Monitoria LpI - Módulo I

Iniciando em Linguagem C:

A linguagem **C** é uma linguagem genérica desenvolvida por programadores para programadores tendo como meta características de flexibilidade e portabilidade, é uma linguagem que nasceu juntamente com o aparecimento da teoria de linguagem estruturada e do computador pessoal. Assim tornou-se rapidamente popular entre os programadores, vitoriosa como ferramenta na programação de qualquer tipo de sistema. Foi utilizada para desenvolver o sistema operacional UNIX (antigamente desenvolvido em Assembly), e para desenvolver novas linguagens, como a linguagem C**++** e Java.

A linguagem C foi desenvolvida a partir da necessidade de escrever programas que utilizassem as potencialidades da linguagem de máquina, mas de uma forma mais simples e portátil do que esta, hoje é extremamente visada em tecnologias portáteis e de limitação de memória, pois a mesma trabalha muito próxima da máquina. C é uma linguagem de propósito geral, sendo adequada à programação estruturada, no entanto é mais utilizada para escrever compiladores, analisadores léxicos, banco de dados, editores de texto. E ainda em sistemas operacionais, planilhas, processadores gráficos, sistemas de transmissão de dados, segurança e para solução de problemas de engenharia e física. Dúvidas ainda de sua potencialidade?

Características da Linguagem C

- Portabilidade entre máquinas e sistemas operacionais;
- Pequeno tamanho da sua definição;
- Subdivisão do código e grande utilização de funções;
- Alguma conversão automática entre tipos de dados;
- Dados compostos de forma estruturada;
- Programas estruturados;
- Total interação com o sistema operacional;
- Compilação separada;
- Utilização fácil e extensa de apontadores para aceder memória, vetores, estruturas e funções;
- Possibilidade de usar construções de alto nível;
- Possibilidade de utilizar operadores de baixo nível;
- Produção de código executável eficiente e de baixo custo de memória;
- Confiabilidade, regularidade, simplicidade;
- C é Case Sensitive

Importante:

A linguagem **C** tem 32 palavras reservadas, ou seja, palavras nativas da linguagem. Que tem função pré-definida, são elas:

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
switch	typedef	union	unsigned
void	volatile	while	struct

Escrevendo o Código Fonte

Escreveremos nosso código fonte no editor de texto que se encontra no próprio compilador (poderia ser em qualquer editor de texto), que faz a tradução da linguagem de programação para a linguagem de mais baixo nível; tendo um compilador instalado pode-se também escrever o código no bloco de notas e salvar o documento com a extensão da linguagem, no nosso caso *.c*, no entanto nesse início recomenda-se usar o compilador para escreve o código devido aos recursos adicionais que ele oferece.

Conhecendo o Compilador

Utilizaremos o compilador Dev C++ que é um ambiente de desenvolvimento de programas em C e C++ com editor, compilador, bibliotecas e debugger [sempre que for salvar um código coloque a extensão *.c* pois ele salva automaticamente como *.cpp* -extensão da linguagem C++]

Atalhos do Dev C++

Ctrl + F9: Compila.
Ctrl + F10: Executa.
F9: Compila, e se não houver nenhum erro, executa o programa.

Evitando Erros

- Termine todas os comandos com **;**
- Salvar o programa antes de compilar e compile antes de executar
- Ocorrendo um erro de compilação, dê um duplo clique sobre a mensagem de erro para destacar o comando errado no programa e verifique a linha anterior, que pode ser a responsável pelo erro

OBS: As palavras reservadas em **C,** aparecem no editor de texto do **Dev C++** em negrito. A dupla barra *'//'* é usada para inserir comentários em uma linha, para inserir comentários em bloco usa-se *'/*'* e *'*/'.*

Exemplo:

```
// Comentário em uma linha

/* Comentário
em bloco */
```

Recomendações ao escrever o código:

- Um comando por linha deixa o código mais legível, e ajuda a identificar um erro apontado pelo compilador, que grifa a linha do editor de texto onde possivelmente se encontra o erro.

- Usar código identado, ou seja, utilizar paragrafação hierárquica, torna-o mais legível, ajuda a identificar que comandos estão internos a um procedimento Veja o exemplo dos primeiros programas, olhando o código, intuitivamente, percebemos que os comandos são internos ao "main".

