EXPERIMENT CALCULATOR

Agenda

- Revision on stack
- Problem description
  - Reverse Polish notation
  - Examples
- Structure of program
  - Multiple files in a project

Stack related exercises

- Give an input sequence of a stack: [1, 2, 3, 4, 5]
  - Which of the following output sequence is not possible?
    a) 5, 4, 3, 2, 1
    b) 1, 3, 2, 5, 4
    c) 1, 2, 3, 4, 5
    d) 1, 4, 2, 3, 5
  - the output sequence of a stack is [2,4,3,5,1], what is the minimal size of the stack?
  - What is the action sequence (pop/push)?

Problem description

- The problem is to write a calculator program that provides the operators +, -, *and /.
- The calculator will use reverse polish notation instead of infix. Because it is easier to implement.

Reverse polish notation

- Reverse Polish notation (RPN) is a mathematical notation in which ***every operator follows all of its operands***.
- The description "Polish" refers to the nationality of logician Jan Łukasiewicz, who invented (prefix) Polish notation in the 1920s.

Examples(1/3)

- In reverse polish notation the operators follow their operands;
  - to add 3 and 4, one would write "3 4 +" rather than "3 + 4".
- If there are multiple operations, the operator is given immediately after its second operand;
  - "3 − 4 + 5" in conventional notation would be written "3 4 − 5 +" in RPN:

Examples(2/3)

- RPN does not need parentheses that are required by infix.
  - “3 − 4 × 5” would be written as “3 4 5 × −”
  - “(3 − 4) × 5” would be written as “3 4 - 5 × ”

Algorithm

- While there are input tokens (operators or operand) left
  - Read the next token from input.
  - If the token is a operand
    - Push it onto the stack.
  - Otherwise, the token is an operator :
    - It is known *a priori* that the operator takes **n** arguments.
    - If there are fewer than **n** values on the stack
      - **(Error)** The user has not input sufficient values in the expression.
    - Else, Pop the top **n** values from the stack.
    - Evaluate the operator, with the values as arguments.
    - Push the returned results, if any, back onto the stack.
- If there is only one value in the stack
  - That value is the result of the calculation.
- Otherwise, there are more values in the stack
  - **(Error)** The user input has too many values.

Example(3/3)

Input	Operation	Stack	Comment
5	Push value	5	
1	Push value	1 5	
2	Push value	2 1 5	
+	Add	3 5	Pop two values (1, 2) and push result (3)
4	Push value	4 3 5	
×	Multiply	12 5	Pop two values (3, 4) and push result (12)
+	Add	17	Pop two values (5, 12) and push result (17)
3	Push value	3 17	
−	Subtract	14	Pop two values (17, 3) and push result (14)
	Result	(14)	

The infix expression
"5 + ((1 + 2) × 4) − 3"
can be written down like this in RPN:
5 1 2 + 4 × + 3 −

Structure of program

- 4 source files and 1 head file:
  - calc.h contains declarations of global functions (external variable) and macro definitions
  - main.c contains main function
  - stack.c contains push/pop functions of stack
  - getop.c contains getop() for fetching the next input token(opeator/operand)
  - getch.c contains get a next character or push character back on input

calc.h

```c
#define NUMBER '0'
void push(double);
double pop(void);
int getop(char []);
int getch(void);
void ungetch(int);
```

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "calc.h"
#define MAXOP 100
main() {
    ...
}
```

getop.c

