**Linguagem C**

Para entendermos os conceitos de linguagem C, precisamos primeiro compreender o que é um software, que não vemos nem sentimos, mas está ali. Para que um computador realize uma tarefa, precisamos que ele compreenda qual o problema a ser resolvido. Para isso precisamos traduzir para a linguagem de máquina, limites de configurações permitidas, diferentemente de uma receita de cozinha, para escrevermos um algoritmo, precisamos de uma linguagem exata e contendo as seguintes características:

* + - * Sempre dar alguma resposta;
      * Sempre dar a resposta correta e nunca uma resposta incorreta;
      * Terminar em um número finito de passos;
      * Trabalhar em todos os exemplos da classe de problemas que o algoritmo se propõe a resolver.

**(Tópicos do autor)**

No começo ficou muito trabalhoso, porque além de produzir o algoritmo pra resolver um problema em questão e, traduzir pra linguagem de máquina, ainda necessita testar, atualizar, fazer manutenção, contudo ficava mais complicado e trabalhoso o raciocínio. Então temos a importância das linguagens de programação de alto nível, onde são inseridos os algoritmos pra resolução de um problema, e por meio de compiladores, traduzem para a linguagem de máquina, otimizando o processo de desenvolvimento. Resumindo compiladores são programas que traduzem algoritmos, da linguagem humana para computadores e vice-versa.

**Algoritmos**

A definição de algoritmo é, uma sequência de passos para resolver um problema de forma clara e precisa. Todos nós sabemos construir um algoritmo, como exemplo a definição de um horário pra acordar, o melhor transporte para chegar ao trabalho. Existem três formas de representação de algoritmo na programação: linguagem natural como descrito anteriormente, fluxograma que é um conjunto de regras padronizadas, para resolver determinado problema e pseudo-linguagem que dispõe de uma linguagem intermediária entre algoritmo e codificação na linguagem.

**Tipos de dados**

Constantes e variáveis são tipos de dados armazenados num espaço de memória no computador, podendo ter valores ou não, os valores podem ser: inteiros, caracteres, strings, reais, etc. Constante é um espaço na memória que não pode ser alterado por exemplo, atribuir o valor de pi (3,14) a um espaço de memória. Variáveis são espaços de memória que podem ser alterados os valores a eles atribuído durante a execução do programa.

Para definirmos uma variável, precisamos atribuir um nome a ela, porem existem algumas regras para a definição dos nomes nas variáveis:

. Nomes de variável só podem conter letras, dígitos e o caractere ‘\_’;

. Todo primeiro caractere deve ser sempre uma letra ou o caractere ‘\_’;

. Letras maiúsculas e minúsculas são consideradas caracteres diferentes, isto é, C diferencia a caixa das letras

. Palavras reservadas não podem ser usadas como nome de variáveis. . . Palavras reservadas são palavras usadas para indicar os comandos da linguagem, tipos de dados e outras funções.

**(Regras retiradas do autor)**

**Entrada e saída do console**

Para que um programa tenha entrada de dados pelo teclado, saída pelo monitor e saída de erros, essas ações são executadas com o uso de bibliotecas que já contém funções especiais. O projeto deve conter a declaração: **#include <stdio.h>**, onde #include no começo do programa determina que o que estiver dentro dos sinais ‘< >’ seja executado da biblioteca como se fosse parte dos comandos do programa.

**O comando printf:**

O comando de saída ‘printf’ tem como função a impressão de dados específicos na saída padrão, normalmente é um monitor, por exemplo:

#include <stdio.h>

int main () {

int numero = 1994;

printf(“Eu nasci em %d”, numero);

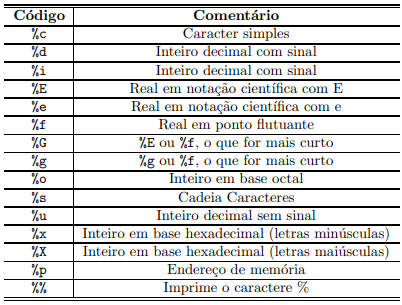
return 0;

}

Este programa deverá imprimir na tela: Eu nasci em 1994, apresentando o que está entre aspas duplas e logo após o sinal de ‘%’ mais o código de conversão, nesse caso como é um número inteiro, é seguido da letra ‘d’, a seguir estão os códigos de conversão dos tipos de dados das variáveis ou constantes que, a linguagem separa na memória do computador para armazenar os valores.

