas, devido ao fato de armazenar no buffer o comando <enter>. Este problema é resolvido se antes, ou logo após sua utilização, seja acrescentado o comando fflush(stdin);.

O comando de saída é utilizado para mostrar os dados na tela ou na impressora, e seus principais comandos são printf e cout <<. Para o printf, seus parâmetros são de dois tipos: o primeiro formado por caracteres que serão exibidos na tela, e o segundo contém comandos de formato que definem a maneira pela qual os argumentos subsequentes serão mostrados. O comando de formato começa por % e é seguido pelo código de formato. O cout << escreve todos os tipos de dados.

Os principais códigos para o scanf e para o printf são:

scanf	printf
%c – caractere (char)	%c – char
%d – números inteiros (int)	%d – int
%f – números reais (float)	%f – float
%s – sequência de caracteres	%s – sequência de caracteres

Os comandos de formato podem ser alterados, por exemplo, para especificação da largura mínima de campo ou do número de casas decimais.

Exemplos:

01)
```
#include <stdio.h>
main() {
    float x;
    x = 10.12345;
```

```
        printf("%f\n", x);
        printf("%5.2f\n", x);
        printf("Conteudo de x = %5.2f\n", x);
}
```

Saída)
```
10.123450
10.12
Conteudo de x = 10.12
```

02)
```
#include <iostream>
using namespace std;
main() {
        float x;
        x = 10.12345;
        cout << x << endl;
        cout << "Conteudo de x = " << x << endl;
}
```

Saída)
```
10.1235
Conteudo de x = 10.1235
```

03)
```
#include <stdio.h>
main() {
        int idade;
        char nome[30];
```

```
        printf("Digite sua idade: ");
        scanf("%d", &idade);
        printf("Digite seu nome: ");
        fflush(stdin);
        scanf("%s", nome);
        printf("%s sua idade eh %d anos.\n", nome, idade);
}
```

Saída)

```
Digite sua idade: 20
Digite seu nome: Anderson
Anderson sua idade eh 20 anos.
```

04)

```
#include <stdio.h>
#include <string.h>
main() {
        int idade;
        char nome[30];
        printf("Digite sua idade: ");
        scanf("%d", &idade);
        printf("Digite seu nome: ");
        fflush(stdin);
        gets(nome);
        printf("%s sua idade eh %d anos.\n", nome, idade);
}
```

Saída)
```
Digite sua idade: 20
Digite seu nome: Anderson
Anderson sua idade eh 20 anos.
```

05)
```
#include <iostream>
using namespace std;
main() {
	int idade;
	char nome[30];
	cout << "Digite sua idade: ";
	cin >> idade;
	cout << "Digite seu nome: ";
	cin >> nome;
	cout << nome << " sua idade eh " << idade << " anos." << endl;
}
```

Saída)
```
Digite sua idade: 20
Digite seu nome: Anderson
Anderson sua idade eh 20 anos.
```

Observe que em strings não se utiliza & na leitura do dado. As chaves { } indicam início e fim de blocos de comando.

É importante comentar que C/C++ possui constantes especiais de caracter de barra invertida para caracteres que não podem ser impressos. Os principais são:

\n – linha nova (line Feed - LF).

\t – tabulação horizontal (TAB).

\\ - barra invertida.

\a – alerta (beep).

\´ - aspas simples.

\" – aspas duplas.

\0 – nulo.

Comentários

Comentários são textos que podem ser inseridos em programas com o objetivo de documentá-los e não são analisados pelo compilador. Podem ocupar uma ou várias linhas devendo ser inseridos nos programas utilizando /* ... */ ou //.

Exemplos:

1)

```
// comentário
```

2)

```
/*
linhas de comentários
linhas de comentários
*/
```

Em (1) a região de comentário é aberta por // e encerrada automaticamente ao final da linha. Em (2) a região de comentários é aberta com /* e fechada com */.

Comando de Atribuição

Este comando é utilizado para atribuir valores ou operações às variáveis, sendo representado por = (sinal de igualdade).

A linguagem C/C++ permite que você atribua o mesmo valor a muitas variáveis usando atribuições múltiplas em um único comando.

Exemplos:

```
x = 4;

x = x + 2;

y = z = 2.5;

sexo = 'F';
```

Caso seja necessário armazenar uma cadeia de caracteres dentro de uma variável, deve-se utilizar uma função para manipulação de caracteres:

```
strcpy(nome, "João");
```

Para que seja possível a utilização da função strcpy deve-se inserir ao programa, por meio da diretiva #include, a biblioteca <string.h>.

Operadores

Em C/C++ temos quatro classes de operadores: aritméticos, relacionais, lógicos e bit a bit. Além disso, C tem alguns operadores especiais para tarefas particulares.

Operadores Aritméticos

Operador	Ação	Precedência
-- ++	Decremento e Incremento	Maior
* /	Multiplicação e Divisão	
%	Resto da Divisão Inteira	
- +	Subtração e Adição	Menor

Os operadores +, -, * e / trabalham em C/C++ da mesma maneira da maioria das outras linguagens.