```c
#include <stdio.h>
#include <ctype.h>
#include "calc.h"
getop() {
    ...
}
```

stack.c

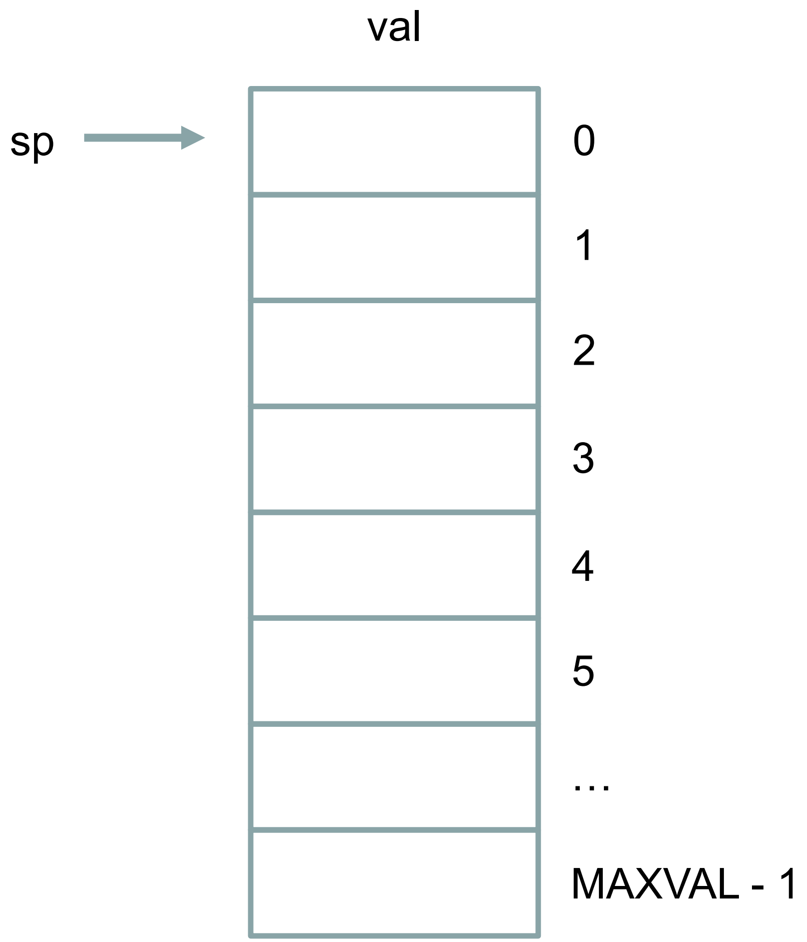
```c
#include <stdio.h>
#include "calc.h"
#define MAXVAL 100
int sp = 0;
double val[MAXVAL];
void push(double) {
    ...
}
double pop(void) {
    ...
}
```

getch.c

```c
#include <stdio.h>
#define BUFSIZE 100
char buf[BUFSIZE];
int bufp = 0;
int getch(void) {
    ...
}
void ungetch(int) {
    ...
}
```

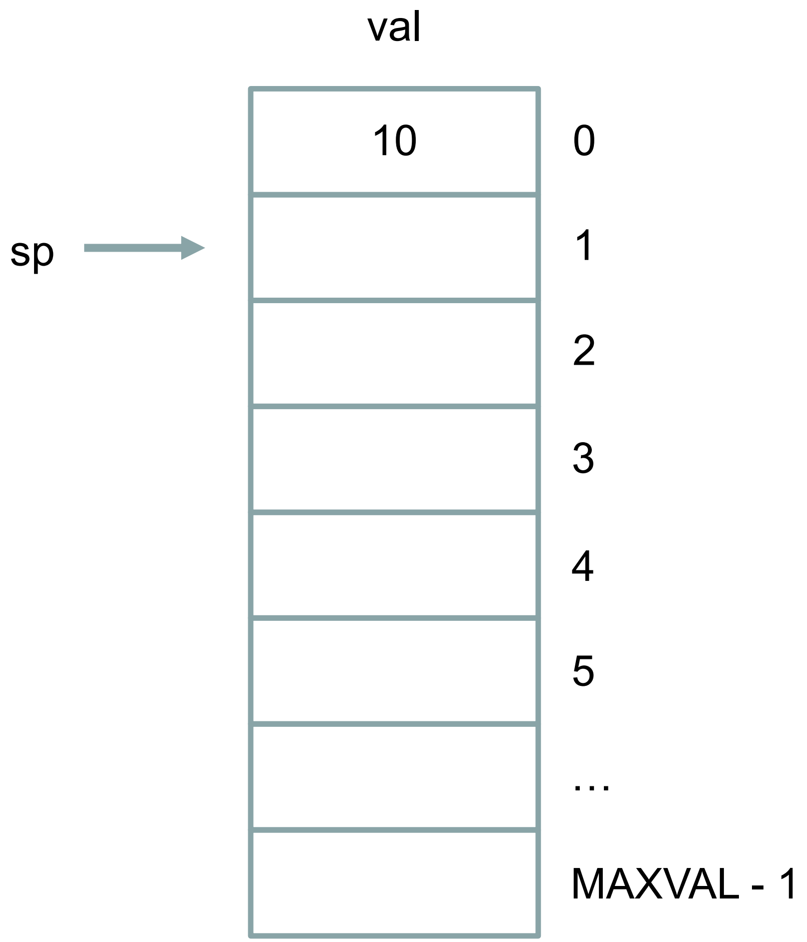
stack.c