**Códigos de conversão**

Observe a tabela:

 **(Tabela retirada do autor)**

**O comando scanf**

O comando scanf é usado como método de entrada de dados pelo teclado no modelo:

int v;

float s;

scanf (“%d %f “, &v, &s);

Onde a variável ‘v’ está definida na memória como número inteiro e a variável ‘s’ como um número real com ponto flutuante, sobre inserção do comando scanf, o primeiro valor digitado sendo inteiro é armazenado no espaço de memória reservado como inteiro pela variável ‘v’ e o segundo valor flutuante é armazenado no espaço de memória flutuante reservado pela variável ‘s’.

O separador do comando scanf é o espaço em branco, então, caso precisemos armazenar em uma variável um nome completo ou uma cadeia de strings como, por exemplo, um artigo, teríamos que usar o comando ‘gets’

**Comandos gets e puts**

O comando gets armazena strings digitadas com espaço entre elas, separando apenas quando pressionada a tecla ‘enter’, ao pressionar a tecla ‘enter’, o programa converte para NULL e armazena o que foi digitado antes. Para imprimir uma variável também pode ser usada a função ‘puts’, é semelhante a função ‘printf’, porém, ela tem um comando ‘\n’ de pular linha, embutido na função, pulando uma linha sem precisar digitar o comando.

**Operadores**

A seguir veremos como usar a forma correta de operadores e variáveis.

**Operador de atribuição**

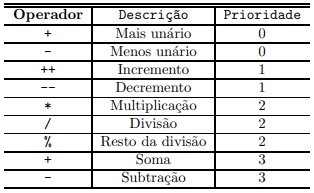
Operador de atribuição é definido na linguagem C com o símbolo de igual (=), mas tenhamos cuidado, porque isso não representa igualdade e sim atribuição, também é possível declarar mais de uma variável com o mesmo valor na mesma linha, não esquecendo de sinalizar pro compilador do caractere (;) no final da declaração, vejamos um exemplo:

x = z = 9;

Não significa que ‘x’ é igual a ‘z’ que é igual a 9. Significa que a variável ‘z’ recebe o valor 9 e, logo após a variável ‘a’ recebe o valor que foi atribuído a variável ‘b’ (9). A princípio não apresenta muitas diferenças, mas não confundamos, atribuição (=), com igualdade (==) que veremos mais à frente.

**Operadores aritméticos**

Na tabela a seguir vejamos os operadores aritméticos e suas ordens de precedência.

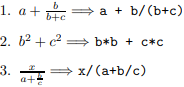


**(Tabela retirada do autor)**

As expressões na linguagem C devem ser feitas de forma linear para facilitar a compreensão do programa e, também porque não existem alguns operadores no teclado habitual por exemplo o de divisão usamos a/b onde ‘a’ é dividido por ‘b’.

Os parênteses são importantes e podem ter parênteses dentro de parênteses onde é executada a operação que estiver dentro dos parênteses mais internos na expressão.

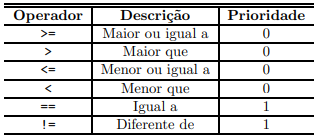
Veja alguns exemplos de conversões de expressão pra linguagem C:



**(Expressões retiradas do autor)**

**Operadores relacionais**

Na tabela a seguir vejamos os operadores relacionais e, suas ordens de precedência entre eles. Mais à frente veremos a ordem de precedência entre todos os operadores.



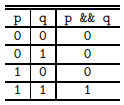
**(Tabela retirada do autor)**

**Operadores lógicos**

Nos operadores lógicos temos como saída das expressões, valor verdadeiro ou falso, em linguagem C, falso é representado pelo valor 0 e, verdadeiro por qualquer outro valor.

**E lógico**

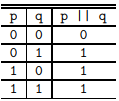
O operador lógico E na linguagem C é representado por ‘&&’. Na tabela verdade, uma condição só pode ser verdadeira se todos os testes lógicos forem verdadeiros, vejamos na tabela a seguir.



