CS11001/CS11002 Programming and Data Structures (PDS) (Theory: 3-0-0)

Teacher: Sourangshu Bhattacharya
sourangshu@gmail.com
http://cse.iitkgp.ac.in/~sourangshu/

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

A complete C program

```
/* prog.c  */
#include <stdio.h>

void PRINT();
void PRINT_I(int);
void PRINT_F(float,float);

void main()
{
        int a=10;
        float x,y;
        scanf(“%d %d”,&x,&y);
        PRINT();
        PRINT_I(a);
        PRINT_F(x,y);
}
```

```
void PRINT()
{
    printf(“Hello Everybody!!!\n”);
}
```

```
void PRINT_I(int b)
{
    printf(“%d \n”,b);
}
```

```
void PRINT_F(float p, float q)
{
    printf(“%2.1f %2.2f \n”,p,q);
}
```

Compilation and Execution

```
$ cc –Wall prog.c
$
$ ./a.out
2.34
3.45
Hello Everybody!!!
10
2.3 3.45
$
```

A complete C program

```
/* prog.c  */
#include <stdio.h>

void PRINT();
void PRINT_I(int);
void PRINT_F(float,float);

void main(????)
{
        int a=10;
        float x,y;
        scanf("%d %d",&x,&y);
        PRINT();
        PRINT_I(a);
        PRINT_F(x,y);
}
```

```
void PRINT()
{
        printf("Hello Everybody!!!\n");
}
```

```
void PRINT_I(int b)
{
        printf("%d \n",b);
}
```

```
void PRINT_F(float p, float q)
{
        printf("%2.1f %2.2f \n",p,q);
}
```

Command Line Arguments

- Command line arguments may be passed by specifying them under main( ).

```
int main(int argc, char *argv[ ]);
```

Argument Count

Array of Strings as command line arguments including the command itself.

Passing parameters to main()

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
	int i;

	for(i=0;i<argc;i++) {
		printf("%d: %s\n",i,argv[i]);
	}

	return 0;
}
```

```
$ cc -Wall wk13_commandline.c
$ ./a.out
0: ./a.out
$ ./a.out HI
0: ./a.out
1: HI
$ ./a.out Indian Institute of Technology
0: ./a.out
1: Indian
2: Institute
3: of
4: Technology
```

Passing parameters to main()

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
	int i;

	for(i=0;i<argc;i++) {
		printf("%d: %s\n",i,argv[i]);
	}

	return 0;
}
```

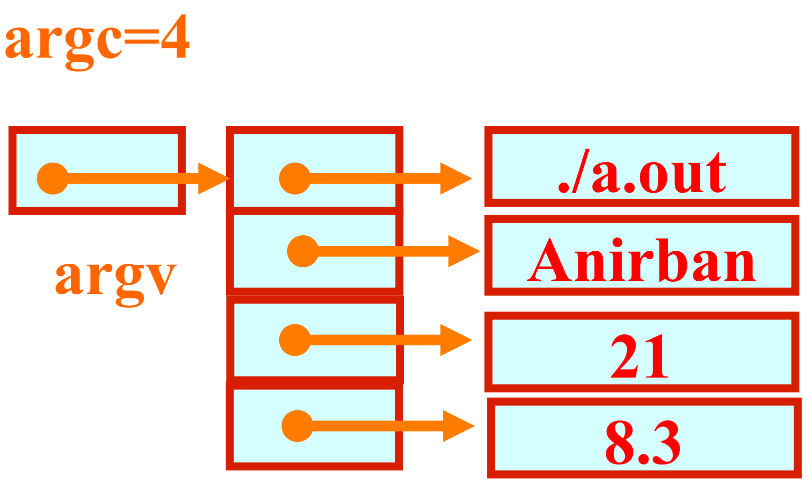
```
$ time ./a.out Indian Institute of Technology
0: ./a.out
1: Indian
2: Institute
3: of
4: Technology

real	0m0.002s
user	0m0.001s
sys	0m0.001s
```

Passing parameters to main()