```
void push(double f);
double pop();
```



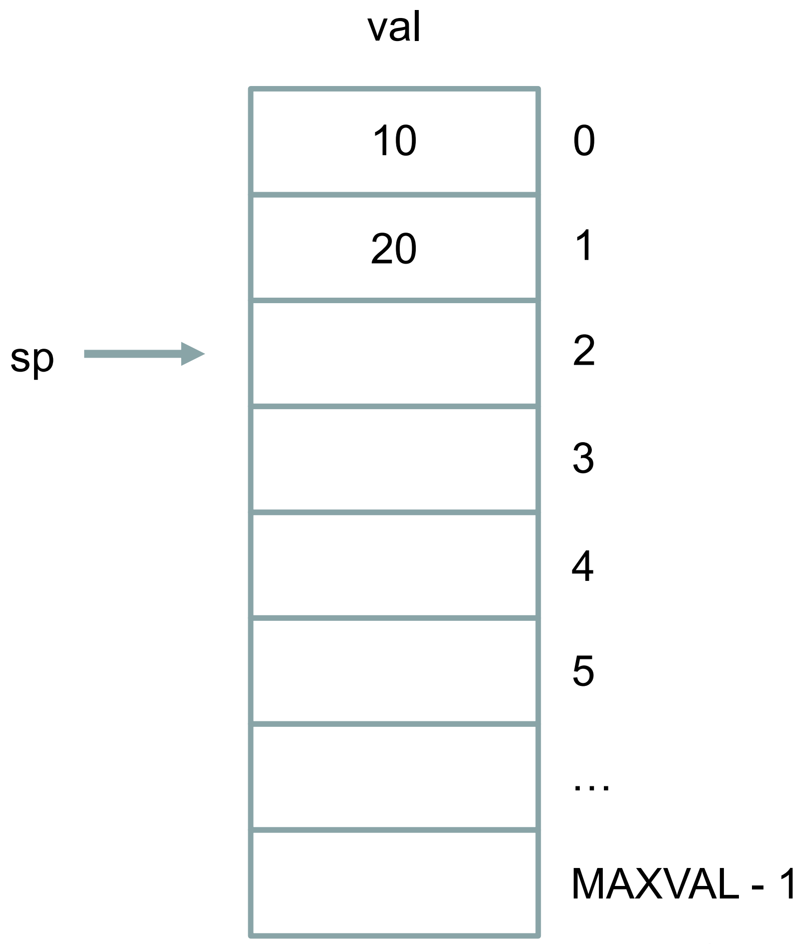
stack.c

```
void push(double f);
double pop();