**(Tabela retirada do autor)**

**OU lógico**

O operador lógico OU na linguagem C é representado por ‘| |’. Na tabela verdade, uma condição só pode ser verdadeira se algum dos testes lógicos for verdadeiro, vejamos na tabela a seguir:



**(Tabela retirada do autor)**

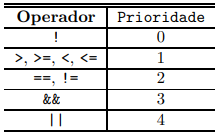
**Não lógico**

O operador lógico de negação na linguagem C é representado por ‘!’. Na tabela verdade ele inverte o valor da expressão, caso a saída da expressão seja verdadeira, o operador a torna falsa e vice-versa.



**(Tabela retirada do autor)**

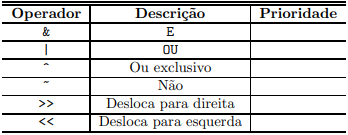
Veja a tabela destes operadores em ordem de precedência:



**(Tabela retirada do autor)**

**Operadores com bits**

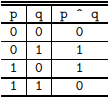
Os operadores com bits são demonstrados na tabela a seguir e só podem ser usadas nos tipos de variáveis int, long, long long e int.



**(Tabela retirada do autor)**

**Operador OU exclusivo**

O operador lógico OU exclusivo na linguagem C é representado por ‘^’. Na tabela verdade é considerado verdadeiro somente se os dois testes lógicos forem diferentes.



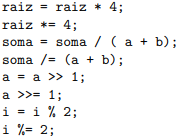
**(Tabela retirada do autor)**

**Operadores de Atribuição Composta**

Os operadores de atribuição composta, são uma otimização de código para simplificar cada vez mais, o objetivo da programação é sempre o caminho mais fácil, veja um exemplo:

x = x + 200 é a mesma coisa que: x += 200

Vejamos mais alguns:



**(Expressões retiradas do autor)**

**Operador vírgula**

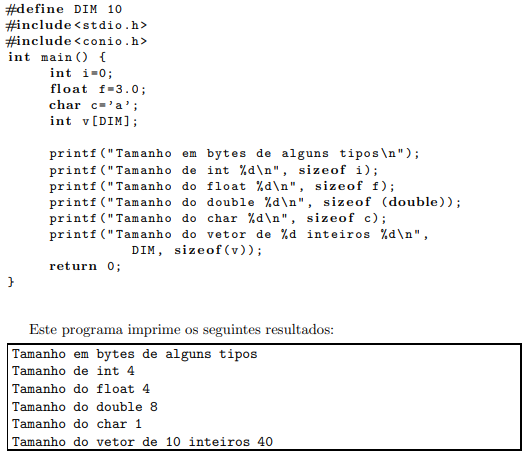
Este operador ‘,’ é definido para separar expressões onde a ordem da execução de resolução das expressões fica da esquerda pra direita veja um exemplo:

carro = (casa=20, casa-10);

Neste exemplo a variável casa recebe o valor 20, logo após a variável carro recebe casa-10 que é igual a 10. Contudo, a variável casa fica com o valor 20 e, a variável carro com o valor 10.

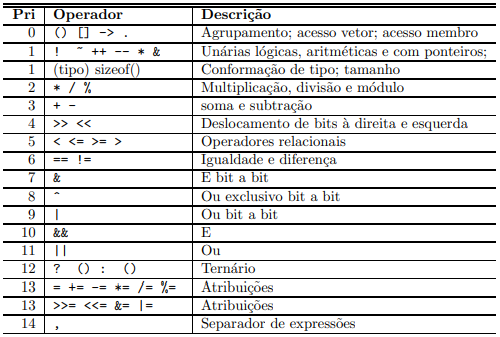
**Operador sizeof()**

Este operador retorna um número em bytes de uma variável de tipo atribuído entre parênteses ou o resultado de uma expressão vejamos num exemplo a seguir:

 **(Programa retirado do autor)**

**Tabela de precedência dos operadores da linguagem C**

Alguns dos operadores listados serão vistos mais à frente.

 **(Tabela retirada do autor)**

**Comandos de controle**

Comandos de controle são comandos pré-definidos pela linguagem, para direcionarmos o programa por caminhos distintos, de acordo com o resultado de testes.