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        int i;

        for(i=0;i<argc;i++) {
                printf("%d: %s\n",i,argv[i]);
        }

        return 0;
}
```

```
$ ./a.out Anirban 21 8.3
0: ./a.out
1: Anirban
2: 21
3: 8.3
```

./a.out Anirban 21 8.3



Library function sscanf()

- Header file:
  - #include <stdio.h>

- Function prototype:
  - int sscanf(const char *str, const char *format, ...);

- Conversion character is same as scanf().

Passing parameters to main()

```
#include <stdio.h>
int main(int argc, char *argv[])
{
        int i;
        for(i=0;i<argc;i++) {
                printf("%d: %s\n",i,argv[i]);
        }
        return 0;
}
```

```
$ ./a.out Anirban 21 8.3
0: ./a.out
1: Anirban
2: 21
3: 8.3
$
```

```
#include <stdio.h>
int main(int argc, char *argv[])
{
        char name[20];
        int age;
        float cgpa;

        sscanf(argv[1],"%s",name);
        sscanf(argv[2],"%d",&age);
        sscanf(argv[3],"%f",&cgpa);

        printf("%s  %d   %f\n",
                name,age,cgpa);
        return 0;
}
```

```
$ ./a.out Anirban 21 8.3
Anirban  21   8.300000
$
```

Functions to convert strings to numbers

- Once we've got a string with a number in it (either from a file or from the user typing) we can use atoi or atof to convert it to a number
- The functions are part of header file `stdlib.h`

```
char numberstring[]= "3.14";
int i;
double pi;
pi= atof (numberstring);
i= atoi ("12");
```

Both of these functions return 0 if they have a problem

Example: Average from Command Line

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc,char *argv[])
{
   float sum=0;
   int    i,num;

   num=argc-1;
   for(i=1;i<=num;i++)
     sum+=atof(argv[i]);
   printf("Average=%f \n",sum/(float) num);
   return 0;
}
```

```
$ ./a.out  45  239  123
Average=135.666667
$
```

Passing parameters to main()

- Two parameters will be passed to function main() through command line – argc and argv.
- Name of the parameters are fixed.
- argc is of integer type and it stores the number of parameters (delimited by space) in the command line.
- argv is a 2D array of characters and it stores all the words in the command line.
- By default all the parameters are taken as array of characters (strings) that can be converted to other data types.

File Handling

Advantages of File handling

- At times size of the program input is very large.
- During the testing phase providing inputs in the interactive way is tedious.
- The size of the program output may be large enough and will not fit in a single screen.
- You may wish to store the output for future analysis.

File handling in C

- A file needs to be opened first for any input/output operations on the file.
  - It may be opened for reading/writing/appending.

- The file must be closed once the use/handling of the file is over.

- In between the address of the file will be stored in a pointer data type viz., FILE *.

File handling in C

- In C we use **FILE *** to represent a pointer to a file.

- **fopen** is used to open a file. It returns a pointer to the file if successfully opened the file else it returns **NULL**.

```
FILE *fptr;
char filename[]= "file2.dat";
fptr= fopen (filename,"w");
if (fptr == NULL) {
  printf (“ERROR IN FILE CREATION”);
     /* DO SOMETHING */
}
```

exit() function

- Sometimes error checking means we want an "emergency exit" from a program. We want it to stop dead.
- In main we can use "return" to stop.
- In functions we can use exit to do this.
- Library file stdlib.h is the header file for exit() function.