push(10);
```



stack.c

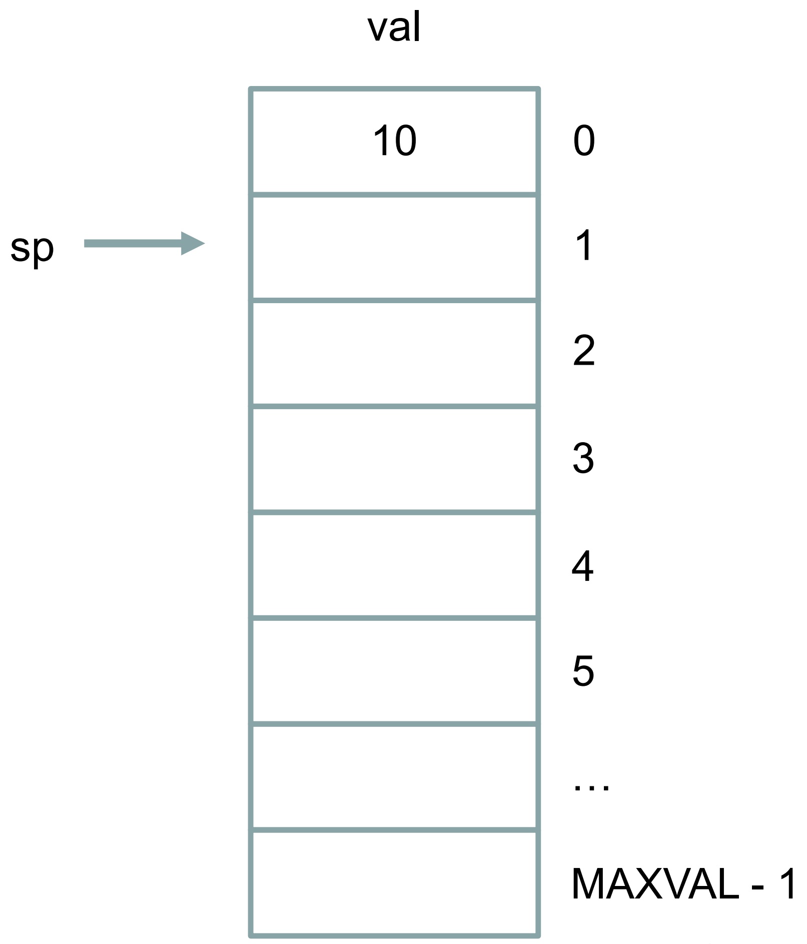
```
void push(double f);
double pop();


push(10);
push(20);
```



stack.c

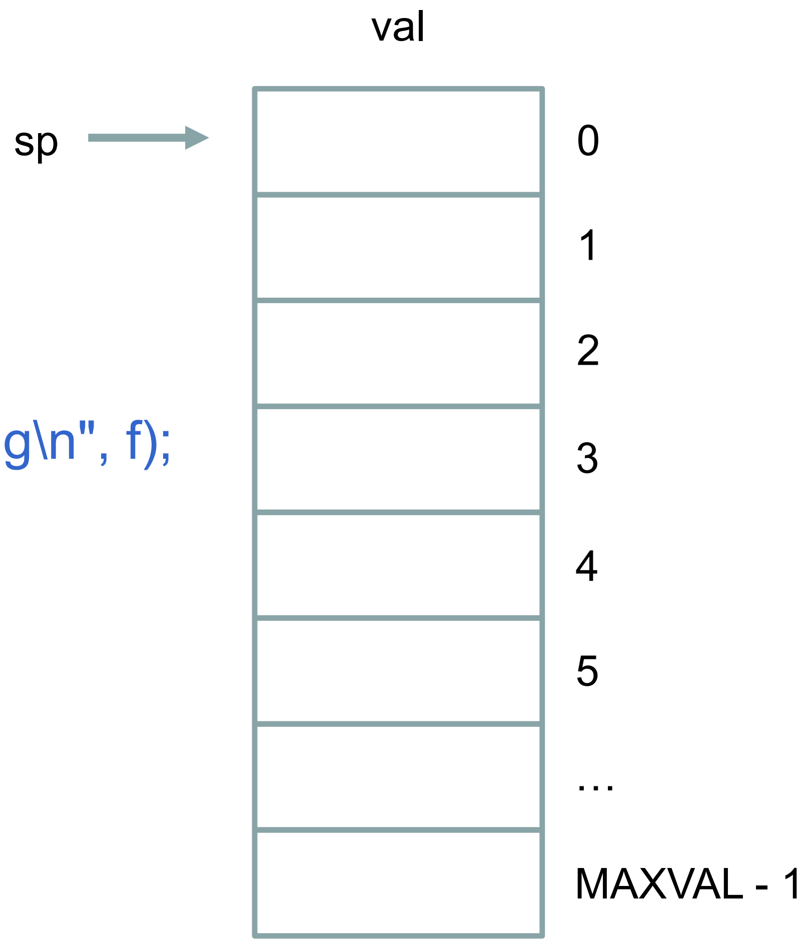
```
void push(double f);
double pop();


push(10);
push(20);
double temp = pop();
```

- temp = 20;



stack.c

```
void push(double f)
{
    if (sp < MAXVAL)
        val[sp++] = f;
    else
        printf("error: stack full, can't push %g\n", f);
}
double pop()
{…}
```



main.c

