**Software Engineering Department**

**Computer Organisation and Programming Course   
final assignment**

**Pocket Calculator application**

**Written by:**

**Yarden Hazan**

**&**

**Gal Argov Sofer**

**5 September 2016**

**Lecturer: Dr Yigal Hoffner**

TABLE OF CONTENTS

TABLE OF CONTENTS 1

1 Pocket Calculator application design 2

1.1 A few words about the project 2

1.2 Major design/implementation decisions 2

1.3 The high-level algorithms 3

The User Guide 6

2 Program listing 7

# Pocket Calculator application design

## A few words about the project

We designed this calculator to offer the most accurate solutions considering the following parameters:

-running time.

-Information storage.

we worte a high lavel code to simplify the process.

We followed the emphasis given at homework and classroom lectures.

## Major design/implementation decisions

The design process of the calculator we followed by high level code.

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

* One of the guidelines of the project were to formulate a uniform variables. And maintain the proper flow of the program, with emphasis on proper management of memory, and application of algorithms with reasonable running time.
* We put maximum emphasis on maximizing the features defined in the simulator means that the correct use variables, functions, arrays conditions and so forth.
* One of the most important decisions we took were actually try to isolate each time the input and get involved in activities on it.
* The design process of the calcloter done by modular way,from big picture to small deatails,main code which served as a guidlines and implemnation of other parts of code.

## The high-level algorithms

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

//function

void Calc(int NumA,int NumB,char Op) //calculator function 4 action

{

int Result=0, Remainder=0;

if(Op == '+'){

Result=Addition(NumA,NumB);

PrintDec(Result);

}

if(Op == '-'){

Result=Substraction(int NumA,int NumB);

PrintDec (Result);

}

if(Op == '\*'){

Result=NumA\*NumB;

PrintDec(Result);

}

if(Op=='/'){

Remainder=NumA%NumB;

Result=Divison(NumA,NumB,Remainder);

PrintDecDiv(Result,Remainder);

}

return;

}

void PrintDec (int Resault){

printf("=%d\n",Resault);

return;

}

void PrintDecDiv (int Resault, int Remainder){

printf("=%d(%d)\n",Resault,Remainder);

return;

}

int Error(char PtrNum)

{

if(PtrNum < '48' || ptrNum > '57' ){

printf("error");

return 1;

}

return 0;

}

/////////////////////

void Gets(int num,int &ptrNum[0])

{

int i=0,j=0;

while(num != 'CR')

{

scanf("%c",&num);

ptrNum1[i]=num; //assume the array under 5 digit

i++;

}

}

////////////////////////////////////

int CheckFirst(char TPtr) //cheak if the coe is nagtive or postive

{

if(TPtr == '-')

return -1;

else

return 1;

}

int Getnumber(int &ptrNum[0],int size)

{

int i=0;

while (size>0)

{

sum=\*ptrNum[i]\*10+\*ptrNum[i+1];

size--;

}

return sum;

}

int Addition(int NumA,int NumB)

{

int result;

result=NumA+numB;

return result;

}

int Substraction(int NumA,int NumB)

{

int result;

result=NumA+numB;

return result;

}

int Multiplication(int NumA, int NumB){

boolean neg = false;

if(NumA < 0 && NumB>= 0){

NumA = -NumA;

neg = true;

}

else if(NumB < 0 && NumA >= 0){

NumB = -NumB;

neg = true;

}else if( NumA < 0 && NumB < 0){

NumA = -NumA;

NumB = -NumB;

}

int res = 0;

while(NumB!=0){

if((NumB & 1) == 1) res += x;

NumA <<= 1;

NumB >>= 1;

}

return neg ? (-res) : res;

}

int division(int NumA, int NumB, int \*remainder) {

int quotient = 0;

while (NumA - NumB > 0) {

NumA -= NumB; // (if numerator is not equal or less than the divisor)

quotient++;

}

remainder = NumA;

return quotient;

}

void main ()