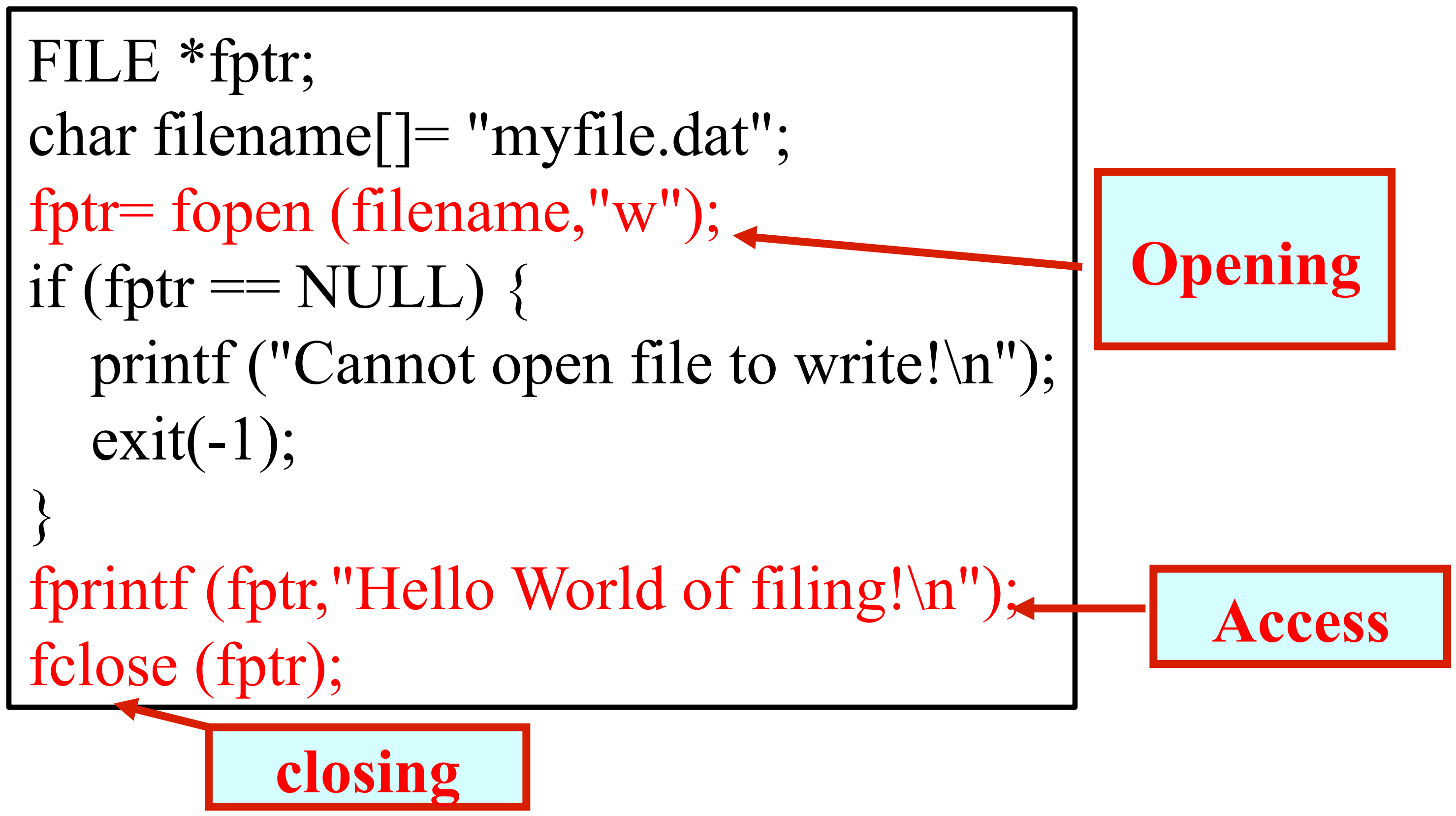
```
FILE *fptr;
char filename[]= "file2.dat";
fptr= fopen (filename,"w");
if (fptr == NULL) {
  printf (“ERROR IN FILE CREATION”);
  exit(-1);
}
```

Modes for opening files

- The second argument of fopen is the *mode* in which we open the file.
- "r" opens a file for reading.
- "w" creates a file for writing - and writes over all previous contents (deletes the file so be careful!).
- "a" opens a file for appending - writing on the end of the file.

Closing a file

- We can close a file simply using fclose() and the file pointer.



```
FILE *fptr;
char filename[]= "myfile.dat";
fptr= fopen (filename,"w");
if (fptr == NULL) {
    printf ("Cannot open file to write!\n");
    exit(-1);
}
fprintf (fptr,"Hello World of filing!\n");
fclose (fptr);
```