Quando / é aplicado a um inteiro, qualquer resto é truncado.

O operador % retornará o resto da divisão inteira. Porém, % não pode ser usado nos tipos ponto flutuante.

Exemplo:

```
#include <iostream>

using namespace std;

main() {

    int x, y;

    x = 5;

    y = 2;

    cout << x/y << endl;

    cout << x%y << endl;

    x = 1;

    y = 2;

    cout << x/y << endl;

    cout << x%y << endl;

    cout << 1.0/2.0 << endl;

}
```

Saída

```
2

1

0

1

0.5
```

Os operadores de incremento e decremento são muito úteis. Por exemplo: x = x+1 é o mesmo que ++x ou x++, assim como x = x-1 é o mesmo que --x ou x--. Apesar dos operadores de incremento e decremento poderem ser utilizados como prefixo ou sufixo do operando, há uma diferença quando são usados em expressões. Quando um operador de incremento ou decremento precede seu operando, C executa a operação de incremento ou decremento antes de usar o valor do operando. Se o operador estiver após seu operando, C usará o valor do operando antes de incrementá-lo ou decrementá-lo. Por exemplo:

```
#include <iostream>

using namespace std;

main() {

        int x, y;

        int a, b, c, i = 3;

        a = b = c = 0;

        x = 10;

        y = ++x;

        cout << x << " " << y << endl;

        x = 10;

        y = x++;

        cout << x << " " << y << endl;

        a = i++;

        cout << a << " " << b << " " << c << " " << i << endl;

        b = ++i;

        cout << a << " " << b << " " << c << " " << i << endl;

        c = --i;

        cout << a << " " << b << " " << c << " " << i << endl;

}
```

Saída

11 11

11 10

3 0 0 4

3 5 0 5

3 5 4 4

Operadores Aritméticos de Atribuição

Operador	Exemplo	Comentário
+=	x += y	Equivale a x = x + y
-=	x -= y	Equivale a x = x – y
*=	x *= y	Equivale a x = x * y
/=	x /= y	Equivale a x = x / y
%=	x %= y	Equivale a x = x % y

Estes operadores realizam as operações aritméticas e posteriormente a operação de atribuição. Por exemplo: x += y, realiza a operação x + y e depois atribui seu resultado à variável x.

Operadores Relacionais

São operadores que comparam duas grandezas.

Operador	Operação
==	Igual
!=	Diferente
<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual

Operadores Lógicos

Os operadores lógicos e relacionais frequentemente trabalham juntos. A idéia de verdadeiro e falso é a base dos conceitos dos operadores lógicos e relacionais. Em C/C++, verdadeiro é qualquer valor diferente de zero. Falso é zero. As expressões que usam operadores lógicos ou relacionais devolvem zero para falso e um para verdadeiro.

Operador	Operação
&&	Conjunção (E)
||	Disjunção (OU)
!	Negação

Temos a seguinte tabela verdade para os operadores lógicos:

p	q	p && q	p || q	! p	! q
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Obs: Existem os operadores & e | que são utilizados para manipular ponteiros. Cuidado para não confundir!

A precedência para os operadores lógicos e relacionais é mostrada na tabela a seguir.

Operador	**Precedência**
!	Maior
> >= < <=	
== !=	
&&	
||	Menor

Prioridades

A linguagem C/C++ permite que sejam combinadas diversas operações em uma expressão, como por exemplo: 10 > 5 && !(10 < 9) || 3 <= 4. Neste caso, o resultado é verdadeiro.

Para que as operações possam ser efetuadas sem problemas, devemos seguir certas prioridades. Operadores de mesmo nível de precedência são avaliados da esquerda para a direita. Obviamente, parênteses podem ser usados para alterar a ordem de avaliação.

A tabela a seguir mostra a lista de precedência dos operadores mencionados anteriormente.

Operador	Precedência
++ --	Maior
* / %	
+ -	
funções pré definidas	
< <= > >=	
== !=	
!	
&&	
||	
= += -= *= /=	Menor

Exercícios

01) Faça um programa que receba quatro números inteiros, calcule e mostre a soma desses números.

02) Faça um programa que receba o salário de um funcionário, calcule e mostre o novo salário, sabendo-se que este sofreu um aumento de 25%.

03) Faça um programa que calcule a área de um círculo. Sabe-se que Área = $\pi * R^2$.

04) Faça um programa que receba um número positivo e maior que zero, calcule e mostre:

O número digitado ao quadrado.

O número digitado ao cubo.

A raiz quadrada do número digitado.

05) Escreva um programa para calcular e exibir a média ponderada de 2 notas dadas. A primeira nota tem peso 6 e a segunda nota tem peso 4.

06) Escreva um programa que leia duas variáveis inteiras e troque o conteúdo entre elas.