{

int i=0,j=0, coe,checkerror;

ptrNum1[4]={};

ptrNum2[4]={};

printf("Hello,please enter a number,another number and opeartor.\n");

printf("q to quit.\n");

printf("1st:");

scanf("%c",&ptrNum1[i]);

start do{

while(ptrNum1[i] != 'q')

{

if (CheckFirst(ptrNum1[i])!=-1){

coe=1;}

else{

coe=-1;

}

Gets(i,ptrNum1[i]));

checkerror=Error(ptrNum1[i]);

if(checkerror==1)

goto start;

else

NumA=GetNum(ptrNum1[i]);

++i;

}

NumA=NumA\*coe;

printf("%d",NumA);

printf("2nd:");

scanf("%c",&ptrNum2[i]);

while(ptrNum2[j] != 'q')

{

if (checkfirst(ptrNum2[j])!=='-'){

coe2=-1;}

else{

coe2=1;

}

Gets(i,ptrNum2[j]));

checkerror=Error(ptrNum1[i]);

if(checkerror==1)

goto start ;

else

NumB=getnum(ptrNum2[j];

++j;

}

NumB=NumB\*coe2;

printf("%d",NumB);

printf("Op");

scanf("%c",&OpPP);

printf("%c",Op);

calc(NumA,NumB,Op);

}

}

# The User Guide

1. Instructions on how to use the application

- Enter the input style:

Tap a number from the range of (- 32,768 to 32,767) and press Enter, then tap another number in the same range and tap Enter , then choose an operator ('+', '-', '\*', '\') and press enter then the result is calculated.

NUMBER1(Enter) NUMBER2(Enter) OPERATOR(Enter) =Result

# Program listing

Your code goes here with comments – PLEASE make sure you use font 9 with line spacing of 9 points:

**// Your Name: Yarder Hazan & Gal Argov Sofer**

**// Main (func)**

**WelcomeLoop, LDA PrintWel  I**

**OUT**

**ISZ  PrintWel**

**SZA**

**BUN  WelcomeLoop // printf("Hello, please enter a number, another number and operator.\n");**

**InstructionsL, LDA Instructions  I**

**OUT**

**ISZ  Instructions**

**SZA**

**BUN  InstructionsL // printf("q to quit");**

**StartCalc, CLA // while (PtrNum1[0] != 'q')**

**// {**

**1stPrint, LDA 1stPP  I //  printf("1st:");**

**OUT**

**ISZ  1stPP**

**SZA**

**BUN  1stPrint**

**LDA  1stPPS**

**STA  1stPP**

**LDA  PtrNum1**

**BSA  Gets // Input string 1**

**STA CountNum1**

**CheckQuit,  LDA  PtrNum1 I        // if (PtrNum1[0] == 'q)  goto Bye**

**CMA**

**INC**

**ADD  Quit**

**SZA**

**BUN  CHECKERR // else goto 2ndPrint**

**BUN  Bye**

**CHECKERR, LDA  ErrFlag1**

**SPA**

**BUN  NoErr**

**BSA PrintErr1**

**LDA  ErrPtr1S**

**STA  ErrPtr1**

**BUN  ClearAllData**

**BUN  StartCalc**

**NoErr, LDA  PtrNum1**

**BSA  CheckFirst // Check if Num1 is negative**

**STA FirstNeg**

**LDA CountNumT**

**STA CountNum1 // Save number of digits in num1**

**LDA FirstNeg**

**STA NumNeg // Save flag negative for Func**

**LDA PtrNum1**

**BSA GetNum // Convert strings to num (int)**

**STA NumA**

**2ndPrint, LDA 2ndPP  I**

**OUT**

**ISZ  2ndPP**

**SZA**

**BUN  2ndPrint**

**LDA  2ndPPS**

**STA  2ndPP**

**LDA PtrNum2**

**BSA  Gets // Input string 2**

**STA CountNum2**

**CheckQuit2, LDA  PtrNum2 I   // if (PtrNum2[0] == 'q)  goto Bye**

**CMA**

**INC**

**ADD  Quit**

**SZA**

**BUN  CHECKERR2 // else goto 2ndPrint**

**BUN  Bye**

**CHECKERR2, LDA  ErrFlag1**

**SPA**

**BUN  NoErr2**

**BSA PrintErr1**

**LDA  ErrPtr1S**

**STA  ErrPtr1**

**BUN  ClearAllData**

**BUN  StartCalc**

**NoErr2, LDA  PtrNum2**

**BSA  CheckFirst // Check if Num2 is negative**

**STA SecondNeg**

**LDA CountNumT**

**STA CountNum2 // Save number of digits in num2**

**LDA SecondNeg**

**STA NumNeg // Save flag negative for Func**

**LDA PtrNum2**

**BSA GetNum // Convert strint to num (int)**

**STA NumB**

**OpPrint, LDA OpPP  I //input an operator**

**OUT**

**ISZ  OpPP**

**SZA**

**BUN  OpPrint**

**LDA  OpPPS**

**STA  OpPP**

**LDA PtrOp // Load Pointer**

**BSA  Gets // Gets(PtrOp)**

**Calcc, LDA FirstNeg // representaion in 2's complement for number 1**

**SPA**

**BUN Next**

**LDA NumA**

**CMA**

**INC**

**STA NumA**

**Next, LDA SecondNeg // representaion in 2's complement for number 2**

**SPA**

**BUN CheckPlus**

**LDA NumB**

**CMA**

**INC**

**STA NumB**

**CheckPlus, LDA PtrOp I**

**STA Op**

**CMA**

**INC**

**ADD Plus**

**SZA**

**BUN CheckMin**

**LDA NumA**

**ADD NumB**

**STA Resault**

**BUN EqPrint**

**CheckMin, LDA PtrOp I**

**STA Op**

**CMA**

**INC**

**ADD Minus**