Desenvolvendo em C:

Analisemos os programas abaixo:

1º

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

main()
	{
		printf("Nosso primeiro programa em C");
		system("pause");
	}
```

2º

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

main()
	{
		int x;
		int y;
		int total;
		x=3;
		y=5;
		total=x+y;
		printf("Soma = %d",total);
		system("pause");
	}
```

Vamos conhecer o significado de algumas linhas comuns aos dois programas que definem a estrutura de um programa em **C**:

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

Como dito anteriormente C possui apenas 32 comandos nativos, programar apenas com eles seria muito limitado. Ao longo dos anos os profissionais da área, a partir desses comandos nativos foram criando funções para determinados fins, muitas dessas funções são úteis até hoje, elas se encontram nas chamadas "Bibliotecas". Adicionando uma biblioteca a um programa você tem acesso a todas as funções nela definidas, elas se caracterizam pela extensão *.h* e para "chamá-las" utilizamos o comando #include <nomedabiblioteca.h>. Esse procedimento deve ser feito logo no início do código.

O "include" <stdio.h> tem como finalidade incluir as definições da biblioteca de C de entrada e saída: STandarD Input Output. Dessas funções, as mais usadas são o scanf e o printf.

O #include <stdlib.h> tem como finalidade incluir as definições da biblioteca padrão de C: STandarD LIBrary. Dessas funções, estaremos sempre usando a função system. Essa função permite, por exemplo, a execução de comandos do DOS, entre eles pause, date, time.

```
main ()
	{
	}
```

Essa função indica a partir de onde o programa começará a ser executado, usamos os delimitadores {} para indicar o escopo do programa, ou seja, o espaço onde as variáveis, funções, procedimentos vão atuar.

```
; (ponto e vírgula)
```

O *'; '* é o separador de comandos, ao final de cada comando devemos colocá-lo. Poderíamos escrever o programa todo em uma só linha, pois, o compilador utiliza o *'; '* para saber onde termina um comando.

```
system("pause");
```

Utilizamos esse comando, especialmente quando trabalhamos no Dev-C++ que serve para pausar o resultado final de uma execução para que o

programa não feche automaticamente, e assim, possibilitar a visualização dos resultados, para alguns compiladores de C não é necessário usar esse comando, mas no Dev-C++ vamos usá-lo.

Tipos Básicos de Dados:

Um tipo de dado caracteriza-se pelo espaço que ocupará na memória quando criado pelo programa e também pelas operações a ele atribuídas.

Tipo	Tamanho em Bytes	Amplitude de Valores
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis digitos de precisão
double	8	Dez digitos de precisão
long double	10	Dez digitos de precisão

Variáveis em C:

Para declarar uma variável em **C** precisamos informar o tipo da variável e que nome daremos a ela, para referenciarmos aquele espaço na memória que ela reservará. Veja as linhas 6,7,8 do segundo programa:

```
int x;
int y;
int total;
```

OBS:
Existem algumas restrições quanto ao nome que se dará a variável:
-Não podem começar com números
-Não podem ter caracteres especiais
-Não podem conter espaços entre os caracteres

Ex: Nomes válidos: alfa, x, X1, Teste_nome
Nomes inválidos: 5x, e(2), A+B, Teste nome

Outra nota importante - **C** é case sensitive, interpreta letra maiúscula e minúscula como diferentes para um mesmo caractere.
Ex: int NUM;
int num;

NUM e num, não são a mesma variável, ocupam espaços distintos na memória.

Procure sempre dar nomes as variáveis que tenham relação com a função por ela desempenhada no programa, para evitar "confusões" e facilitar o trabalho, principalmente quando se escreve um código grande.

Atribuindo valores a variáveis:

Para atribuir valores usamos o operador "=", veja as linhas 9,10,11 do segundo programa:

```
x=3;
y=5;
total=x+y;
```

Estamos atribuindo a x o valor 3 e a y o valor 5. O comando de atribuição sempre trabalha nessa ordem : lê-se da direita pra esquerda, o três é atribuído a x.

Devemos atribuir apenas valores compatíveis com o tipo de variável - valor inteiro para variável inteira se atribuíssemos um float (número real) a uma variável inteira só iria a parte inteira - int x=3.6; - quando for pedido para imprimir o valor de x na tela, apareceria só o valor inteiro, no caso '3'.

Outro ponto em que temos de ter atenção é o de sempre atribuir valores a variáveis já declaradas no programa:

```
z=7;
```

Imagine essa linha no segundo programa (exemplo do início), o código não seria compilado e uma mensagem de erro iria ser mostrada dizendo que z ainda não foi declarada.

