Olá Pessoal

Imagino que ficaram várias dúvidas na nossa primeira aula sobre E/S (Entrada e Saída) em C.

Aqui pretendo dar uma breve introdução sobre o assunto e deixar algumas dicas que, espero, facilitarão a resolução do exercícios no Run.codes.

Como se lê as 'coisas' em C?

Há várias maneiras de se ler 'valores' que precisaremos nos nosso programas. A função mais usada é o **scanf()**. Link para o manual: https://en.cppreference.com/w/c/io/fscanf

No início este manual vai parecer bastante incompreensível, mas qual a ideia do scanf()? A ideia principal é que ela lê tudo que vem da entrada padrão (teclado). Imagine que no run.codes tem alguém digitando tudo por um teclado quando vc pede para ler alguma coisa com scanf(). E imagine mais: tudo que vai sendo digitado (a entrada) é uma cadeia de caracteres (ou uma string). Vou representar uma string por algo delimitado por " ".
Então, uma entrada típica para um programa em C poderia ser:

"253 Z 4AC"

E imagine ainda que o seu programa define este formato de entrada porque ele necessita de um valor **inteiro** (253), um **caracter** ('Z') e um outro valor inteiro, mas no formato **Hexadecimal** (4AC). *** $AC em decimal é = 1196, certo?

Como, sendo uma string, é possível converter uma string "253 Z 4AC" nesses 3 diferentes variáveis (inteiro, char e inteiro - mas com representação em hexadecimal)?

Este é o papel do scanf(). Essa função do C, **consome** uma sequência de caracteres e vai **interpretando** tais caracteres de acordo com o que vc, programador/a deseja !!!

Veja a sintaxe do scanf() que utilizo para ler a sequência como ilustrada acima:

scanf("%d %c %X", &n, &c, &nHexa); (n e nHexa são variáveis inteiras e c é char)

Veja que o primeiro argumento de scanf() é uma string que representa exatamente sua receita de interpretação ! O caracter '%' é chamado de interpretador e o caracter que vem logo a seguir dita como vc quer interpretar o seu input !

%d >> converte a sequência de caracteres '253' no inteiro 253 e armazena na variável n !!!
%c >> converte a sequência de caracteres "Z" no caracter 'Z' e armazena em c
%X (ou %x) converte a sequência de caracteres "4AC" (em hexadecimal) em inteiro e armazena na variável inteira nHexa.

IMPORTANTE: a ordem importa e muito! se vc tem 3 interpretadores, significa que as variáveis n, c e nHexa tem que ser colocadas nesta sequência.

IMPORTANTE 2: veja que as variáveis n, c e nHexa sâo precedidas do símbolo '&' na chamada da função scanf(). Por ora, peço que apenas aceitem que, sem este símbolo, seu programa não compila. Explicaremos isso mais adiante, quando falarmos de ponteiros.

Para verificar se leu direitinho, use o **printf()**. Manual da função >> https://en.cppreference.com/w/c/io/fprintf

Repare que printf() tem semelhanças com scanf(). Ele vai converter (**interpretar**) as suas variáveis em uma sequência de saída (um **stream de caracteres** !!) para ser 'jodado' na saída padrão, que é o monitor de vídeo !

veja a sintaxe:

printf("%d * %c *** %d\n", n,c,nHexa);**

Gera uma string (cadeia de caracteres) segundo o formato definido no primeiro argumento **"%d *** %c *** %d\n"**, cria uma string e põe no monitor !

NOTA: veja que no final da string tem um caracter '\n' que em C, é a quebra de linha. Porque eu quis que se pulasse uma linha ao final !

a saída do seu programa seria:
253 *** Z *** 1196

MUITO IMPORTANTE: Se a minha entrada fosse:

" 253 Z 4AC"

Como funcionaria o meu scanf() acima?

RESPOSTA: exatamente da mesma forma !!! Leia o manual. Lá diz que um interpretador (conversor) do tipo '%d' , **antes de converter**, pula todos os whitespaces que existem !!

O que são whitespaces ? R. >>> são os caracteres **blank space (' '), newline ('\n'), horizontal tab ('\t'), carriage return ('\r'), form feed ('\f') or vertical tab ('\v')**.

Portanto, a sequência de espaços em branco que aparecem à esquerda de 253 serão consumidas e descartadas, não afetando em nada a leitura das variáveis.

ATENÇÃO, ATENÇÃO: (não diga que não avisamos!).

O conversor '%c' **NÃO** consome whitespaces a não ser que seja precedido na sua string de conversão por um espaço !! Confuso? Explicarei….

seja a entrada: "210 y" e vc quer ler um inteiro e um caracter.

versão 1 do scanf(): **scanf ("%d %c", &n, &c)**;
OK !!!! veja que entre %d e %c TEM um espaço em branco na string de conversão. Então, o valor y será lido corretamente na variável c.

