**Exercícios de Revisão – 1º Bimestre**

***Soluções***

1. Como é estruturado um programa em C? Por onde começa a execução de um programa C?

Um programa C é estruturado por funções, que são implementadas em arquivos fontes. O programa em C deve conter uma função *main*, que é por onde inicia a sua execução.

1. Para que serve a diretiva *#include*?

Serve para a inclusão de uma biblioteca em um programa. Deste modo, funções implementadas na biblioteca incluída podem ser utilizadas no programa.

1. Qual a utilidade de um tipo de dado? Quais são os tipo de dados primitivos na linguagem C?

O tipo de dado define o conjunto de valores que podem ser armazenados em uma variável. Os tipos de dados em C são *char*, *int*, *float* e *double*. A estes tipos pode-se aplicar *modificadores*, que são *signed* (com sinal), *unsigned* (sem sinal), *short* (curto) e *long* (longo).

1. Como é a instrução padrão de saida em C? Quais os principais controles de impressão? E os principais formatos de valores?

A instrução de saída padrão em C é a função *printf* (redução de *print formatted*) da biblioteca *stdio*. O primeiro parâmetro da função printf especifica o formato da saída, os demais parâmetros, opcionais, são as variáveis que se deseja imprimir.

Os principais controles de impressão são:

|  |  |  |  |
| --- | --- | --- | --- |
| **Código** | **Significado** | **Código** | **Significado** |
| \n | Nova linha | \f | Salta uma página |
| \t | Tabulação horizontal | \v | Tablulação vertical |
| \b | Retrocesso (backspace) | \r | Retorna ao inicio da linha |
| \a | Emite beep | \0 | Caracter nulo (null) |
| \\ | Barra invertida | \x*nn* | Código ASCII *(nn* hexadec.) |
| \’ | Aspa simples | *\’”* | Aspas duplas |

Os principais formatos de saída são:

|  |  |  |  |
| --- | --- | --- | --- |
| **Código** | **Significado** | **Código** | **Significado** |
| %d | Inteiro | %u | Decimal sem sinal |
| %c | Caractere | %s | String |
| %f | Float | %lf | Reais longos |
| %x | Hexadecimal | %o | Octal |
| %e | Notação científica | *%%* | Porcentagem |

1. Qual é a instrução padrão de entrada de dados em C? Explique sua sintaxe e semântica.

É a função *scanf*, também da biblioteca *stdio*.

Sua sintaxe é: *scanf(“expressão de controle”, lista de argumentos);*

<expressão de controle> especifica o formato dos valores de entrada enquanto que <lista de argumentos> especifica os endereços das variáveis onde os valores de entrada serão armazenados.

1. Qual é a instrução mais adequada para obter a entrada de uma string em C? Por que?

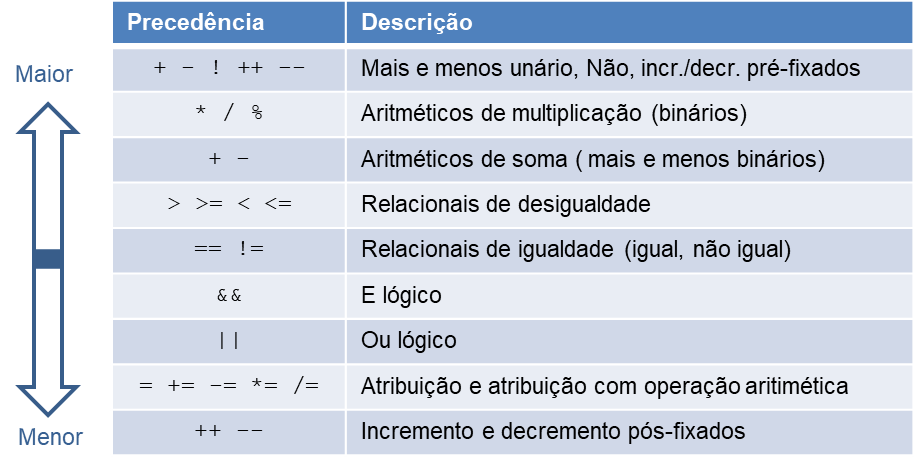
É a função *gets()*. O formato *%s* utilizado no comando *scanf* interrompe a leitura ao encontrar o primeiro caractere branco ou outros separadores. Deste modo, pode truncar a string de entrada. Para evitar isso, deve-se utilizar a função *gets()*, da biblioteca *stdio*, que lê e retorna toda a string presente no buffer de teclado, evitando perda de valor digitado.

1. Quais são os operadores relacionais em C? Qual operador nega qual?

==, !=, >, >=, <, <=;

== nega !=; > nega <= e < nega >=

1. Qual é a precedência de operadores em C?



1. Considere o trecho de código abaixo:

**if**( control > 5 ) **if**( control > 10 ) printf(“Maria”);

**else**

printf(“Pedro”);

**else if**( control < 0 )

printf(“Joana”); **else** printf(“José”);

O que será impresso se o conteúdo da variável *control* for:

1. 10 Pedro
2. -5 Joana
3. 3 José
4. 0 José
5. 17 Maria
6. Qual é a sintaxe e semântica do comando Switch em C?

