

APRENDENDO A PROGRAMAR

Introdução à programação

Nesta apostila vamos aprender como construir nossas próprias aplicações usando uma linguagem de programação. A linguagem que iremos utilizar será a C. Para programar em C, não precisamos de nada além de um editor de textos ASCII (Notepad do Windows, por exemplo) e de um compilador, que vai transformar os códigos que programarmos em linguagem de máquina (binário).

Existem vários compiladores para C, mas vamos usar nesta apostila o MingW, que é uma versão para Windows do compilador gratuito GCC para Linux.

Para facilitar o trabalho dos programadores, foram criadas IDEs (Interfaces de desenvolvimento), que unem editor, compilador e outras ferramentas. Existem várias IDEs gratuitas. A que iremos utilizar se chama Code::Blocks e pode ser baixada em:

http://prdownloads.sourceforge.net/codeblocks/codeblocks-1.0rc2_mingw.exe?download

CAPÍTULO I – Programação teórica

Antes de aprender a linguagem C, vamos ter uma rápida aula sobre lógica de programação, para que possamos desenvolver algoritmos do que pretendemos programar, tendo assim uma lógica aplicável a qualquer linguagem de programação.

Um algoritmo tem a mesma organização de um código-fonte, mas não pode ser compilado, pois não pertence a linguagem nenhuma, por isso é chamado de pseudo-código. Vamos tentar resolver um problema: como proceder para se trocar o pneu de um carro?

- 1- Sair do carro
- 2- Abrir o porta-malas
- 3- Pegar o macaco, estepe e chave de roda
- 4- Usar a chave para tirar os parafusos do pneu furado
- 5- Posicionar o macaco e levantar o carro
- 6- Tirar o pneu furado e colocar o estepe
- 7- Colocar os parafusos no novo pneu
- 8- Abaixar o carro e tirar o macaco
- 9- Apertar os parafusos

Quando desenvolvemos um código, devemos fragmentar o problema em várias partes e cuidar de cada uma delas individualmente. Obviamente no problema do pneu furado podemos ter uma situação como esta:

- 3- Pegar o macaco, estepe e chave de roda
Se não houver
Fim
Senão
- 4- Usar a chave para tirar os parafusos do pneu furado

Impomos uma condição para que a troca do pneu continue. Se não tivermos ao menos um dos três itens do passo 3, não conseguiremos trocar o pneu, mas se tivermos a troca continua. Isso se chama condição e é representada por "SE".

Quando se desenvolve um programa usa-se o que chamamos de variáveis, que funcionam como "containers", armazenando valores dentro delas. As variáveis podem ser dos tipos:

- Inteiro (-1,0,1,2,...)
- Real (-54.6666, 0.000, 6.5,...)
- Caracteres ('o', '\$', "blábláblá",...)
- Lógica (Verdadeiro, Falso)

Um exemplo de utilização de variáveis dentro de um algoritmo é:

```
Inteiro: N;  
N←10;
```

O que fizemos foi criar uma variável chamada 'N', do tipo inteiro e atribuir a ela o valor 10. Os ';' no final de cada linha são uma convenção das linguagens de programação para dizer que uma instrução terminou, assim temos duas instruções nesse código: "Inteiro: N" e "N←10".

Vamos a mais um exemplo com variáveis:

```
Inteiro: A, B;  
Escreva("Digite um número: ");  
Leia(A);  
Escreva("Digite mais um número: ");  
Leia(B);  
Se((A+B)=10)  
Escreva("A soma é igual a 10");  
Senão  
Escreva("A soma é diferente de 10");
```

Nesse exemplo temos duas variáveis inteiras (A e B). O comando "escreva" serve para mostrarmos uma mensagem na tela para o usuário. O comando "leia" serve para guardar o que o usuário digitar dentro de uma

variável. Temos ainda a condição "Se ((A+B)=10)", que diz que se a soma de A e B for igual a 10, o programa deve executar o comando dentro do bloco "Se", se for diferente, deve ignorar esse bloco. A soma de A e B está entre parênteses por convenção, funciona exatamente igual às regras de prioridade na Matemática, o que está entre parênteses deve ser executado primeiro. Como queremos comparar dois valores (o resultado da soma e 10), primeiro temos que saber qual é o resultado da soma, para depois o comparar com um valor. Cada "Senão" funciona para apenas um "Se", o que está logo antes dele.