**SZA**

**BUN CheckMulti**

**LDA NumB**

**CMA**

**INC**

**ADD NumA**

**STA Resault**

**BUN EqPrint**

**CheckMulti, LDA PtrOp I**

**STA Op**

**CMA**

**INC**

**ADD Multipl**

**SZA**

**BUN CheckDiv**

**BSA Multiple**

**BUN EqPrint**

**CheckDiv, LDA PtrOp I**

**STA Op**

**CMA**

**INC**

**ADD Division**

**SZA**

**BUN  ErrorInOp**

**CheckDivZero,  LDA  NumB**

**SZA**

**BUN  Divvvvv**

**BSA PrintErr2**

**LDA  ErrPtr2S**

**STA  ErrPtr2**

**BUN  ClearAllData**

**BUN  StartCalc**

**Divvvvv, BSA  DivFunc**

**BUN  EqPrint**

**ErrorInOp, ISZ Y**

**BSA  PrintDec**

**CLA**

**BUN  StartCalc**

**Bye, HLT**

**// Gets (func)**

**Gets, HEX 0**

**STA  TPtr**

**CLA**

**STA CountNumT**

**Loop1, SKI**

**BUN Loop1**

**INP**

**OutLoop, SKO**

**BUN OutLoop**

**OUT**

**STA Temp**

**LDA  Temp**

**ADD MinusEnter**

**SZA**

**BUN CheckOF**

**BUN FI1**

**CheckOF, LDA Temp**

**ADD  RangeMax**

**SPA**

**BUN  CheckUF**

**ISZ  ErrFlag1**

**CheckUF, LDA  Temp**

**ADD  RangeMin**

**SNA**

**BUN  Body**

**ISZ  ErrFlag1**

**Body, LDA Temp**

**STA TPtr I**

**ISZ TPtr**

**ISZ CountNumT**

**BUN Loop1**

**FI1, LDA Enter**

**STA TPtr I**

**ISZ TPtr**

**LDA CountNumT**

**BUN Gets I**

**// Check if the first char in string is minus (func)**

**CheckFirst, HEX 0**

**STA TPtr**

**CLA**

**STA NumNeg**

**LDA TPtr  I**

**CMA**

**INC**

**ADD Minus**

**IF3, SZA // if(array[0] == '-')**

**BUN End**

**Body1, CLA**

**ISZ NumNeg // FirsrMin++;**

**CLA**

**LDA Minus1**

**ADD CountNumT**

**STA CountNumT // CountNumT--;**

**End, LDA NumNeg**

**BUN CheckFirst I**

**// Convert srting to variable (func)**

**GetNum, HEX 0**

**STA TPtr**

**LDA NumNeg**

**SPA**

**BUN GetNumS**

**ISZ TPtr**

**GetNumS, LDA CountNumT**

**SPA**

**BUN GetNumE**

**LDA Num1**

**ADD Num1**

**STA NumT**

**LDA Num1**

**CIL**

**CIL**

**CIL**

**ADD NumT**

**STA Num1**

**ADD TPtr  I**

**ADD HexOffset**

**STA Num1**

**ISZ TPtr**

**LDA CountNumT**

**ADD Minus1**

**STA CountNumT**

**BUN GetNumS**

**GetNumE, LDA Num1**

**STA NumT**

**CLA**

**STA Num1**

**LDA NumT**

**BUN GetNum  I**

**// Calculate multiplication (fucn)**

**Multiple, HEX  0**

**CLA**

**STA FlagMinA**

**STA  FlagMinB**

**LDA  NumA**

**SNA**

**BUN   CheckNumB**

**CMA**

**INC**

**STA NumA**

**ISZ FlagMinA**

**CheckNumB, LDA  NumB**

**SNA**

**BUN   WhileLoop2**

**CMA**

**INC**

**STA NumB**

**ISZ FlagMinB**

**WhileLoop2, LDA FuncCounter //**

**SZA // WHILE(FuncCounter > 0)**

**BUN MultiBody //**

**BUN SetSign //**

**MultiBody, LDA NumA //  NUM1 >> = 1;**

**CIR //**

**STA NumA //**

**SZE //  IF(E == 0)**

**BUN DoMulti //**

**BUN OdMulti //**

**DoMulti, LDA Resault // Resault += NUM2;**

**ADD NumB //**

**STA Resault //**