Agora supondo que vc criou um código com 2 scanf - nada impede que vc prefira consumir a entrada com várias chamadas à função scanf() -, da seguinte maneira:

scanf ("%d", &n);
scanf ("%c", &c);

Ao imprimir o valor das variáveis com printf("%d *** %c\n"), o resultado seria:

210 32

O QUE??? 32 ?? de onde veio isso???? o valor inteiro 32, na tabela ascci corresponde ao espaço em branco ' ' .

Como o seu segundo scanf é **scanf ("%c", &c)** [não tem um espaço em branco antes de %c, logo NÃO consumirá whitespaces] o caracter válido a ser armazenado na variável c será o espaço em branco entre 210 e y !!!!

Se vc quiser manter o formato de 2 scanf() no seu código, terá que fazer:

scanf ("%d", &n);
scanf (" %c", &c);

observe o espaço em branco antes do %c, na string de conversão.

Os principais conversores que vc precisará saber para os exercícios:

exercício 1: Bom dia alguém… Pede para ler um nome (string = vetor de char = char nome[50], por exemplo)

conversor:

- %s

exercício 2: ler uma data, cujo formato é: vftuwkkoxdbrzhrfzy5EF0416

conversores:

- %*[a-z]: o símbolo ‘*’ depois do ‘%’ indica que simplesmente deve-se ignorar a sequencia de letras de ‘a’ a ‘z’ [a-z] da entrada. Entao, isso significa que, embora sua string de conversão possua 4 conversores, vc precisa passar somente 3 variáveis para serem lidas…

- %3X (%X interpreta a sequência como um nro hexadecimal (já vimos isso acima!), MAS neste caso, temos %3X, que significa que devo consumir apenas 3 caracteres da sequência (“5EF”)

- %2d : interpreta 2 caracteres como inteiro
- %o: interpreta os demais caracteres como octal

exercício 3:

- %d: para ler o inteiro..

exercício 4: interpretadores já explicados anteriormente..

- %d
- %c

Um pouco mais de detalhes sobre o **printf()**.

Repare que os exercícios pedem que vc produza uma saída com algumas particularidades, quer seja na apresentação dos valores (número de casas decimais, notação científica, etc) ou mesmo de formatação (recuo, quebra de linha, etc…)

Vamos exercício por exercío.

exercício 1: pede apenas que vc, após imprimir a string (nome), coloque uma exclamação e quebre a linha….

- %s: para converter seu vetor de char em string
- ‘\n’ no final da string de conversão, para pular linha.

exercício 2: todas suas variáveis são inteiras. O programa nao requer nada de muito especial, a nao ser separar dia, mes e ano com o caracter ‘/’ e uma quebra de linha:

- %d
- ‘\n’

exercício 3: essa é mais ‘chatinho’ porque requer um avanço para imprimir o valor, a raiz e potência, além de um formato específico (casas decimais, notação científica)

a) casas decimais: seja uma variável real double r = 3.45678912345. **printf("%lf\n", d)** produzirá o seguinte resultado: 3.456789
Está claro aqui que o conversor para double é '%lf'.
Por padrão, toda saída de um float ou double em c, é dada com 6 casas decimais de precisão. Se vc quiser regular o nro de casas decimais deve fazer: **printf("%.2lf\n", d)**. Agora a saída será: 3.46. Note que o '.2' entre % e lf dita quantas casas decimais vc quer.

finalmente, se quiser 'regular' o nro de valores da parte inteira do double (se nao houver, ele provoca um 'empurrão' para a direita) vc deve fazer: **printf("%10.2lf\n", d).** Neste caso, seu número será deslocado para a direita x espaços, considerando tab o nro de caracteres do nro produzido. Neste caso o nro produzido por **printf("%.2lf\n", d)** será 3.46 (4 caracteres, certo!). Assim sendo, **printf("%10.2lf\n", d)** colocará 6 espaços em branco à esquerda e imprimirá o nro de forma que o último caracter esteja alinhado 10, mais à direita.
a sequencia para os 3 printfs() acima seria:

3.456789
3.46
        3.46

b) se a variável for do tipo float, o conversor será '%f'. tudo que foi dito acima, serve para float tb.
c) notação científica:
    - %e

E tudo que foi dito acima para o nro de casas decimais, parte inteira e recuos serve para esta notação… Então, se substituir 'lf' por 'e' e executar os 3 printf() acima teremos como saída:

3.456789e+00
3.46e+00
 3.46e+00

exercício 4: Acho que não requer nenhuma observação adicional sobre como criar o printf.