A working simple calculator in C code(1)

/* Simple integer arithmetic calculator according to the EBNF:

<exp> → <term> { <addop> <term> }

<addop> → + | -

<term> → <factor> { <mulop> <factor> }

<mulop> → *

<factor> → (<exp>) | Number

Inputs a line of text from stdin.

Outputs “error” or the result.

*/

A working simple calculator in C code(2)

```
#include <stdio.h>
#include <stdlib.h>
char  token; /* global token variable */
/*function prototype for recursive calls*/
int exp(void);
int term(void);
int factor(void);
void error(void)
{
    fprintf(stderr, “error\n”);
    exit(1);
}
```

A working simple calculator in C code(3)

```
void match(char expectedToken)
{
    if (token==expectedToken)  token=getchar();
    else error();
}

main()
{
     int result;
     token=getchar();  /*load token with first character for lookahead*/
     result=exp();
     if (token=='\n')     /*check for end of line*/
            printf("Result = %d\n", result);
     else error();        /*extraneous chars on line*/
     return 0;
}
```

A working simple calculator in C code(4)

<exp> → <term> { <addop> <term> }

```
int exp(void)
{
    int temp=term();
    while ( (token=='+')  ||  (token=='-') )
        switch (token)
        {
        case '+':  match ('+');
                   temp+=term();
                   break;
         case '-':  match ('-');
                   temp-=term();
                   break;
        }
    return temp;
}
```

A working simple calculator in C code(5)

```
int  term(void)
{
    int temp=factor();
    while (token==‘*’)
    {
        match(‘*’);
        temp*=factor();
    }
    return temp;
}
```

<term> → <factor> { <mulop> <factor> }

<factor> → (<exp>) | Number

```
int  factor(void)
{
    int temp;
    if (token==‘(’)
    {
        match (‘(’);
        temp = exp();
        match(‘)’);
    }
    else
        if (isdigit(token))
        {
            ungetc(token,stdin);
            scanf(“%d”,&temp);
            token = getchar();
        }
        else error();
    return temp;
}
```