Agora vamos aprender a usar estruturas de repetição, que servem para repetir comandos ou blocos de comandos. As principais estruturas são:

- Enquanto/faça
- Repita/até
- Para/passo

Vamos dar um exemplo simples, vamos supor que quiséssemos ler uma variável e ir somando esse valor à outra variável até que a soma da segunda variável seja maior ou igual a 100. Com o "Enquanto/faça" ficaria assim:

```
Inteiro: valor, soma;  
soma ← 0;  
  
Enquanto(soma < 100) faça  
Início  
    Escreva("SOMA: ", soma);  
    Escreva("Digite um valor para somar: ");  
    Leia(valor);  
    soma ← soma + valor;  
Fim  
Escreva("Sua soma atingiu o limite, parou em ", soma);
```

Isso resolve nosso problema, temos duas variáveis (valor e soma), sendo que antes de começar o processo de soma, damos o valor 0 à variável soma. Depois impomos uma condição para repetição, que diz que enquanto a variável soma for menor ou igual a 100, o bloco deve ser repetido. Depois, mostramos o valor da variável soma na tela e perguntamos o próximo valor para somar, feito isso adicionamos esse valor à variável soma. Isso conclui o bloco, que será repetido tantas vezes quando necessário. O bloco está contido entre o "Início" e o "Fim". Depois que as somas forem feitas e a condição for satisfeita, é executada a próxima linha do código, dizendo que não é mais possível somar e mostrando o valor final da soma.

O mesmo problema pode ser resolvido com "Repita/até", desta forma:

```
...  
Repita  
Início  
    Escreva("SOMA: ", soma);  
    Escreva("Digite um valor para somar: ");
```

```
Leia(valor);  
soma←soma+valor;  
Fim  
Até que (soma>=100)  
...
```

A lógica é a mesma, mas dessa vez dissemos que o bloco tem que ser repetido até que a variável soma seja maior ou igual a 100.

Para a estrutura “Para/ passo”, modificar um pouco nosso programa: Queremos que o máximo de somas seja igual a 10, para isso fazemos assim:

```
Inteiro: valor, soma, contador;  
soma←0;  
  
Para contador de 1 até 10 faça  
Início  
    Escreva("SOMA: ", soma);  
    Escreva("Digite um valor para somar: ");  
    Leia(valor);  
    soma←soma+valor;  
Fim  
Escreva("Sua soma atingiu o limite, parou em ", soma);
```

A estrutura “Para/ passo” define um raio de repetição. O “passo” pode não ser utilizado, quando queremos que a variável contador seja somada com 1 a cada repetição, caso contrário, teremos que usar o “passo”, como no exemplo:

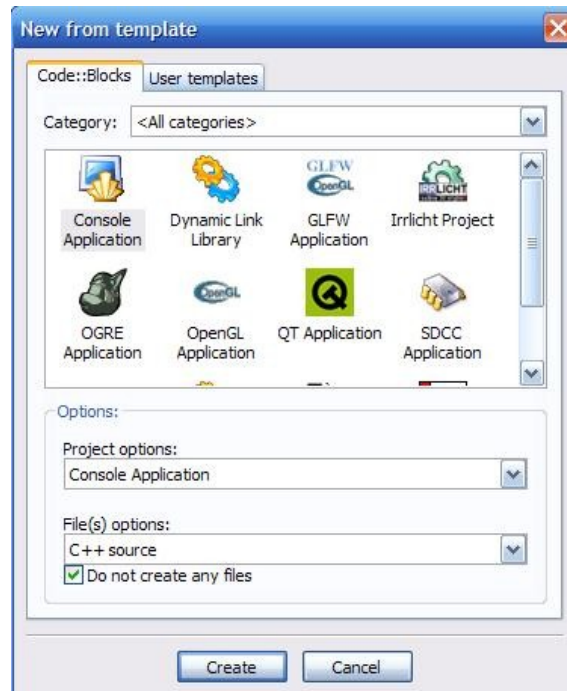
```
Para contador de 1 até 10 passo 5 faça  
Início  
    <comandos>  
Fim
```

