Linguagem C:

O que realmente é necessário para o início da codificação em ‘C’?

Sabemos que a prática e a perseverança em determinados temas podem nos tornar em verdadeiros profissionais em quaisquer que sejam a área de atuação. Mas sabemos que um bom embasamento científico colabora para esse desenvolvimento profissional, tendo isso em vista, por onde começar estudar? O que fazer e como fazer?  
 Essa é a pergunta que estou fazendo pra mim nesse momento, e em contato com meus amigos que já trabalham na área, vou desenvolver esse documento conforme eu avanço na linguagem C e na codificação em geral, quem sabe um dia posto isso ou não. Não sei. Mas se esse documento chegou em você, leia-o, mas não leve à risca tudo o que for dito aqui, porque isso não é um artigo científico e sim um “diário” das minhas experiências, ainda juvenis na programação.

Devemos começar falar sobre a importância dessa linguagem para os dias atuais e como ela chegou no nível que está e como influenciou linguagens novas a serem o que são hoje.

A linguagem C teve sua origem em 1969 e foi concluída por volta de 1973 por Dennis Ritchie e Ken Tompson, criada pela Bell Telephone Labs, com a ideia de criar um sistema operacional, e ele foi criado, o famoso ‘Unix’. A linguagem C foi inspirada por linguagens como Algol e a linguagem B (e isso originou esse nome de ‘C’) E então a linguagem C virou uma sensação entre os programadores do momento.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Estrutura** | **Propósito Geral** | **Portabilidade** | **Popularidade** | **Nível Médio** |
| **Padronizada** | **Sintaxe Simples** | **Case Sensitive** | **Variedade de bibli. com n funções** | **Grande influência no dev de outras ling.** |

((Isso é o que eu lembro até aqui, caso eu aprenda algo novo, colocarei neste doc.))

As linguagens eram definidas em dois tipos:  
A de Alto nível e a de baixo nível.  
A de baixo nível estava mais próxima da   
linguagem de máquina, ou seja, a linguagem  
binária. “0010101110011001”  
Já a linguagem de alto nível estava mais próxima  
da nossa linguagem e podia ser interpretada   
por um humano.

Códigos e bibliotecas fundamentais para o início:

**#include:** Este comando tem o poder de incluir uma biblioteca já existente ou não no programa a ser criado.

<stdio.h> ‘’ <stdlib.h> <= Estes são exemplos de biblioteca.  
standard input and output ‘’ standard library.

**Stdbool.h:** inclui uma biblioteca do tipo lógico ‘bool’. Essa biblioteca foi incluída na versão ISO C99.

Caso o programa da ling C seja um pouco mais antigo, a biblioteca deverá ser incluída manualmente usando o comando: **typedef enum {false=0, true=1} logico;**

**Int: Inclui um valor INTEIRO a variável ‘int’.**

**Float: inclui um valor DECIMAL a variável ‘float’.**

**Char: inclui caracteres ao programa. 1byte (8bits) – 0 - 255**

**Printf: imprime uma informação na tela do usuário.**

**Scanf: lê uma informação na tela do usuário.**

**Char: inclui caracteres ao programa.**

**While: comandos que fazem um ‘loop’ interminavelmente ou não no programa. “flag” pode ser usado para determinar o fim do loop, seja com uma tecla ou um valor especificado no código.**

while (condição) {

instruções executadas enquanto condição verdadeira}

**for:** mesma coisa que o while, porém mais simples. E podendo realizar uma estrutura do tipo while {for}. Exemplo:

int main ()

{

int cont;

char cod;

float nota, media, total;

total = 0;

cod = 'C';

while (cod != 'F') {

for (cont = 1; cont <=4; cont++){

printf ("Digite a nota: ");

scanf ("%f", &nota);

total = total + nota;

}

media = total / 4;

printf("Nota média: %.2f\n\n", media);

total = 0;

nota = 0;

printf ("Deseja calcular a média de mais um aluno? Pressione qualquer tecla ou F para encerrar \n");

cod = getche ();

}

return 0;

}

**If/else: ‘Se/então’ {Se a condição for verdadeira, então o comando executa algo. Se for falso o programa chega ao fim.} {IF aninhado: segundo if dentro do else}**

**Bool: tipo lógico, daquele que denota se a info. É ‘true’ or ‘false’.**

**/n: pula uma linha.**

**/t: esqueci**

**Pow(): potência**

**Getche (): captura um comando de tecla pra ser executado dentro do programa.**

**{} = executa um comando dentro das chaves**

**[] = usado para incluir vetores (ainda não estudei isso direito)**

**“” = inclui uma frase ou palavras dentro do comando para ser printado e/ou lido na tela.**

**& =colocado antes do alvo no ‘scanf’ (EX: &número) para ser printado o resultado na tela.**

**%\*c = o asterisco no meio do comando na função scanf diz para ignorar o caractere.**

**% = le o numero de uma variável já existente no programa.**

* **A declaração de casa a partir de números positivos (direita) ou negativos (esquerda) é essencial para formatação do programa.**
* **A precisão na exibição de dados pode ser configurada a partir de número depois do símbolo ‘%’ Ex: %8.5 => Onde ‘8’ é o tamanho total e ‘5’ é o que será realmente printado na tela.**

**Funções:**

As funções como já dizem, incluem comandos essenciais dentro do programa. Temos alguns exemplos como o void, main e até mesmo o printf e scanf são funções importantes dentro da linguagem C. As funções devem ser bem definidas e simples. Caso a função seja de complexidade alta, será necessário fragmentar ela em funções menores e mais simples pra evitar possíveis atritos ou conflitos. O printf por exemplo é uma função objetiva, ela formata e escreve os caracteres na tela, viu? Simples e objetivo e nossas funções devem ser assim, para que o processador tenha facilidade ao executar.

