CURSO DE TECNOLOGIA EM DEFESA CIBERNÉTICA

FIAP

ESPIRITO SANTO – ES

<ABRIL/2021>

VITOR PRANDO GUSMÃO

O trabalho está em busca de mostrar um exemplo simples de como um programa de calculadora pode ser feita utilizando a linguagem C como base.

A calculadora possui 4 operações simples e 1 avançada, sendo as simples:

- Soma
- Subtração
- Multiplicação
- Divisão

E a avançada:
- Exponenciação

Para começar, vou dar a introdução de como a calculadora funciona.

Basicamente você pode escolher entre essas 5 opções supracitadas.
1 - Soma
2 - Subtracao
3 - Multiplicacao
4 - Divisao
5 - Exponenciacao

E apartir daí, o programa pede 2 números para realizar a operação desejada.

Exemplo: Se os inputs forem 4 5 2 o resultado vai ser 5 / 2 em numero inteiro. Neste caso, 2 (2.5)

* Todas as operações são realizadas com números inteiros de 64 bits de tamanha (tipo long).

```
int main(int argc, char argv[]){
banner();

unsigned char r;
r = conf();
unsigned long nums[2] = {0};


if(argc != 3)
{
    printf("Digite o primeiro operando: ");
    scanf("%d", &nums[0]);
    printf("Digite o segundo operando: ");
    scanf("%d", &nums[1]);
}
```

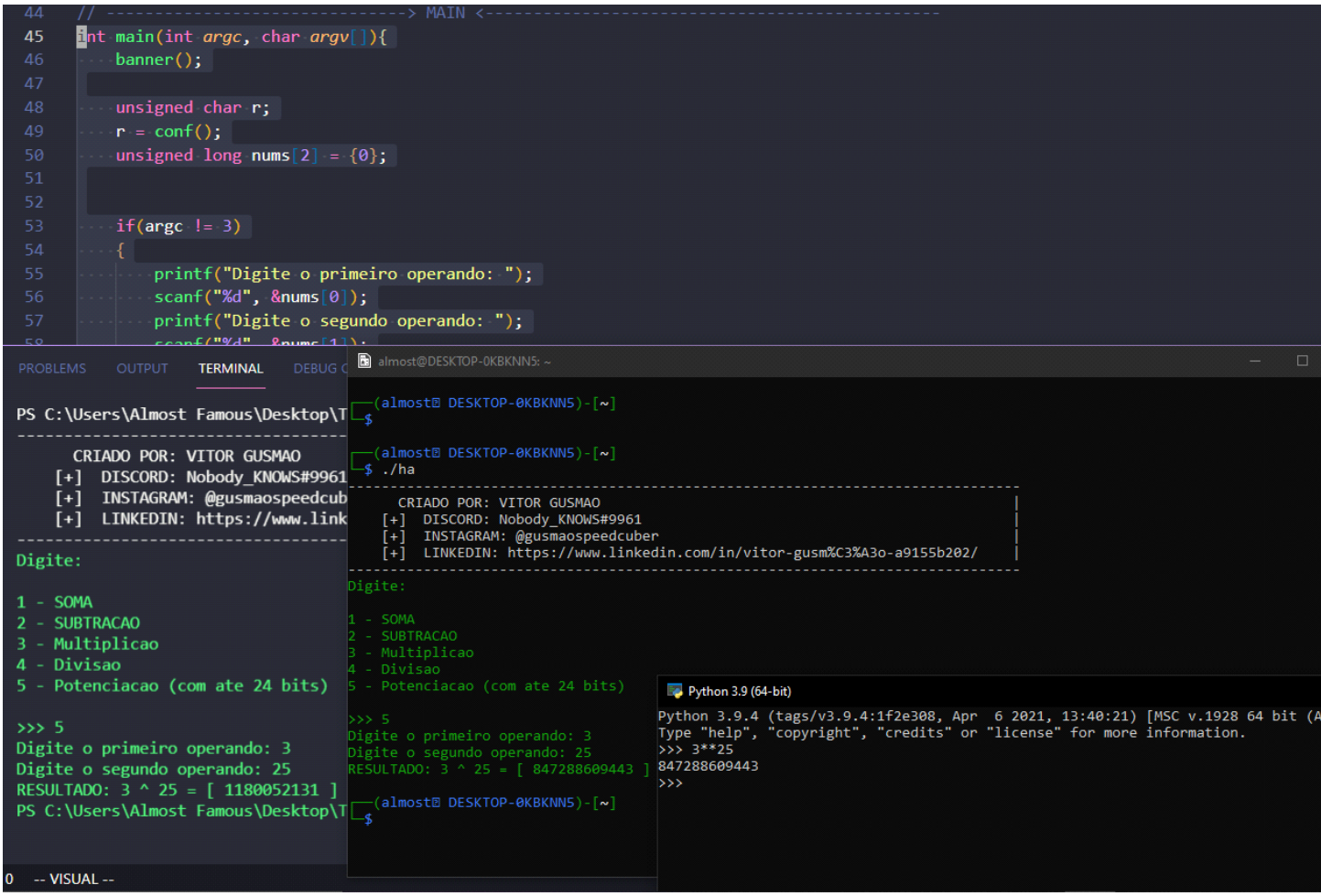
```
    else {
        nums[0] = atol(argv[1]);
        nums[1] = atol(argv[2]);

        if (!nums[0] || !nums[1]){
            fprintf(stderr, "\x1b[31m MODO DE USO: %s <NUMBER_1>
<NUMBER_2>\x1b[0m", argv[0]);
            return 1;
        }
    }

    if (r == '5' && power(nums[0], nums[1]) < 0) // Ocorreu um overflow
    {
        fprintf(stderr, "\x1b[31m Ocorreu um overflow... Tente novamente
mais tarde.\x1b[0m", argv[0]);
        return 1;
    }
```