No exemplo acima, adicionaremos 2 ao valor da variável contador a cada repetição que for feita. Sendo assim, o código vai se repetir 5 vezes.

CAPÍTULO II – Programação prática

Vamos agora aplicar o conhecimento com algoritmos com linguagem C, para isso devemos ter instalado o Code::Blocks com MingW.

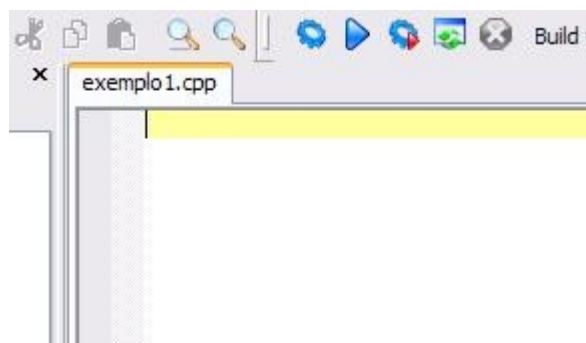
Abra o Code::Blocks e crie um novo projeto (File – New Project), escolha a opção “Console Application” e marque “Do not create any files”. Agora clique em OK para salvar o projeto. Crie uma nova pasta e coloque seu projeto dentro dela para facilitar.



Depois de salvar seu projeto, vamos adicionar um novo código a ele, clicando no menu File- New File e usando um nome com a extensão “.cpp”, e salvando o arquivo na mesma pasta do projeto. Depois clicamos em “Sim” quando formos perguntados sobre adicionar o novo arquivo ao projeto.



Agora estamos prontos para programar, nossa IDE deve ter aberto o arquivo “exemplo1.cpp” para que nós o editemos.



Agora estamos prontos para começar. Vamos começar falando um pouco sobre como a linguagem C funciona.

A linguagem C depende de bibliotecas, que contém as funções (os comandos nos algoritmos). Sem incluir as bibliotecas, não temos como programar nada, pois não vai funcionar.

Então vamos adicionar duas dessas bibliotecas. A sintaxe para se fazer isso é “#include <>”, vamos incluir as seguintes bibliotecas:

```
#include <stdio.h>
#include <stdlib.h>
```

Agora vamos criar o nosso bloco de código principal (chamado main), desse jeito:

```
main() {
}
```

Os símbolos '{' e '}' funcionam como o "Início" e o "Fim" nos algoritmos.

É dentro do bloco `main` que o compilador vai buscar primeiro as funções a executar. E é nele que iremos trabalhar.

Vamos adicionar os seguintes comandos no bloco `main`:

```
printf("Estou programando em C!");
getchar();
```

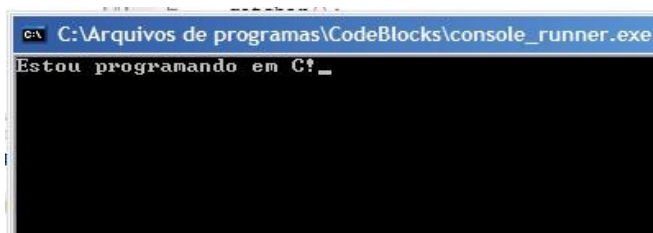
A função `printf` funciona como o "Escreva" nos algoritmos. O problema é que o compilador fecha o programa quando não existem mais comandos para serem executados, para isso colocamos a função `getchar`, que serve para ler um caractere do teclado, ela está sem parâmetros, o que significa que não fará nada com o caractere digitado. Essa função faz com que o programa fique esperando por uma resposta do teclado para continuar a execução. Isso nos dá tempo de ler a mensagem que digitamos.