Operações

C tem seus operadores matemáticos básicos:

Soma: +

Subtração: -

Multiplicação: *

Divisão: /

Mod: % (retorna o resto de uma divisão**)**
OBS:

```
int x,y,mod;
x=9;
y=2;
mod=x%y; //se lê modulo de x por y

//Em C o valor final de mod será 1(o resto da divisão de 9 por 2 é 1)
```

Operadores Booleanos:

```
/* Operadores booleanos retornam True ou False, em C 0 ou 1. São
fundamentais para trabalhos com condicionais.
 < menor que
 > maior que
 == igual
 != diferente */

int a,b;
a=2;
b=3;

a<b    //Retorna 0, proposição verdadeira
a>b    //Retorna 1, proposição falsa
a==b  //Retorna 1, proposição falsa
a!=b   //Retorna 0, proposição verdadeira
```

Operações Alternativas:

```
//Em C existem “operações atalhos”, servem para facilitar o desenvolvimento
//do programa, facilitam na leitura do código e são de simples implementação.

int x,y;

x++ // é o mesmo que x = x +1
x--  // é o mesmo que x = x -1
x*=y // é o mesmo que x = x*y
x/=y // é o mesmo que x = x/y
x+=y // é o mesmo que x = x+y
x-=y // é o mesmo que x = x-y
```

Saída e entrada de valores (Funções básicas de I/O)

Saída:

Vamos falar primeiro sobre saída, veja o primeiro programa, ele imprime na tela a frase:

Nosso primeiro programa em C

O comando usado para isso é o printf, como ele não é um comando nativo da linguagem temos que incluir a biblioteca a qual ele pertence (stdio.h). Podemos usar printf para imprimir frases sem variáveis para tal veja o exemplo:

```
printf ("O conteúdo da frase vem entre aspas");
```

Já para imprimir conteúdos de variáveis:

```
printf ("o q irá aparecer na tela",lista_de_argumentos);
```

Vejamos o exemplo do segundo programa:

```
printf("Soma = %d",total);
```

Onde queremos que seja impresso o conteúdo de uma variável coloca-se um código de controle correspondente ao tipo da variável - no exemplo %d usado para inteiros - depois de terminada a sentença que queremos que saia na tela fechamos aspas, e separamos com uma virgula da lista de variáveis, logo onde tem %d será mostrado o valor da variável total. Se fossem 2 ou mais variáveis:

```
printf("A soma de %d com %d eh %d",x,y,total);
```

Colocamos os códigos de controle nas posições desejadas pra aparecerem na tela, e na lista de variáveis colocamos na ordem que queremos que apareçam. Veja a lista de códigos de controle para os tipos de dados e também caracteres especiais

%d	Inteiro
%f	Numero Flutuante
%c	Caractere
%s	String
%%	Coloca na tela um %

\n	mudança de linha
\t	tabulação
\a	aviso sonoro
\\	barra invertida
\?	sinal de interrogação

Entrada:

O comando de entrada de dados em **C** mais conhecido é o scanf. Tal comando interrompe o programa para inserção de dados até a digitação da tecla enter. Os valores lidos são associados às variáveis precedidas de '&'. O caractere '&' indica referência ou endereço da variável. E é através dessa referência ou endereço que o valor da mesma será atualizado após a leitura do correspondente dado.

3º

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

main() {

      int x;
      printf(“Digite um valor inteiro:”);
      scanf(“%d”, &x);
      printf(“Voce digitou %d”, x);
      system(“pause”);

        }
```

No 3º exemplo o procedimento é:

- Alocar espaço na memória para um inteiro chamado ‘x’ (declaração da variável);
- Imprimir na tela *Digite um valor inteiro;*
- Atribui a *x* o valor inteiro digitado no teclado (a atribuição & indica a posição de *x* na memória, ou seja, onde irei “guardar” aquele valor);
- imprime o valor de x, se o usuário teclou 2, o programa imprimirá *Voce digitou 2*

Assim como em printf , temos de colocar o código de controle (ou código de formatação) para indicar com que tipo de variáveis estamos trabalhando, são válidos para scanf os mesmos códigos usados em printf.

Usando scanf para mais de uma variável

Num único comando de scanf podemos pedir para serem captados mais de uma entrada:

scanf("%d %d", &x, &y);

Não separamos os códigos de formatação com vírgula, mas o fazemos com os endereços de variáveis e a ordem que essas referências aparecem será a mesma de atribuição de valor ou seja o primeiro valor digitado (depois de se

apertar “enter”) será atribuído a &x e então digitamos o segundo valor, confirmamos com o “enter” e este será atribuído a &y.

Erros comuns com printf e scanf:

printf(“Alguns erros comuns com essas funções”)

printf(“A soma de %d e %d eh %d , x , y, total);

printf("A soma de %d com %d eh %d",x y total);

scanf(“%d” , x);

scanf(“%d” &x);

scanf("%d, %d", &x, &y);

scanf("%d %d", &x &y)