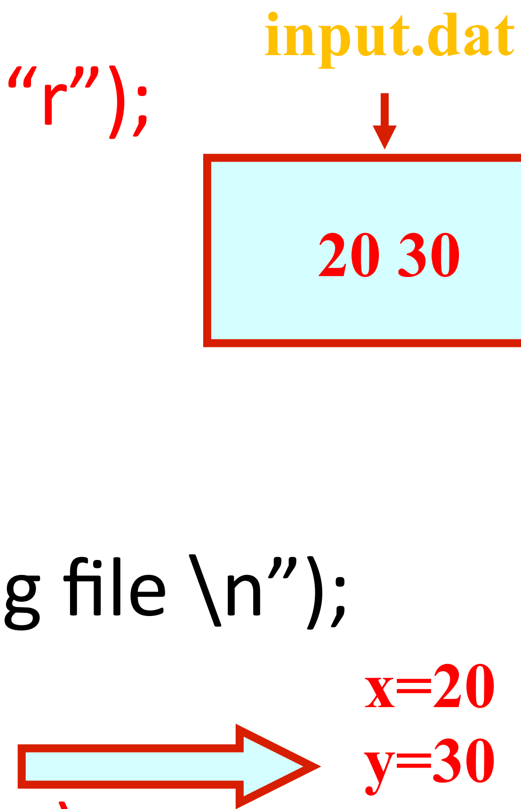
Writing to a file using fprintf( )

- fprintf( ) works just like printf and sprintf except that its first argument is a file pointer.

```
FILE *fptr;
fptr= fopen ("file.dat","w");
/* Check it's open */
fprintf (fptr,"Hello World!\n");
```

Reading Data Using fscanf( )

```
FILE *fptr;
fptr= fopen (“input.dat”,“r”);
/* Check it's open */
if (fptr==NULL)
  {
    printf(“Error in opening file \n”);
   }
fscanf(fptr,“%d%d”,&x,&y);
```



Reading lines from a file using

We can read a string using fgets().

```
FILE *fptr;
char line [1000];
/* Open file and check it is open */
while (fgets(line,1000,fptr) != NULL) {
   printf ("Read line %s\n",line);
}
```

fgets() takes 3 arguments, a string, a maximum number of characters to read and a file pointer.
It returns NULL if there is an error (such as EOF).

Check whether a matrix is symmetric or not – V1

```
#include <stdio.h>

int ReadMat(int mat[1000][1000])
{
        int i,j,size;

        printf(“Matrix size? : \n");
        scanf("%d",&size);
        for(i=0;i<size;i++)
                for(j=0;j<size;j++)
                        scanf("%d",&mat[i][j]);
        return size;
}
```

```
int main()
{
        char symm='y';
        int i,j,size,mat[1000][1000];

        size=ReadMat(mat);
        for(i=0;i<size;i++) {
                for(j=i+1;j<size;j++) {
                        if(mat[i][j]!=mat[j][i])
                                symm='n';
                                break;
                }
                if(symm=='n')  break;
        }
        if(symm=='y')   printf("Symmetrix Matrix\n");
         else  printf("Not Symmetric");
        return 0;
}
```

Check whether a matrix is symmetric or not – V2

```
#include <stdio.h>
int ReadMat(int mat[1000][1000])
{
        int i,j,size;
        FILE *fp;

        fp=fopen("Matrix.txt","r");
//      printf("Matrix size: \n");
        fscanf(fp,"%d",&size);
        for(i=0;i<size;i++)
                for(j=0;j<size;j++)
                  fscanf(fp,"%d",&mat[i][j]);

        fclose(fp);
        return size;
}
```

```
int main()
{
        char symm='y';
        int i,j,size,mat[1000][1000];

        size=ReadMat(mat);
        for(i=0;i<size;i++) {
                for(j=i+1;j<size;j++) {
                        if(mat[i][j]!=mat[j][i])
                                symm='n';
                                break;
                }
                if(symm=='n')  break;
        }
        if(symm=='y')   printf("Symmetrix Matrix\n");
         else  printf("Not Symmetric");
        return 0;
}
```

Check whether a matrix is symmetric or not – V3

```
#include <stdio.h>
int ReadMat(int mat[1000][1000])
{
        char filename[100];
        int i,j,size;
        FILE *fp;
        scanf("%s",filename);
        fp=fopen(filename,"r");
//      printf("Matrix size: \n");
        fscanf(fp,"%d",&size);
        for(i=0;i<size;i++)
                for(j=0;j<size;j++)
                  fscanf(fp,"%d",&mat[i][j]);

        fclose(fp);
        return size;
}
```

```
int main()
{
        char symm='y';
        int i,j,size,mat[1000][1000];

        size=ReadMat(mat);
        for(i=0;i<size;i++) {
                for(j=i+1;j<size;j++) {
                        if(mat[i][j]!=mat[j][i])
                                symm='n';
                                break;
                }
                if(symm=='n')  break;
        }
        if(symm=='y')   printf("Symmetrix Matrix\n");
         else  printf("Not Symmetric");
        return 0;
}
```

