This program is a simple calculator that reads its input from a character buffer. If integers are read, they are pushed on a stack. If one of the operators (+ - \* ⁄) is read, the top two elements are popped off the stack, the operation is performed on them, and the result is pushed on the stack. The = operator writes out the value of the top element of the stack to a buffer.

**CALC.H**

/\*----- FILE CALC.H --------------------------------------------------\*/

/\* \*/

/\* Header file for CALC.C PUSHPOP.C READTOKN.C \*/

/\* a simple calculator \*/

/\*--------------------------------------------------------------------\*/

typedef enum toks {

T\_INTEGER,

T\_PLUS,

T\_TIMES,

T\_MINUS,

T\_DIVIDE,

T\_EQUALS,

T\_STOP

} Token;

Token read\_token(char buf[]);

typedef struct int\_link {

struct int\_link \* next;

int i;

} IntLink;

typedef struct int\_stack {

IntLink \* top;

} IntStack;

extern void push(IntStack \*, int);

extern int pop(IntStack \*);

**CALC.C**

/\*----- FILE CALC.C --------------------------------------------------\*/

/\* \*/

/\* A simple calculator that does operations on integers that \*/

/\* are pushed and popped on a stack \*/

/\*--------------------------------------------------------------------\*/

#include <stdio.h>

#include <stdlib.h>

#include "calc.h"

IntStack stack = { 0 };

main()

{

Token tok;

char word[100];

char buf\_out[100];

int num, num2;

for(;;)

{

tok=read\_token(word);

switch(tok)

{

case T\_STOP:

break;

case T\_INTEGER:

num = atoi(word);

push(&stack,num); /\* CALC1 statement \*/

break;

case T\_PLUS:

push(&stack, pop(&stack)+pop(&stack) );

break;

case T\_MINUS:

num = pop(&stack);

push(&stack, num-pop(&stack));

break;

case T\_TIMES:

push(&stack, pop(&stack)\*pop(&stack));

break;

case T\_DIVIDE:

num2 = pop(&stack);

num = pop(&stack);

push(&stack, num/num2); /\* CALC2 statement \*/

break;

case T\_EQUALS:

num = pop(&stack);

sprintf(buf\_out,"= %d ",num);

push(&stack,num);

break;

}

if (tok==T\_STOP)

break;

}

return 0;

}

**PUSHPOP.C**

/\*----- FILE PUSHPOP.C -----------------------------------------------\*/

/\* \*/

/\* A push and pop function for a stack of integers \*/

/\*--------------------------------------------------------------------\*/

#include <stdlib.h>

#include "calc.h"

/\*--------------------------------------------------------------------\*/

/\* input: stk - stack of integers \*/

/\* num - value to push on the stack \*/

/\* action: get a link to hold the pushed value, push link on stack \*/

/\* \*/

extern void push(IntStack \* stk, int num)

{

IntLink \* ptr;

ptr = (IntLink \*) malloc( sizeof(IntLink)); /\* PUSHPOP1 \*/

ptr–>i = num; /\* PUSHPOP2 statement \*/

ptr–>next = stk–>top;

stk–>top = ptr;

}

/\*--------------------------------------------------------------------\*/

/\* return: int value popped from stack \*/

/\* action: pops top element from stack and gets return value from it \*/

/\*--------------------------------------------------------------------\*/

extern int pop(IntStack \* stk)

{

IntLink \* ptr;

int num;

ptr = stk–>top;

num = ptr–>i;

stk–>top = ptr–>next;

free(ptr);

return num;

}

**READTOKN.C**

/\*----- FILE READTOKN.C ----------------------------------------------\*/

/\* \*/

/\* A function to read input and tokenize it for a simple calculator \*/

/\*--------------------------------------------------------------------\*/

#include <ctype.h>

#include <stdio.h>

#include "calc.h"

/\*--------------------------------------------------------------------\*/

/\* action: get next input char, update index for next call \*/

/\* return: next input char \*/

/\*--------------------------------------------------------------------\*/

static char nextchar(void)

{

/\*--------------------------------------------------------------------\*/

/\* input action: \*/

/\* 2 push 2 on stack \*/

/\* 18 push 18 \*/

/\* + pop 2, pop 18, add, push result (20) \*/

/\* = output value on the top of the stack (20) \*/

/\* 5 push 5 \*/

/\* ⁄ pop 5, pop 20, divide, push result (4) \*/

/\* = output value on the top of the stack (4) \*/

/\*--------------------------------------------------------------------\*/

char \* buf\_in = "2 18 + = 5 ⁄ = ";

static int index; ⁄\* starts at 0 \*⁄

char ret;

ret = buf\_in[index];

++index;

return ret;

}

/\*--------------------------------------------------------------------\*/

/\* output: buf - null terminated token \*/

/\* return: token type \*/

/\* action: reads chars through nextchar() and tokenizes them \*/

/\*--------------------------------------------------------------------\*/

Token read\_token(char buf[])

{

int i;

char c;

⁄\* skip leading white space \*⁄

for( c=nextchar();

isspace(c);

c=nextchar())

;

buf[0] = c; ⁄\* get ready to return single char e.g."+" \*⁄

buf[1] = 0;

switch(c)

{

case '+' : return T\_PLUS;

case '-' : return T\_MINUS;

case '\*' : return T\_TIMES;

case '⁄' : return T\_DIVIDE;

case '=' : return T\_EQUALS;

default:

i = 0;

while (isdigit(c)) {

buf[i++] = c;

c = nextchar();

}

buf[i] = 0;

if (i==0)

return T\_STOP;

else

return T\_INTEGER;

}

}