Sintaxe: switch (<expressão>){

case const1: comandos1; [break];

case const2: comandos2; [break];

. . .

case constn: comandosn; [break];

[<default>]: comandosDefault;

}

Semântica: *expressão* (uma variável do tipo **int** ou **char**) é avaliada e executa o comando ou comandos cujo *case* corresponde ao valor avaliado. As constantes nas cláusula *case* devem ser do mesmo tipo de expressão. A cláusula *default* é opcional e se presente, executa o(s) comando(s) default no caso em que nenhuma constante das cláusulas *case* coincidir com o valor da expressão.

1. O comanda *break* é obrigatório no comando Switch? Justifique sua resposta.

Não. O comando *break* é opcional e faz com que o *switch* seja interrompido após execução do(s) comando(s) da respectiva cláusula *case.*

1. Quais são as estruturas de repetição em C? Explique a sintaxe e semântica de cada. Qual é a melhor delas para programar iterações em programas C?

Comando For: **for( inicialização; teste; incremento ) {**

**comandos;**

**}**

A variável de controle do *for* é iniciada, depois o *teste* é avaliado e sendo **verdadeiro**, executa o(s) *comando(s)*. Na sequência o *incremento* é feito e o *teste* novamente executado. E assim sucessivamente, até que *teste* seja avaliado **falso**, quando encerra o laço.

Comando While: **while( condição ) {**

**comandos;**

**}**

*Condição* é avaliada e sendo **verdadeira**, executa o(s) *comando(s)*. Na sequência *condição* é novamente avaliada. E assim sucessivamente, até que *condição* seja avaliada **falso**, quando encerra o laço.

Comando Do While: do {

comandos;

} while ( condição );

*Comando(s)* são executados e ao final *condição* é avaliada e sendo **verdadeira**, retorna para executar o(s) *comando(s)* novamente. E assim sucessivamente, até que *condição* seja avaliada **falso**, quando encerra o laço.

Não há estrutura melhor ou pior para programar iterações. A estrutura mais simples de ser utilizada é o comando *for*, porém este só pode ser utilizado em situações em que a quantidade de iterações é conhecida no momento de codificação. No mais, o uso das estruturas de repetição fica ao critério e estilo de cada programador.

1. Elabore um programa em C que leia dois números inteiros positivos e calcule o MDC – máximo divisor comum entre ambos, pelo método das subtrações sucessivas. Por este método, subtraia o o maior do menor sucessivamente, até que ambos se tornem iguais, que é o MDC. Por exemplo, MDC(10, 35) = 35 – 10 = 25; 25 – 10 = 15; 15 – 10 = 5; 10 – 5 = 5 » 5 = 5, logo MDC(10, 35) = 5.

#include <stdio.h>

int main(){

int num1, num2, A, B;

printf("Informe dois números inteiros positivos: ");

scanf("%d", &num1); A = num1;

scanf("%d", &num2); B = num2;

while(A != B){

if (A > B)

A -= B;

else

B -= A;

}

printf("\n\tMDC(%d, %d) = %d.\n\n", num1, num2, A);

}

1. Elabore um programa em C que leia um número inteiro positivo exatamente de 4 dígitos (entre 1000 e 9999) e um procedimento que apresente todos os números primos do intervalo entre 1 e o número lido. Caso o número informado não atenda à regra, mostre a mensagem “Numero incorreto”. Nota: um número é primo quando é divisível apenas por ele mesmo e por 1.

#include <stdio.h>

int main(){

int num, i, j, ehPrimo;

do{

printf("\n\tInforme um numero inteiro: ");

scanf("%d", &num);

if (num < 1000 || num >= 10000)

printf("\n\tNumero incorreto.");

} while(num < 1000 || num >= 10000);

printf("\n\tNumeros primos entre 1 e %d: \n", num);

for (i = 1; i <= num; i++){

ehPrimo = 1;

for (j = 2; j < i; j++){

if(i%j == 0){

ehPrimo = 0;

break;

}

}

if(ehPrimo)

printf("\n\t %d.", i);

}

}

1. Considerando o cursor posicionado inicialmente no canto superior esquerdo da tela, informe o que cada trecho de código vai imprimir na tela:
   1. int i, j;

for(i = 1; i <=6; i++){

for(j = 1; j <= i; j++)

printf("#");

printf("\n");

}

**#**

**##**

**###**

**####**

**#####**

**######**

* 1. int i, j;

for(i = 6; i >=1; i--){

for(j = 1; j <= i; j++)

printf("\*");

printf("\n");

}

**\*\*\*\*\*\***

**\*\*\*\*\***

**\*\*\*\***

**\*\*\***

**\*\***

**\***

* 1. int i, j;

for(i = 1; i <=6; i++){

for(j = 1; j <= i; j++)

printf("o");

printf("\n");

}

for(i = 5; i >=1; i--){

for(j = 1; j <= i; j++)

printf("o");

printf("\n");

}

**o**

**oo**

**ooo**

**oooo**

**ooooo**

**oooooo**

**ooooo**

**oooo**

**ooo**

**oo**

**o**