Check whether a matrix is symmetric or not – V4

```
#include <stdio.h>
int ReadMat(char filename, int
mat[1000][1000])
{
    int i,j,size;
    FILE *fp;

    fp=fopen(filename,"r");
//  printf("Matrix size: \n");
    fscanf(fp,"%d",&size);
    for(i=0;i<size;i++)
        for(j=0;j<size;j++)
          fscanf(fp,"%d",&mat[i][j]);

    fclose(fp);
    return size;
}
```

```
int main(int argc, char *argv[])
{
    char symm='y';
    int i,j,size,mat[1000][1000];

    size=ReadMat(argv[1],mat);
    for(i=0;i<size;i++) {
        for(j=i+1;j<size;j++) {
            if(mat[i][j]!=mat[j][i])
                symm='n';
                break;
        }
        if(symm=='n')  break;
    }
    if(symm=='y')   printf("Symmetrix Matrix\n");
     else  printf("Not Symmetric");
    return 0;
}
```

Balanced Symbol Checking

In processing programs and working with computer languages there are many instances when symbols must be balanced { } , [ ] , ( )

Write a program that will take a C program file as command line input and prints whether all the parenthesis, curly and square brackets that are opened has closed or not.

Balanced Symbol Checking

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    FILE *fp;
    char line[500];
    int i,pbracket,cbracket,sbracket;
    fp=fopen(argv[1],"r"):
    if(fp==NULL) {
        printf("File opening error... exiting");
        exit(0);
    }
    pbracket=cbracket=sbracket=0;
    while(!feof(fp)) {
        fgets(line,100,fp);
        i=0;
```

Did file open properly?

File end is not reached.

```
while(line[i]!='\0') {
            switch(line[i]) {
                case '(': pbracket++; break;
                case ')': pbracket--; break;
                case '{': cbracket++; break;
                case '}': cbracket--; break;
                case '[': sbracket++; break;
                case ']': sbracket--; break;
            } i++;
        }
    }
    fclose(fp);
    if(pbracket==0) printf("Parentesis Open-Close.\n");
    else printf("Parentesis Mismatches.\n");
    if(cbracket==0) printf("Curly Open-Close.\n");
    else printf("Curly Mismatches.\n");
    if(sbracket==0) printf("Square Open-Close.\n");
    else printf("Square Mismatches.\n");
    return 0;
}
```

Balanced Symbol Checking

In processing programs and working with computer languages there are many instances when symbols must be balanced { } , [ ] , ( )

Following C syntax:

- { } [] () …. is allowed
- { [}] ( ) … is not allowed.

Algorithm: Balanced Symbol Checking

Homework

Write a C program that will take any C program as command line input and will report any syntax error due to balancing in bracket symbols.

- Make an empty stack
- Read characters until end of file
  - If a symbol is an opening symbol push it onto the stack
  - If a symbol is a closing symbol pop the stack
    - if the stack is empty report an error
    - if the popped symbol does not match the closing symbol report an error
- If the stack is empty report symbols are balanced.
- Else report an error

Three special I/O streams

- Three special file streams are defined in the <stdio.h> header
- *stdin* reads input from the keyboard
- *stdout* send output to the screen
- *stderr* prints errors to an error device (usually also the screen)
- What might this do?

