Aula prática #2 – Variáveis e tipos básicos de dados

Problema 1

Com este exercício pretende-se criar uma familiarização com o debugger **GDB**. Para tal iremos usar um programa cujo objetivo seria calcular a formula resolvente, que tem a seguinte formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \tag{1}$$

No entanto, devido à existência de bugs, o valor não é corretamente calculado.

1.1 – Compile e corra o seguinte programa. Este programa imprime ":)" caso os valores da formula resolvente tenham sido bem calculados e ":(" caso contrário. Conforme esperado, o programa deverá imprimir ":(".

```c
#include <stdio.h>
#include <math.h>

float f_res_sqrt(float a, float b, float c) {
    int tmp = -4 * a * c;
    tmp = b - tmp;
    tmp = sqrt(tmp);
    return tmp;
}

float f_res_pos(float a, float b, float c) {
    return (-b + f_res_sqrt(a, b, c)) / 2 * a;
}

float f_res_neg(float a, float b, float c) {
    return (-b - f_res_sqrt(a, b, c)) / 2 * a;
}

int main() {
    float PRECISION=0.00001, A=30.4, B=-123, C=75, R_NEG=0.748063f, R_POS=3.29799;
    float v_pos = f_res_pos(A, B, C);
    float v_neg = f_res_neg(A, B, C);

    puts(((fabs(v_pos - R_POS) < PRECISION) &&
          (fabs(v_neg - R_NEG) < PRECISION))?":)": ":(");
    return 0;
}
```

Nota: Sempre que utilizar funções da biblioteca "math.h" deve incluir a flag "-lm" nos parâmetros de compilação.

Nota: Sempre que pretender usar o **GDB** para fazer debug de um programa, deve **sempre** incluir a flag "-g" nos parâmetros de compilação. Esta flag faz com que o programa inclua informação de debugging permitindo assim associar instruções máquina a linhas de código.

1.2 – Execute novamente o programa mas agora usando o **GDB**. Para tal execute a instrução que se segue:

```
1 $ gdb ./a.out
2 (gdb)
```

Neste momento o **GDB** encontra-se pronto e à espera de instruções de debugging para executar. Insira a palavra "run" ou simplesmente "r" para correr o programa dentro do **GDB**. Apesar de, desta vez, aparecerem mensagens extra provenientes do próprio debugger, podemos ver que o programa executou pela presença do output ":(". Para poder fazer debug de um programa, convém saber um conjunto de comandos básicos:

"break <localização>" ou "b <localização>" – adiciona um breakpoint na localização especificada. Sempre que o programa passe num breakpoint, a sua execução é interrompida, entrando em modo interativo e permitindo assim executar comandos.

"continue" ou "c" – continua a execução do programa de forma não interativa.

"next" ou "n" – avança o programa para a próxima instrução. No caso de uma função ser chamada nessa instrução, o debugger executa a função de forma não interativa, voltando ao modo interativo no final da execução da função.

"step" ou "s" – avança o programa para a próxima instrução. No caso de uma função ser chamada nessa instrução, o debugger entra na função em modo interativo.

"print <expressão>" ou "p <expressão>" – imprime o valor de uma expressão. De um modo geral, a expressão é simplesmente o nome de uma variável.

Para mais informações, consulte as seguintes páginas:

- http://gdb-tutorial.net/
- https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf

1.3 – O exemplo seguinte mostra uma execução do **GDB** em que foram seguidos os seguintes passos:

1. Criou-se um breakpoint na função "main" (linha 2).
2. Deu-se ordem para executar o programa (linha 4).
3. O programa atingiu um breakpoint, entrando assim de novo em modo interativo (linha 7).
4. Deu-se ordem para avançar para a próxima sem entrar nas funções "f_res_pos" e "f_res_neg" (linhas 9, 11 e 13).
5. Imprimiu-se o valor das variáveis "v_pos" e "v_neg" (linhas 15 e 17).