**OdMulti, LDA NumB //**

**CIL //  NUM2 << = 1;**

**STA NumB //**

**ISZ FuncCounter //  FuncCounter++;**

**BUN WhileLoop2 //  goto WHILE5;**

**SetSign, LDA  FlagMinA**

**SPA**

**BUN  CheckFlagB**

**LDA  FlagMinB**

**SPA**

**BUN     PrintMinus**

**BUN Func\_Out**

**CheckFlagB, LDA  FlagMinB**

**SPA**

**BUN Func\_Out**

**PrintMinus, ISZ PriMinusDev**

**Func\_Out, BUN Multiple I  // end func**

**// Calculate division (fucn)**

**DivFunc, HEX  0**

**CLA**

**STA FlagMinA**

**STA  FlagMinB**

**LDA  NumA**

**SNA**

**BUN   CheckNumB2**

**CMA**

**INC**

**STA NumA**

**ISZ FlagMinA**

**CheckNumB2, LDA  NumB**

**SNA**

**BUN   WhileLoop3**

**CMA**

**INC**

**STA NumB**

**ISZ FlagMinB**

**WhileLoop3, LDA NumB**

**CMA**

**INC**

**ADD NumA**

**SNA**

**BUN  BIG**

**BUN LESS**

**BIG, STA  NumA**

**ISZ Resault**

**BUN WhileLoop3**

**LESS, LDA NumA**

**STA Remainder**

**SetSign2, LDA  FlagMinA**

**SPA**

**BUN  CheckFlagB2**

**LDA  FlagMinB**

**SPA**

**BUN     PrintMinus2**

**BUN Func\_Out2**

**CheckFlagB2, LDA  FlagMinB**

**SPA**

**BUN Func\_Out2**

**PrintMinus2, ISZ  PriMinusDev**

**Func\_Out2, BUN DivFunc I  // end func**

**// Print resault (func)**

**PrintDec, HEX  0**

**CLA**

**EqPrint, LDA EqPP  I**

**OUT**

**ISZ  EqPP**

**SZA**

**BUN  EqPrint**

**CLA**

**LDA  EqPPS**

**STA  EqPP**

**CLA**

**LDA  PriMinusDev**

**SPA**

**BUN  Go\_Flag**

**LDA  Minus**

**OUT**

**Go\_Flag, CLA**

**STA Flag\_0**

**STA Count**

**LDA BasePoint**

**STA GlobalPoint**

**LDA Resault**

**SNA**

**BUN ForLoop**

**CMA**

**INC**

**STA Resault**

**LDA Minus**

**OUT**

**ForLoop, LDA  CNT**

**SZA**

**BUN WhilLoop**

**BUN Last**

**WhilLoop, LDA Resault**

**ADD GlobalPoint  I**

**SNA**

**BUN BodyLoop**

**BUN  PrintC**

**BodyLoop, STA Resault**

**ISZ  Count**

**BUN  WhilLoop**

**PrintC, ISZ GlobalPoint**

**LDA CNT**

**INC**

**STA CNT**

**LDA Count**

**SZA**

**BUN NO\_LEAD\_0**

**LDA Flag\_0**

**SPA**

**BUN ForLoop**

**NO\_LEAD\_0, ISZ Flag\_0**

**LDA Count**

**ADD OFFSET**

**OUT**

**CLA**

**STA Count**

**BUN ForLoop**

**Last, LDA Resault**

**ADD OFFSET**

**OUT**

**LDA  Remainder**

**SPA**

**BUN  EndPrint**

**LDA  BracketL**

**OUT**

**LDA  Remainder**

**ADD OFFSET**

**OUT**

**LDA  BracketR**

**OUT**

**EndPrint, LDA W**

**OUT**

**BSA  ClearAllData**

**BUN  PrintDec I**

**ClearAllData,  HEX  0**

**CLA**

**STA  TPtr**

**STA  TPtr1**

**STA  Temp**

**STA  Temp1**

**STA  Temp2**

**STA  NumT**

**STA  Num1**

**STA  NumA**

**STA  NumB**

**STA  Op**

**STA  CountFirst**

**STA  NumNeg**

**STA  FirstNeg**

**STA  SecondNeg**

**STA  CountNumT**

**STA  CountNum1**

**STA  CountNum2**

**STA  Resault**

**STA  TResault**

**STA  Remainder**

**STA  X**

**STA  Y**