Este código acima é o da Main. No final do código existe uma verificação de overflow que neste caso serve mais como uma proteção extra que poderia ocorrer em alguns casos (que ocorre apenas no windows, pois o compilador de C para Windows NÃO é capaz de compreender o tipo **unsigned long** como sendo de 64 bits, e resulta no mesmo que **unsigned int** de 32 bits e ainda com sinal.

Esse é apenas um exemplo de diversos para estar na ponta da lingua quando o usuário perguntar ("por que sistemas *Unix são melhor para programar?")



Como pode-se vizualizar, o exato mesmo código quando executado no Windows gera um resultado TOTALMENTE diferente do correto. Pode-se verificar utilizando o python para calcular 3**25 e tirar a prova real.

Agora, quando executado num interpretador WSL de Linux, o resultado vem como o esperado. A linguagem C foi criada pensando em desenvolver o Unix, daí a explicação de fazer mais sentido funcionar 100% com esses sistemas.

Também é daí que vem a explicação do porquê C++ e C# foram criados! Para terem 100% de compatibilidade com o Windows (já que seu próprio Kernel foi escrito utilizando as duas descritas)

---

switch(r)

```
   {
      case '1':
         printf("RESULTADO: \x1b[32m%ld + %ld = [ %ld ]\n\x1b[0m",
nums[0], nums[1], nums[0] + nums[1]);
         return 0;

      case '2':
         printf("RESULTADO: \x1b[32m%ld - %ld = [ %ld ]\n\x1b[0m",
nums[0], nums[1], nums[0] - nums[1]);
         return 0;

      case '3':
         printf("RESULTADO: \x1b[32m%ld * %ld = [ %ld ]\n\x1b[0m",
nums[0], nums[1], nums[0] * nums[1]);
         return 0;

      case '4':
         printf("RESULTADO: \x1b[32m%ld / %ld = [ %.2f ]\n\x1b[0m",
nums[0], nums[1], nums[0] / nums[1]);
         return 0;

      case '5':
         printf("RESULTADO: \x1b[32m%ld ^ %ld = [ %ld ]\n", nums[0],
nums[1], power(nums[0], nums[1]));
         return 0;
   }
// ------------------------------> END Switch
<==========================================================
======
return 0;
}
```

Isso é o que vem depois da Main. É basicamente a conferência do que o usuário digitou utilizando um Switch case para tal proeza.

Funcoes:

```
// ------------------------------------------------------------------>
unsigned long pow_recursive(unsigned long n, int times)
```

```
{
    if (times == 1)
        return n;
    return n * pow_recursive(n, times-1);
}

unsigned long power(unsigned long n, int times)
{
    unsigned long number = n;

    if(times <= 50){  // Recursivo apenas para casos que precise exxecutar
ate 50 vezes.
        number = pow_recursive(n, times);
        return number;
    }

    else {
        unsigned long aux = n;

        #pragma omp parallel   // Isso faz usar todos os nucleos do
processador ao mesmo tempo, ao inves de um apenas  para processar o
loop
        while(times--)
            n *= aux;
    }

return n;
}
```

Decidi criar duas funções para realizar os calculos de Exponenciacao. Sendo uma delas executada de forma recursiva apenas se o usuario elevar um numero ate 50. Isso para proteger o gasto excessivo de memoria do usuario.

E utilizei um recurso da linguagem C **#pragma omp parallel** para basicamente utilizar todos os nucleos da maquina para processar esse while loop. Ao invés de apenas uma.