Tente agora repetir o processo mas desta vez executando o interior das funções de forma interativa.

Exemplo

```
$ gdb ./a.out
(gdb) break main
Breakpoint 1 at 0x400594: file main.c, line 22.
(gdb) r
Starting program: ./a.out

Breakpoint 1, main () at p1.c:22
22      float PRECISION=0.00001, A=30, B=-123, C=75, R_NEG=0.7452f, R_POS=3.3548f;
(gdb) n
23      float v_pos = f_res_pos(A, B, C);
(gdb) n
24      float v_neg = f_res_neg(A, B, C);
(gdb) n
26      puts(((fabs(v_pos - R_POS) < PRECISION) &&
(gdb) print v_pos
$1 = 3255
(gdb) print v_neg
$2 = 3255
```

1.4 – Use os comandos apresentados na pergunta 1.2 para descobrir a localização do(s) bug(s).

Sugestão: Tente resolver a equação no papel para os valores codificados no programa de forma a poder perceber onde é que o programa calcula um valor diferente do esperado.

1.5 – Corrija o bug. Depois de o bug ficar devidamente corrigido, o programa deverá imprimir ":)".

Problema 2

Escreva um programa que determine o perímetro e área de uma circunferência cujo raio é especificado pelo utilizador.

Nota: Considere $\pi = 3.1416$

Exemplo

```
Raio? 6
Perimetro = 37.6992

Area = 113.0976
```

Problema 3

Escreva um programa que leia a temperatura em graus Celsius e apresente a temperatura equivalente em Kelvin e em graus Fahrenheit. Os valores devem ser apresentados com 2 casas decimais.

> **Nota**: $k = c + 273.15$ e $f = c * \frac{9}{5} + 32$, sendo k, c e f as temperatura em graus Kelvin, Celsius e Fahrenheit, respetivamente.

Exemplo

```
Qual a temperatura? 23.7
23.7 C = 296.85 K
23.7 C = 74.66 F
```

Problema 4

Escreva um programa que leia dois números inteiros e indique se o primeiro é múltiplo do segundo.

Exemplo

```
Numero inteiro 1? 336
Numero inteiro 2? 7
336 e multiplo de 7
```

Exemplo

```
Numero inteiro 1? 210
Numero inteiro 2? 9
210 nao e multiplo de 9
```

Problema 5

Escreva um programa que leia um número decimal e escreva o número com 3 casas decimais, a parte inteira e a parte decimal.

> **Nota**: A inclusão da biblioteca matemática externa obriga a acrescentar "-lm" no final da instrução de compilação do programa.

Exemplo

```
Insira um numero: 3.12146
Numero com 3 casas decimais = 3.121
Parte inteira = 3
Parte decimal = 0.121460
```

Problema 6

Escreva um programa que leia um número, arredonde-o, e escreve os dois números pares mais próximos (usando apenas operações aritméticas).

> **Nota**: Por operações aritméticas entendem-se as operações de adição ($+$), subtração ($-$), multiplicação ($*$), divisão ($/$) e módulo/resto da divisão inteira ($\%$).

Exemplo

```
Insira um numero: 4.5
4 6
Insira um numero: 4.2
2 6
```

Problema 7

Escreva um programa que leia o valor total de segundos e mostre o equivalente em dias, horas, minutos e segundos.

Exemplo

```
Quantos segundos? 105747
105747 segundos correspondem a 1 dia, 5 horas, 22 minutos e 27 segundos
```

Problema 8

Escreva um programa que leia dois números e troque os valores das variáveis usando **apenas duas variáveis** no código.

> **Sugestão**: Considere a utilização de operações aritméticas para fazer a troca dos valores das variáveis.

Exemplo

```
Insira numero para variavel 1: 10
Insira numero para variavel 2: 20
Valor da variavel 1 depois da troca: 20
Valor da variavel 2 depois da troca: 10
```