```
fprintf (stdout,"Hello World!\n");
```

An example program

```c
#include <stdio.h>
int main()
{
	int i;

	fprintf(stdout,"Give value of i \n");
	fscanf(stdin,"%d",&i);
	fprintf(stdout,"Value of i=%d \n",i);
	fprintf(stderr,"No error: But an example to show error message.\n");
	return 0;
 }
```

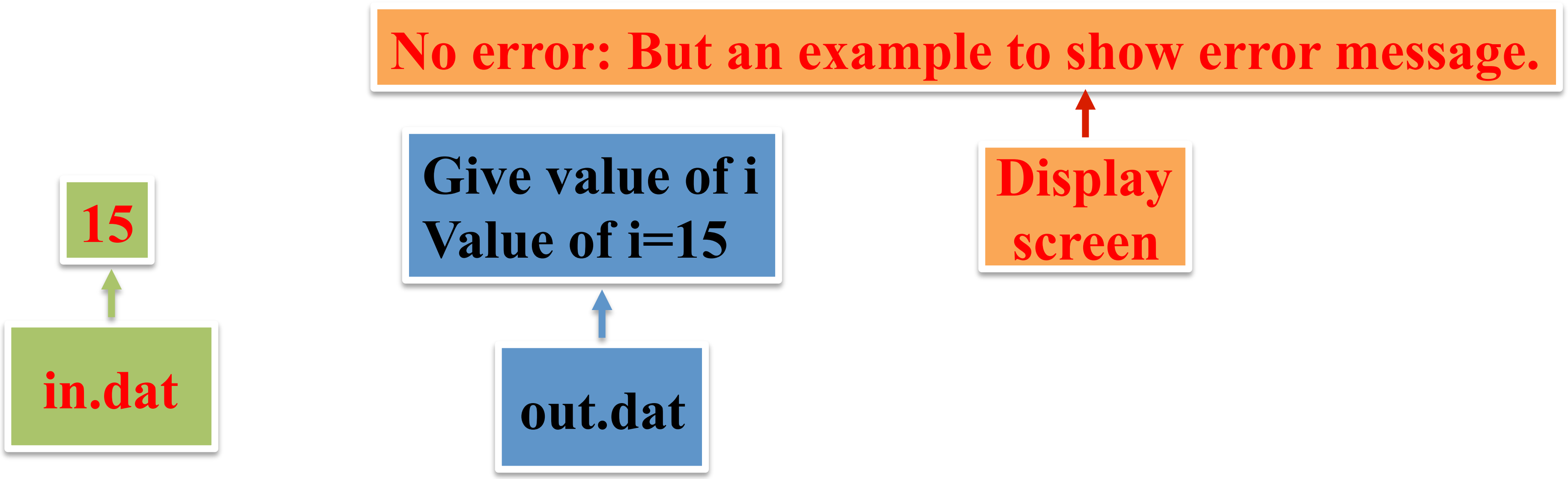
```
$ ./a.out
Give value of i
15
Value of i=15
No error: But an example to show error message.
$
```

Display on the screen

Input File & Output File redirection

- One may redirect the input and output files to other files (other than stdin and stdout).
- Usage: Suppose the executable file is a.out