**STA  GlobalPoint**

**STA  Count**

**STA  Flag\_0**

**STA  FlagMinA**

**STA  FlagMinB**

**STA  PriMinusDev**

**STA  ErrFlag1**

**STA  ErrFlag2**

**LDA  CNTS**

**STA  CNT**

**LDA  FuncCounterS**

**STA  FuncCounter**

**LDA  WS**

**STA  W**

**BUN  ClearAllData  I**

**//PrintErr1:**

**PrintErr1, HEX 0**

**LABEL1, LDA ErrPtr1  I**

**OUT**

**ISZ  ErrPtr1**

**SZA**

**BUN LABEL1**

**BUN  PrintErr1 I**

**//PrintErr2:**

**PrintErr2, HEX 0**

**LABEL2, LDA ErrPtr2  I**

**OUT**

**ISZ  ErrPtr2**

**SZA**

**BUN  LABEL2**

**BUN  PrintErr2  I**

**// DATA**

**// Pointers to strings**

**PrintWel, HEX  300**

**OpPP,  HEX  370**

**OpPPS,  HEX  370**

**1stPP,  HEX  380**

**1stPPS, HEX  380**

**2ndPP,  HEX  390**

**2ndPPS, HEX  390**

**EqPP,  HEX  395**

**EqPPS,  HEX  395**

**Instructions, HEX  400**

**PtrNum1, HEX 500**

**PtrNum2, HEX 600**

**PtrOp, HEX 700**

**BasePoint, HEX 800**

**// Variable**

**TPtr, HEX 0**

**TPtr1, HEX 0**

**Temp, DEC 0**

**Temp1, DEC 0**

**Temp2, DEC 0**

**NumT, DEC 0**

**Num1, DEC 0**

**NumA, DEC 0**

**NumB, DEC 0**

**Op, HEX  0**

**CountFirst, DEC 0**

**NumNeg, DEC 0**

**FirstNeg, DEC 0**

**SecondNeg, DEC 0**

**CountNumT, DEC 0**

**CountNum1, DEC 0**

**CountNum2, DEC 0**

**Resault, DEC 0**

**TResault, DEC  0**

**Remainder, DEC  0**

**X, DEC 0**

**Y, DEC 0**

**GlobalPoint, HEX  0**

**Count, DEC 0**

**Flag\_0, DEC 0**

**FlagMinA, DEC  0**

**FlagMinB, DEC  0**

**CNT, DEC -4**

**CNTS, DEC -4**

**FuncCounter, DEC -16**

**FuncCounterS, DEC -16**

**W, HEX A**

**WS,  HEX  A**

**PriMinusDev,  DEC  0**

**ErrFlag1,  DEC  0**

**ErrFlag2, DEC  0**

**ErrPtr1, HEX  850**

**ErrPtr2, HEX  900**

**ErrPtr1S, HEX  850**

**ErrPtr2S, HEX  900**

**// Strings**

**ORG  300**

**Welcome, HEX 48 // 'H'**

**HEX 65 // 'e'**

**HEX 6C // 'l'**

**HEX 6C // 'l'**

**HEX 6F // 'o'  (Hello)**

**HEX 2C // ',' (,)**

**HEX 20 //  (SPACE)**

**HEX 70 // 'P'**

**HEX 6C // 'L'**

**HEX 65 // 'E'**

**HEX  61 // 'A'**

**HEX 73 // 'S'**

**HEX  65 // 'E'  (please)**

**HEX 20 //  (SPACE)**

**HEX 65 // 'E'**

**HEX 6E // 'N'**

**HEX 74 // 'T'**

**HEX  65 // 'E'**

**HEX 72 // 'R'  (enter)**

**HEX 20 //  (SPACE)**

**HEX 61 // 'A'  (a)**

**HEX 20 //  (SPACE)**

**HEX 6E // 'N'**

**HEX 75 // 'U'**

**HEX 6D // 'M'**

**HEX 62 // 'B'**

**HEX 65 // 'E'**

**HEX  72 // 'R'  (number)**

**HEX 2C // ','  (,)**

**HEX 20 //  (SPACE)**

**HEX 61 // 'A'**

**HEX 6E // 'N'**

**HEX 6F // 'O'**

**HEX 74 // 'T'**

**HEX 65 // 'E'**

**HEX 72 // 'R'**

**HEX 20 //  (SPACE)**

**HEX 6E // 'N'**

**HEX 75 // 'U'**

**HEX 6D // 'M'**

**HEX 62 // 'B'**

**HEX 65 // 'E'**

**HEX  72 // 'R'  (number)**

**HEX 20 //  (SPACE)**

**HEX 61 // 'A'**

**HEX 6E // 'N'**

**HEX 64 // 'D'**

**HEX 20 //  (SPACE)**

**HEX 6F // 'O'**

**HEX 70 // 'P'**

**HEX 65 // 'E'**

**HEX 72 // 'R'**

**HEX 61 // 'A'**

**HEX 74 // 'T'**

**HEX  6F // 'O'**

**HEX 72 // 'R'**

**HEX 2E // '.'