**Comando if**

O comando “if”, em português significa “se” define entre duas opções de caminho, onde o programa deve seguir de acordo com o resultado de alguma expressão, vejamos um exemplo a seguir:

int i, j = 10;

int k = i + j;

if (k <= 25) {

printf(“%d + %d e menor ou igual a 25”, i, j);

}

Neste trecho de código, o programa executa o bloco de operações da função ‘if’, por conta da variável ‘k’ ser igual a 20, ou seja, menor ou igual a 25, como solicitado na condicional da função. Neste caso como é apenas uma operação não precisava das chaves limitando os blocos, mas apresentamos aqui para melhor entendimento. Caso o valor na variável ‘k’ fosse 26, por exemplo, o programa seguiria o fluxo e não executava o bloco de código do comando ‘if’.

**Comando else**

O comando “else”, significa em português “outro”, ou seja, não é independente, só podemos usar um else depois de um if, no else não vai condição como no if, apenas o bloco de comando, por exemplo:

int i, j = 20;

int k = i + j;

if (k <= 25) {

printf(“%d + %d e menor ou igual a 25”, i, j);

} else {

printf(“%d + %d e maior que 25”, i, j);

}

**Comando else if**

Caso precisemos de mais condicionais podemos usar diversos if’s. Porém, para otimizar o processo de testes podemos utilizar ‘else if’ numa mesma estrutura de código, vejamos os dois modelos num trecho de código mais complexo à seguir:

int i, j = 30;

int k = i + j;

if (k <= 25) {

printf(“%d + %d e menor ou igual a 25”, i, j);

}

if (k <= 35){

printf(“%d + %d e menor ou igual a 35”, i, j);

} else if (k <=45) {

printf(“%d + %d e menor ou igual a 45”, i, j);

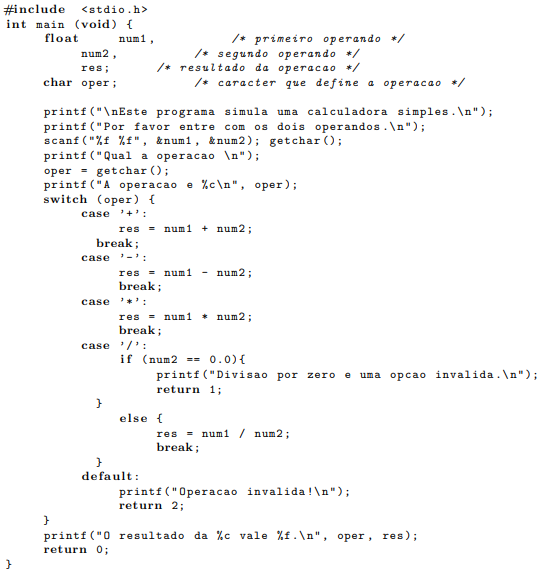
} else {

printf(“%d + %d e maior que 45”, i, j);

}

**Comando switch**

No comando switch podemos aninhar de forma mais simples as condicionais, porém com algumas restrições, nela só podemos testar igualdade e, não podemos por duas constantes iguais no mesmo case. Cases são as condicionais que definimos no comando, o programa percorre todo o switch até que encontre uma saída de desvio. A saída de desvio comum do comando é o ‘break’, quando encontrado ele desvia o foco para o próximo case. O comando ‘default’ é opcional, caso ele não encontre nenhuma igualdade nos cases executa o bloco de códigos do default e, se ele não aparecer nenhum comando será executado, vejamos um exemplo:



**(Programa retirado do autor)**

**Operador ternário**

No operador ternário, é dado esse nome por que precisa de três elementos para ser analisado. Vejamos um exemplo a seguir:

Teste1 ? Teste2 : Teste3

#include <stdio.h>

int main (){

int n1 = 2;

int n2 = 1;

menor = (n1 < n2) ? n1 : n2;

printf(“O menor numero e: %d”, menor);

return 0;

}

Neste caso o resultado da função é: “O menor número é 1”, onde 1 representa a variável n2, que foi adicionada na saída ao ser analisada como a variável de menor valor pelo comando ternário.

**Laços de repetição**

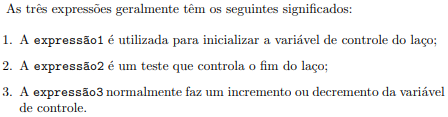
São comandos de repetição normalmente com um contador, que determina o número de vezes que o programa deve repetir aquela rotina de testes.

