שנקר - בי"ס גבוה להנדסה ולעיצוב
רחוב אנה פרנק 12, רמת גן 52526
טל.: 03-6110000 פקס: 03-7521141
דוא"ל: info@shenkar.ac.il

Software Engineering Department

Computer Organisation and Programming Course
final assignment

Pocket Calculator application

Written by:

Bader Daka 208219212

&

Ameen Abd Elghni 207888702

14 August 2020

Lecturer: Dr Yigal Hoffner

TABLE OF CONTENTS

1 Pocket Calculator application design

1.1 Extra Work Carried Out (extra 25% items)

- Input just number between 0-9
- checking for input over and under flow and check the result in the range
- fast multiplication
- for sophisticated input

1.2 Major Design/Implementation Decisions

The following major design decisions were made during the design and implementation phases:

- We have decided to request the **operator** from user, **before** the 2 numbers,
  Because when getting a number from user, it's been echoed right away and we want
  To separate the two numbers by the operator so it would actually seem like an exercise. (e.g: 50*6)

- Operator input check, instead of checking each time if the input is **not equal** to (+,-,*,/,X)
  And then writing an error message to screen, we have chosen to check for **equality** occurrence.
  When equality wasn't been found, automatically an error message is being printed to screen.
  It may seem less pleasant but it is actually more efficient than writing a long number of code
  Lines which their only job is to check for an unequal char.

Implementation decisions

- Code order, we have decided to start programming the Main function first.
  This way we demand our functions (such as: phraseCout,MultiplyFunc etc.) to get a certain
  Variable(s) and return a specific value, So when we are starting to program any function
  We know right away what will it recieve from "Main" and what will it return.

- Handling the sign of final result.
  As we all know, there are 4 situations which decides the sign of a result during a multiplication
  Or division actions: [(-) (-) = + , (-) (+) = - , (+) (+) = + , (+) (-) = -].
  To determain which one is suitable to a current exercise, we need to know two things:
  1. If the left number is positive/negative.
  2. If the right number is positive/negative.

  In MultiplyFunc:
  1. The sign of both numbers is being checked and stored.
  2. Checking if a result is a postive or negative after that doing the fast multiply
  3. After the above has been done, all we have to do is – take care of the result sign.

  In DivideFunc:
  1. The sign of both numbers is being checked and stored.
  2. if the right number is positive, it will be turned immediately to negative, nothing will be done otherwise.
  3. if the left number is negative, it will be turned immediately to positive, nothing will be done otherwise.

  ** These 3 steps are being done to ensure handling with one situation always. (left number Is always positive and the right one is always negative).

  4. After the above has been done, all we have to do is – take care of the result sign.

- Real-time numbers input error check
  For efficiency reasons, we have decided to check for input errors right after each char is
  Being obtained.
  If this check would have been performed only after the whole number has been inserted
  It would have waste a precious time.

1.3 The High-level Algorithms

```
#define max 32768

#define min -32767

                                        /////// FUNCTIONS  ///////

int add (int a, int b)

{


R=X+Y

 V=0;
 IF ( Xmsb == Ymsb )

 THEN IF ( Xmsb != Rmsb )
         THEN V =1  // printf("Big Number" )//Over&&Under
         ELSE  Return R;
 FI

}


int sub (int a, int b)

{
         b = b * (-1);
         return a+b;

}


int mul (int a, int b)

{
       MultResult = 0;
       FOR (DIGIT_COUNT1 = 0; DIGIT_COUNT1<16; DIGIT_COUNT1++)
       DO
            binary_digit = get next binary digit of multiplier from right to left;
            IF (binary_digit == 1)
            THEN MultResult = MultResult + multiplicand;
            FI;
            Shift(multiplicand) 1 place to left;
       END;
       Return MultResult;

       int divide (int a, int b)
       {
       int Counter=0;
         b = b * (-1);
         while (a > 0)
                  {
                           a = a + b;
                           Counter++;
                  }
         return Counter;
         }
```

```
  ///////  MAIN  //////
int main()
{
 Main,    int num1,num2;
          char oper=NULL;
          while (1)
          {
          Printf("Please choose operator +, -, *, / (X to end: ");
          Scanf("%c",oper);
          if (oper + (-'X') == 0) { return -1; }
          printf("Please enter two numbers: ");
          Scanf("%d",num1);
          Printf("%c\n",oper);
          Scanf("%d",num2);
          if ((num1 + (-max) > 0) || (num1+ (-min) < 0) || (num2 + (-max) > 0) || (num2 + (-min) < 0))
                    {
                              Printf("Big Number");
                              Goto main;
                    }
          if (oper + (-'+') == 0) { Result= add(num1,num2));
          IF(Result<MAX&& Result>MIN)
          THEN     Printf("Result:%d", Result);
          ELSE     Printf("Big Number");
          else if (oper + (-'-') == 0) { Result = sub(num1,num2);}
          THEN     Printf("Result:%d", Result);
          else if (oper + (-'*')  == 0) { Result = mul(num1,num2);}
          IF(Result<MAX&& Result>MIN)
          THEN     Printf("Result:%d", Result);
          ELSE     Printf("Big Number");
          else if (oper + (-'/')  == 0) { Result= divide(num1,num2);}
          Printf("Result:%d", Result);
          else printf("Invalid input");
          }
          return 0;
}
```

2 The User Guide

Mano Pocket Calculator
This is a signed integer calculator, no fractions support.
The calculator supports 4 basic operators: +,-,/,* and numbers range between (-32,767) – (+32,768).

To perform some kind of excercise between 2 numbers, first you need to enter the operator.
Secondly, the 1st number, then press Carriage Return and then the 2nd number.

Extra features:

1) Invalid number input check (out of the range stated above).
2) Input errors check, throughout **any** input.
3) Invalid math-actions check. (e.g: deviding by zero)

In case you choose to quit, when asked to input an operator – insert 'X' to exit.