**

**HEX D // CR**

**HEX 0 // '\0'**

**ORG  370**

**OpP, HEX  4F  // 'O'**

**HEX  70  // 'p'**

**HEX  3A  // ':'**

**HEX 0 // '\0'**

**ORG  380**

**1stP, HEX  31  // '1'**

**HEX  73  // 's'**

**HEX  74  // 't'**

**HEX  3A  // ':'**

**HEX 0 // '\0'**

**ORG  390**

**2ndP, HEX  32  // '2'**

**HEX  6E  // 's'**

**HEX  64  // 't'**

**HEX  3A  // ':'**

**HEX 0 // '\0'**

**ORG  395**

**EqP, HEX  3D  // '='**

**HEX 0 // '\0'**

**ORG  400**

**Instruct,  HEX  54  // 'T'**

**HEX  6F  // 'o'**

**HEX 20 //  (SPACE)**

**HEX  71  // 'q'**

**HEX  75  // 'u'**

**HEX  69  // 'i'**

**HEX  74  // 't'**

**HEX 20 //  (SPACE)**

**HEX  70  // 'p'**

**HEX  72  // 'r'**

**HEX  65  // 'e'**

**HEX  73  // 's'**

**HEX  73  // 's'**

**HEX 20 //  (SPACE)**

**HEX  71  // 'q'**

**HEX 2E // '.'**

**HEX D // CR**

**HEX 0 // '\0'**

**ORG  500**

**arrayNum1, HEX  0**

**HEX 0**

**HEX 0**

**HEX 0**

**HEX 0**

**HEX 0**

**HEX 0**

**ORG  600**

**arrayNum2, HEX  0**

**HEX 0**

**HEX 0**

**HEX 0**

**HEX 0**

**HEX 0**

**HEX 0**

**ORG  700**

**arrayOp, HEX  0**

**HEX 0**

**ORG  800**

**Bases, DEC -10000**

**DEC -1000**

**DEC -100**

**DEC -10**

**ORG  850**

**Error1, HEX 45 // 'E'**

**HEX 72 // 'r'**

**HEX 72 // 'r'**

**HEX 6F // 'o'**

**HEX 72 // 'r'**

**HEX 20 //  (SPACE)**

**HEX 77  // w**

**HEX 72  // r**

**HEX 6F  // o**

**HEX 6E  // n**

**HEX 67  // g**

**HEX 20 //  (SPACE)**

**HEX 64  // d**

**HEX 69  // i**

**HEX 67  // g**

**HEX 69  // i**

**HEX 74  // t**

**HEX 2E // '.'**

**HEX D // CR**

**HEX 0 // '\0'**

**ORG  900**

**Error2, HEX 45 // 'E'**

**HEX 72 // 'r'**

**HEX 72 // 'r'**

**HEX 6F // 'o'**

**HEX 72 // 'r'**

**HEX 20 //  (SPACE)**

**HEX 63 // 'c'**

**HEX 61 // 'a'**

**HEX 6E // 'n'**

**HEX 74 // 't'**

**HEX 20 //  (SPACE)**

**HEX 64 // 'd'**

**HEX 69 // 'i'**

**HEX 76 // 'v'**

**HEX 69 // 'i'**

**HEX 64 // 'd'**

**HEX 65 // 'e'**

**HEX 20 //  (SPACE)**

**HEX 62 // 'b'**

**HEX 79 // 'y'**

**HEX 20 //  (SPACE)**

**HEX 7A // 'z'**

**HEX 65 // 'e'**

**HEX 72 // 'r'**

**HEX 6F // 'o'**

**HEX 2E // '.'**

**HEX D // CR**

**HEX 0 // '\0'**

**// Operators**

**RangeMin, DEC  -42**

**RangeMax, DEC  -57**

**HexOffset, HEX     -30**

**OFFSET, HEX 30**

**Minus1, DEC -1**

**MinusTEN, DEC  -10**

**MinusEnter, DEC -13**

**Enter, HEX D**

**Plus, HEX 2B  // '+'**

**Minus, HEX 2D  // '-'**

**Minus-, HEX -2D  // '-'**

**Multipl, HEX 2A  // '\*'**

**Division, HEX 2F  // '/'**

**Equal, HEX 3D  // '='**

**BracketL,  HEX  28  // '('**

**BracketR,  HEX  29  // ')'**

**Quit, HEX  71  // 'q'**