**Void:** inclui uma função sem retorno no programa. E você pode criar funções variadas a partir dessa. (vide exercício salvo como ‘funções’ no local: C:\Users\Rafael\Desktop\code)

Escopo das variáveis:

As **variáveis locais** só podem ser demonstradas ou inseridas dentro de uma função, fora desta função ela não aparecerá e não poderá ser inserida.  
 Os **parâmetros de** **função** não são diferentes das variáveis locais, sendo limitadas também a uma função.

E as **variáveis globais** são as variáveis que tem toda a liberdade de inserção dentro do programa, seja dentro ou fora das funções presentes. Na linguagem C você pode ter duas variáveis do mesmo nome, porém é um tipo perigoso de se manusear variáveis, uma vez que pode ser confundido facilmente, então neste modo, recomenda-se cautela.

Bubblesort:

O bubblesort é um mecanismo de ordenação de vetores que tem uma mecânica simples, e muito objetiva. Ela simplesmente faz a comparação dos valores e a substituição deles pela forma como é definido no código. Veremos a seguir o código que executa essa comparação e a coloca em prática a seguir.

**// Algoritmo de ordenação Bubblesort:**

**for (contador = 1; contador < TAM; contador++) {**

**for (i = 0; i < TAM - 1; i++) {**

**if (numeros[i] > numeros[i + 1]) {**

**aux = numeros[i];**

**numeros[i] = numeros[i + 1];**

**numeros[i + 1] = aux;**

O **TAM** foi definido por uma inclusão no cabeçalho chamado: #define TAM 10, isso define um número de vetores.

Em comparação com outros métodos existentes, a classificação Bubblesort é, no geral, lenta, mas é o mais simples dos algoritmos de classificação, e encontra aplicação em determinadas situações, como a classificação de arquivos pequenos e conjuntos de dados que já se encontram semi-classificados.

O algoritmo bubblesort funciona, de forma simplificada, executando duas tarefas principais, que são executadas em loop até que os dados estejam totalmente ordenados (classificados). São elas:

* Comparação de itens adjacentes
* Troca de posição dos itens, quando for necessário

Matrizes:

As matrizes são praticamente outro tipo de arrays, porém elas tem colunas x linhas e são definidas por [5] x [4], ou seja, 5 colunas e 4 linhas somando 20 espaços para preenchimento. O preenchimento dessa matriz pode ser inserido como no exemplo a seguir:

#define NUM\_L 4

#define NUM\_C 5

**float notas[5][4] = {{ 7.5, 6.8, 9.6, 6.7 },**

**{ 6.5, 6.3, 8.4, 7.6 },**

**{ 5.7, 8.6, 9.0, 4.5 },**

**{ 4.5, 5.8, 6.8, 7.0 },**

**{ 3.6, 7.6, 8.1, 6.5 } };**

E então esses valores decimais serão exibidos em forma de tabela no programa.

Arquivos de cabeçalho:

Os arquivos de cabeçalho são essenciais para atribuir determinadas funções a um programa, sem que ocupe tanto espaço no código fonte. E claro facilita a vida do programador, claro se for usado de forma eficaz. Como Deitel disse em seu livro “Não tente reinventar a roda, use a que já existe”.

Para criar um arquivo de cabeçalho deve iniciar um projeto ‘**.h’** e em seguida começar pelo comando ‘**infndef**’ e colocar o nome do arquivo a seguir. Em seguida #defini e depois dar um #endif para concluir a inclusão do comando como vamos ver a seguir:

#infndef CALCULOS\_H\_INCLUDED

#define CALCULOS\_H\_INCLUDED

//aqui será incluído os comandos da função

#define \_PI 3.14159 (Aqui eu defini uma constante que poderá ser exibido na criação de um programa)

Int **quadrado** (int x);

Int **cubo** (int x);

#endif //aqui se encerra a função

Então aqui eu criei um arquivo de cabeçalho e em seguida veremos como incluir isso em um programa. Mas primeiro vamos criar um outro arquivo chamado cálculos.c e incluir a biblioteca que acabamos de criar.

#include “cálculos.h” <= **está entre aspas porque nesse caso não se usa o “><”.**

Int **quadrado** (int x)

{

Return x \* x;

}

Int **cubo** (int x)

{

Return x \* x \* x;

}

Pronto nosso cabeçalho está pronto e vamos incluir isso no projeto principal “main”.

**#include <stdio.h>  
#include <stdlib.h>  
#include “cálculos.h”**  
int main ()  
{  
 printf(“Usando headers\n”);  
 **int** y = 5;  
  **int** quad = quadrado (Y);  
 **int** cub = cubo (Y);  
  
 printf (“quadrado de %d: %d\n”, y, quad);  
 printf (“Cubo de %d: %d\n”, y, cub);  
 printf (“Numero PI %f: %f\n”, \_PI);  
  
 **return** 0;  
}

Dessa forma podemos criar diferentes headers e com diversas funções pré estabelecidas. Mas sempre lembrando das boas práticas de programação, pra não acabar fazendo uma bagunça.