Ao final disso tudo, nosso código deve estar assim:

```
#include <stdio.h>
#include <stdlib.h>

main() {
    printf("Estou programando em C!");
    getchar();
}
```

Salve seu trabalho com CTRL+S e vamos executar nosso projeto teclando 'F9'. Temos algo assim:



Agora vamos tentar resolver o problema da soma que tínhamos falado no capítulo I, com “Enquanto/faça”. O código, com alguns incrementos, fica assim:

```
#include <stdio.h>
#include <stdlib.h>

main(){
    int valor, soma, contador;

    soma = 0;
    contador = 0;

    while(soma<100){
        system("cls");

        printf("SOMA: %d\n\n", soma);

        puts("Digite um valor para somar: ");
        scanf("%d", &valor);

        soma=soma+valor;
        contador++;
    }

    printf("Sua soma atingiu o limite e parou em %d", soma);
    printf("\nForam necessarias %d somas para atingir o limite", contador);
    getch();
}
```

A função `system` serve para que o nosso programa execute um comando do MS-DOS. O comando `cls` serve para limpar a tela. Na função `printf` temos algumas coisas estranhas:

`%`: mostra na tela uma variável (`%d` significa variável inteira), a variável é colocada fora da mensagem e antecedida por uma `' , '`.

`\n`: serve para pular uma linha. Como colocamos dois, serão puladas duas linhas.

A função `puts` tem a mesma função do `printf`, porém mostra apenas mensagens fixas, e não valores de variáveis. Quando a mensagem é exibida, automaticamente uma linha é pulada.

A função `scanf` tem a mesma função do “Leia” nos algoritmos. Em C, devemos dizer o tipo de variável que vamos ler dentro de uma mensagem (“”), depois colocamos a variável com um `'&'` no início, antecedido por uma `' , '`.

`contador++`: Adiciona 1 ao valor da variável `contador`. É o mesmo que: `contador=contador+1;`

Vamos agora a um exemplo com vetores:

```

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

main(){
    int vetor[10], i;

    for(i=0;i<10;i++){
        system("cls");

        printf("Digite o valor do %d termo do vetor: ", i+1);
        scanf("%d", &vetor[i]);
    }

    for(i=0;i<10;i++)
        printf("%d ", vetor[i]);

    getch();
}

```

A variável vetor tem 10 posições, ou seja, pode armazenar 10 valores diferentes.

A função `for` funciona como a estrutura “Para/passos” em algoritmos. A diferença é que separamos as partes com ‘;’.

No `printf` temos (“...%d...”, `i+1`). O que isso faz é imprimir o valor da variável `i`, mas somando-se 1. Isso serve apenas para a impressão no vídeo e não altera o valor da variável.

No `scanf` estamos lendo `vetor[i]`. Isso significa que a posição do vetor que estamos gravando é a mesma da variável `i`. Quando se trabalha com vetores, usa-se o índice para dizer qual posição está sendo utilizada. A estrutura “Para/passos” serve para aumentar o valor da variável `i`, fazendo com que acessemos os valores do vetor, um a um.

Quando precisarmos usar apenas um comando, não precisamos criar blocos, como foi o caso do último `for`. Essa regra não vale para a função `main`.

E assim concluímos nossa introdução à programação. Ainda há muitas outras coisas para falar, mas depende do seu interesse em buscar essas informações.

Alguns exemplos de código em C podem ser encontrados em:

www.avt3d.pop.com.br/exemplos_c.rar