**Comando for**

Imaginemos da seguinte forma, imagine que teríamos que escrever nossos movimentos ao almoçar: pega comida com o garfo, leva o garfo até a boca, mastiga a comida. Essa rotina se repete por várias vezes então, por exemplo, seria mais fácil escrever ‘fiz essa rotina por 20 vezes’. A estrutura do laço for é a seguinte:

for(1ª expressão, 2ª expressão, 3ª expressão)

bloco de funções;



**(Regras retiradas do autor)**

**Comando while**

No comando while o resultado do teste é verdadeiro ou falso, ou seja, 0 ou diferente de 0. É feita a análise, caso o resultado seja falso, nem é executado o bloco de código. Vejamos o exemplo:

int main () {

int contador = 1;

while (contador <= 3){

printf(“%d\n”, contador);

contador++;

}

return 0;

}

**Comando do-while**

Semelhante ao comando while, porém no comando do while o bloco de códigos é executado pelo menos uma vez e depois é feita a análise, vejamos um exemplo a seguir:

int main () {

int contador = 1;

do{

printf(“%d\n”, contador);

contador++;

} while (contador <= 3);

return 0;

}

**Comandos de desvio**

Na linguagem C existe o comando ‘break’ como vimos pra desviar o programa na operação usando switch, mas também serve para desviar um laço de repetição vejamos no exemplo de trecho de código for a seguir:

for (a = 0; a < 10; a++){

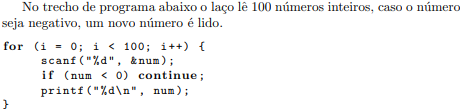
printf(“%d”, a);

if (a > 5)

break;

}

Também existe na linguagem C o comando ‘continue’ semelhante ao comando break, com a diferença que ele interrompe apenas o laço no momento da iteração atual, nos comandos ‘while’ e ‘do while’ o programa passa a atuar na fase de análise. Vejamos um exemplo num laço ‘for’ a seguir:

**(Exemplo retirado do autor)**

**Comando goto**

Este comando serve para saltar o programa para um rótulo definido em qualquer parte da função. Não é muito recomendado usar por desfazer os princípios de código limpo, quebrando a sequência do algoritmo, mas em alguns casos é necessário, vejamos a seguir como chamá-lo na função:

 **(Exemplo retirado do autor)**

**Função exit ()**

A função exit() tem por finalidade a execução do fim do programa, devolvendo o controle ao sistema operacional, naturalmente quando um valor diferente de zero (0) é retornado, é sinal que não houve erros na finalização da aplicação, caso o valor seja zero (0) é sinal de erro na aplicação, vejamos como exemplo a sua aplicação:



**(Exemplo retirado do autor)**

**Comando return**

Este comando, tem a funcionalidade de retornar uma expressão para a função do programa que a solicitou. Em uma função de tipo void não é usada, porque não é uma função de retorno de valores.

**Cadeias de caracteres e vetores**

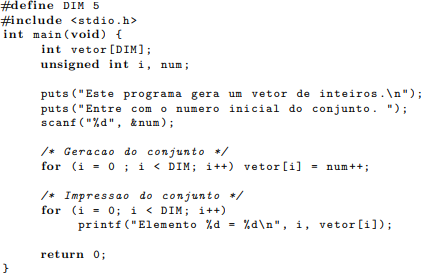
Os vetores são muito usuais, pois são capazes de armazenar uma sequência de dados e, podermos ter referências de cada dado pelo índice

Nos vetores unidimensionais temos a seguinte estrutura:

tipo nome [tamanho];

**(exemplo retirado do autor)**

Tipo, refere-se ao tipo de dado a ser armazenado, nome é a identificação do vetor e tamanho é o número de índices que o vetor armazena. Vejamos a seguir um programa completo que, “*calcula um produto escalar de dois vetores inteiros (retirado do autor p. 108)”*, com a nova biblioteca de busca: #define DIM 5, que define uma constante pra referenciar o número de elementos do vetor, podendo ser alterada de acordo com a necessidade do programa.