CODIGO FINAL:

```
// gcc calculadora.c -o calculadora -O2 -w

#include <stdio.h>
#include <string.h>
#include <unistd.h>

int msleep(unsigned int tms) {
  return usleep(tms * 1000);
}

void pslp(const char *msg){
    printf("%s\n",msg);
    msleep(500);
}

void banner()
{
    pslp("-------------------------------------------------------------------------------");
    pslp("      CRIADO POR: VITOR GUSMAO
|");
    pslp("    [+]  DISCORD: Nobody_KNOWS#9961
|");
    pslp("    [+]  INSTAGRAM: @gusmaospeedcuber
|");
    pslp("    [+]  LINKEDIN: https://www.linkedin.com/in/vitor-
gusm%C3%A3o-a9155b202/    |");
    pslp("-------------------------------------------------------------------------------");
}

unsigned long pow_recursive(unsigned long n, int times);
unsigned long power(unsigned long n, int times);
// ------------------------------------------------>
unsigned char conf(){
    printf("\x1b[32mDigite:\n\n1 - SOMA\n2 - SUBTRACAO\n3 -
Multiplicao\n4 - Divisao\n5 - Potenciacao \n\n>>> ");

    unsigned char r;
```

```
    scanf("%c", &r);

    while(r < '1' || r > '5'){
        fprintf(stderr, "\x1b[31mDigite um numero entre 1 e 4, por
favor!\n\x1b[0m");
        printf("\x1b[32mDigite:\n\n[ 1 ]-> SOMA\n[ 2 ] -> SUBTRACAO\n[ 3
] -> Multiplicao\n[ 4 ] -> Divisao\n[ 5 ] -> Potenciacao\n\n>>>
\x1b[33m");
        scanf("%c", &r);
    }

    return r;
}

// ------------------------------> MAIN <--------------------------------------------
int main(int argc, char argv[]){
    banner();

    unsigned char r;
    r = conf();
    unsigned long nums[2] = {0};


    if(argc != 3)
    {
        printf("Digite o primeiro operando: ");
        scanf("%d", &nums[0]);
        printf("Digite o segundo operando: ");
        scanf("%d", &nums[1]);
    }

    else {
        nums[0] = atol(argv[1]);
        nums[1] = atol(argv[2]);

        if (!nums[0] || !nums[1]){
            fprintf(stderr, "\x1b[31m MODO DE USO: %s <NUMBER_1>
<NUMBER_2>\x1b[0m", argv[0]);
            return 1;
        }
```

```
    }

    if (r == '5' && power(nums[0], nums[1]) < 0) // Ocorreu um overflow
    {
        fprintf(stderr, "\x1b[31m Ocorreu um overflow... Tente novamente
mais tarde.\x1b[0m", argv[0]);
        return 1;
    }

// ----------------------------> Switch
<==================================================
======
    switch(r)
    {
        case '1':
            printf("RESULTADO: \x1b[32m%ld + %ld = [ %ld ]\n\x1b[0m",
nums[0], nums[1], nums[0] + nums[1]);
            return 0;

        case '2':
            printf("RESULTADO: \x1b[32m%ld - %ld = [ %ld ]\n\x1b[0m",
nums[0], nums[1], nums[0] - nums[1]);
            return 0;

        case '3':
            printf("RESULTADO: \x1b[32m%ld * %ld = [ %ld ]\n\x1b[0m",
nums[0], nums[1], nums[0] * nums[1]);
            return 0;

        case '4':
            printf("RESULTADO: \x1b[32m%ld / %ld = [ %.2f ]\n\x1b[0m",
nums[0], nums[1], nums[0] / nums[1]);
            return 0;

        case '5':
            printf("RESULTADO: \x1b[32m%ld ^ %ld = [ %ld ]\n", nums[0],
nums[1], power(nums[0], nums[1]));
            return 0;
    }
```

```
// ----------------------------> END Switch
<==========================================================
======

   printf("\n >>>>>>> %ld <<<<< \n", power(3,30));

return 0;
}

// ------------------------------------------------------------------->
unsigned long pow_recursive(unsigned long n, int times)
{
   if (times == 1)
      return n;
   return n * pow_recursive(n, times-1);
}

unsigned long power(unsigned long n, int times)
{
   unsigned long number = n;

   if(times <= 50){  // Recursivo apenas para casos que precise
exxecutar ate 50 vezes.
      number = pow_recursive(n, times);
      return number;
   }

   else {
      unsigned long aux = n;

      #pragma omp parallel   // Isso faz usar todos os nucleos do
processador ao mesmo tempo, ao inves de um apenas  para
processar o loop
      while(times--)
         n *= aux;
   }

return n;
}
```