```
char s[MAXOP]; //store the operand
while ( (type = getop(s)) != EOF ) {
        //You can implement the computation with a "switch" judgement
        /*
                type == NUMBER
                        do something...
                type == + - * /
                        do something...
                type == '\n'
                        do something...
        */
  }
```

getop.c

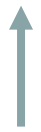
```
/* getop: get next character or numeric operand */
int getop(char s[])
{
    /*
        use getch() to read characters and ungetch() to put the characters in a buffer
which will be used next time
        hint: you may react by regarding the operand in the following types:
        operand is a separator: ' ' or '\t', etc.
        operand is a digit.
        operand is '.‘
        operator
    */
    return NUMBER; //if the operand is a number
}
```

getch.c

```
#define BUFSIZE 100
char buf[BUFSIZE]; /* buffer for ungetch */
int bufp = 0; /* next free position in buf */

int getch(void) /* get a (possibly pushed-back) character */
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}
void ungetch(int c) /* push character back on input */
{
    if (bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}
```

getop.c

12.34   56   +   78   +



```
/*1 Operator*/
int getop(char s[])
{
  /*
     if c = getch() is an operator (not a digit or '.')
     return c
  */
}
```

getop.c

```
12.34   56   +   78   +
↑
```

```
/*2 Digit*/
int getop(char s[])
{
   /*
      Store the whole number in s[ ]
      return NUMBER  // defined in calc.h
   */
}
```

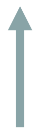
getop.c

```
12.34   56   +   78   +
```

```
/*2.1 Integer part*/
int getop(char s[])
{
   /*
      while (c = getch()) is a digit
         Store the digit of integer part in s[ ]
   */
}
```

getop.c

12.34   56   +   78   +



```
/*2.2 Point*/
int getop(char s[])
{
    /*
        if (c = getch()) is a point
        Store the point in s[ ]
    */
}
```

getop.c

12.34   56   +   78   +

```
/*2.3 Fraction Part*/
int getop(char s[])
{
    /*
        After stored the point:
        while (c = getch()) is a digit
            Store the digit of fraction part in s[ ]
when you have finished saving a number, don't forget to add a '\0' in s[ ]
         return NUMBER
    */
}
```

getop.c

```
12.34   56   +   78   +
      ↑
```

```
/*3 Separator*/
int getop(char s[])
{
   /*
      While c = getch() is ‘  ’ or ‘\t’
         Continue to do getch() until some other characters occur
   */
}
```

getop.c

```
/* getop: get next character or numeric operand */
int getop(char s[])
{
    int i, c;
    while ((s[0] = c = getch()) == ' ' || c == '\t')
        ;
    s[1] = '\0';
    if (!isdigit(c) && c != '.')
        return c; /* not a number */

    ……
}
```

main.c

```
char s[MAXOP]; //store the operand
while ( ( type = getop(s) ) != EOF ) {
        //You can implement the computation with a "switch" judgement
        /*
            type == NUMBER
                do something...
                    atof(s)  (ascii to floating point numbers)
                        recognize '\0' as the end of the number
            type == + - * /
                do something... (Pay attention to – and /)
            type == '\n'
                do something...
        */
  }
```

Test

- 1+2
- 3*4
- 1*2 - 3.14*2*2=-10.56
  - [input as] 1 2 * 3.14 2 2 * * -
- 1*3 - 3.14*1 = -0.14
  - [input as] 1 3 * 3.14 1 * -
- 1.5 + (12.345+2)/3.14 = 6.0684713
  - [input as] 1.5 12.345 2 + 3.14 / +
- 2.2 + (3.25 - 1.1/2.5) * 1.32 =5.9092
  - [input as] 2.2 3.25 1.1 2.5 / - 1.32 * +

Test.in

```
1 2 +
3 4 *
1 2 * 3.14 2 2 * * -
1 3 * 3.14 1 * -
1.5 12.345 2 + 3.14 / +
2.2 3.25 1.1 2.5 / - 1.32 * +
```

Test.out

```
3
12
-10.56
-0.14
6.0684713
5.9092
```

Furthermore

- Considering that the number can be negative
- Complete error handing