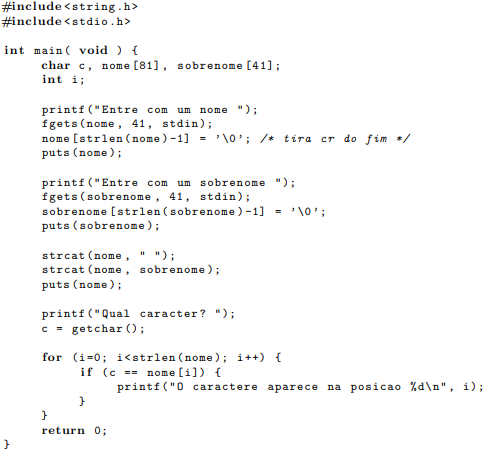


**(Programa retirado do autor)**

**String**

Uma string é uma cadeia de caracteres, ela sempre é terminada com o índice ‘\0’ para indicar o fim no armazenamento do vetor, por exemplo, caso a string tenha 100 caracteres, precisará de um vetor de 41 espaços de índices.

Existem funções para tratamento de strings para melhor desenvolvimento de programação, para isso, é necessário a importação da biblioteca ‘<string.h>’, vejamos um programa envolvendo strings:



**(Exemplo retirado do autor)**

**Vetores multidimensionais**

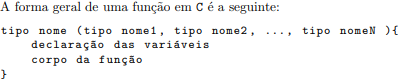
Vetores multidimensionais são vetores com grupos separados e/ou aninhados, onde a formatação de programação se baseia da seguinte forma:



**(Exemplo retirado do autor)**

**Funções**

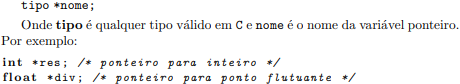
Funções são instruções de comando agrupadas, em C não existe um programa principal, existe apenas funções. A única regra é que a função ‘*main’* seja a primeira a ser executada. Existem hoje as bibliotecas na programação que já foram testadas por outros programadores, diminuindo assim os erros que podem ser encontrados. Essa facilidade serve para a otimização no desenvolvimento do programa esperado, assim como uma pesquisa documentada, para que os próximos pesquisadores tenham conhecimento das experiências e, seguirem com mais bagagem de referências na pesquisa. Vejamos um conceito sobre funções a seguir:



**(Conceito retirado do autor)**

**Ponteiros**

O ponteiro é uma variável que armazena endereço, e não o conteúdo de uma variável, um ponteiro é muito usual quando o programa necessita de mais de uma resposta, vejamos a maneira geral de iniciar um ponteiro na programação.

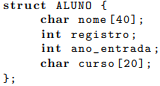


**(Regra retirada do autor)**

**Estruturas**

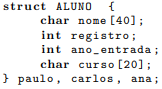
Uma estrutura é um agrupamento de variáveis podendo ser de diferentes tipos ou não, como exemplo, podemos imaginar um cadastro de empresas que contém nome, cnpj, etc, ou seja, otimiza a manipulação e organização de dados diferentes na estrutura.

Pra declararmos uma estrutura usamos a palavra-chave *‘struct’*, vejamos um exemplo de dados de um aluno a seguir:



**(Exemplo retirado do autor)**

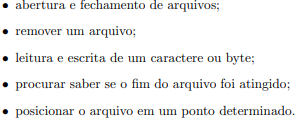
Nesse exemplo temos apenas uma reserva de memória para os seguintes dados de um aluno, na linguagem C, podemos declarar as estruturas juntamente com as variáveis, otimizando o desenvolvimento vejamos:



**(Exemplo retirado do autor)**

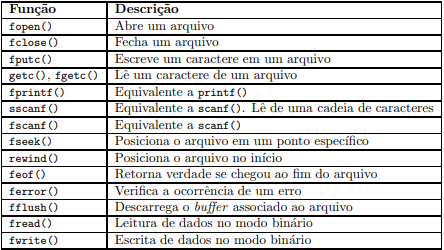
**Manipulação de arquivos**

Arquivo, é um espaço de memória, que pode ser armazenado, porém, para manipularmos, se faz necessário acessar os dados do arquivo, uma prática que dependendo do tipo de informação transportada, seja crucial uma segurança da informação, de modo que não comprometa o sistema. Vejamos algumas práticas habituais de manipulação de arquivos.



**(Práticas retiradas do autor)**

Como tudo em C é realizado por meio de funções, vejamos também as funções de entrada e saída de arquivos:



**(Funções retiradas do autor)**

**Referências**

DE OLIVEIRA CRUZ, Adriano Joaquim. Curso de Linguagem C Em Construçao v0. 001. 2007.