```
$ ./a.out  <in.dat   >out.dat
```



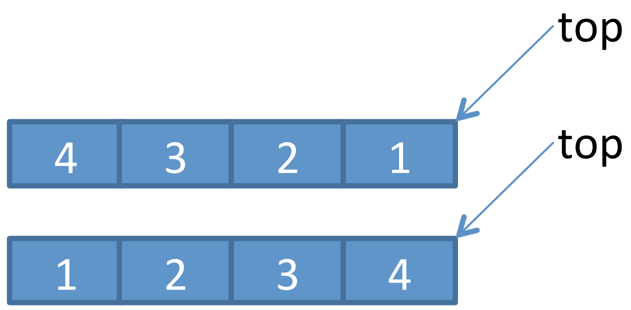
```
Give value of i
Value of i=15
```

Example 1:Reverse a stack using recursion

You are not allowed to use loop constructs like while, for.. etc

Sample input: 1 2 3 4

Sample output: 4 3 2 1

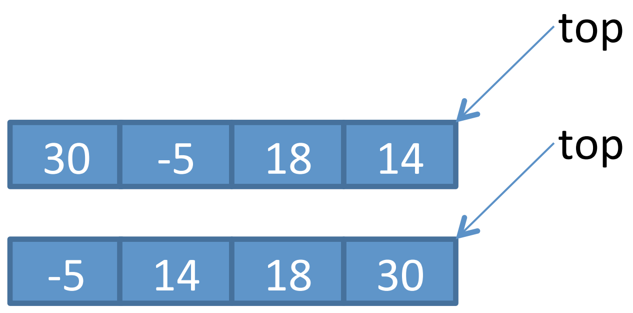


Example 2: Sort a stack using recursion

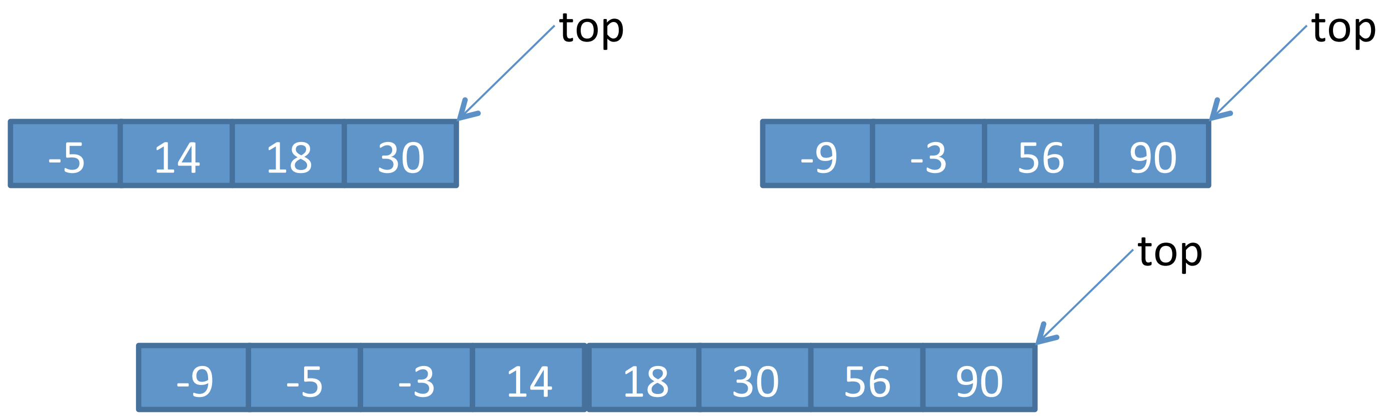
You are not allowed to use loop constructs like while, for.. etc

Sample input: **14 18 -5 30**

Sample output: **30 18 14 -5**



Example 3: Implementation of Merging operation in Merge Sort using Stacks



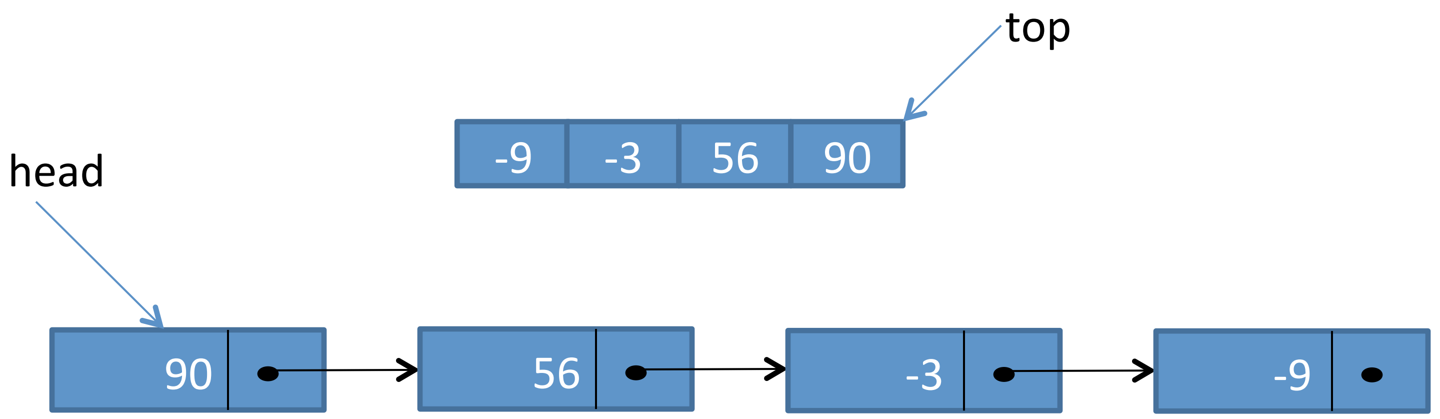
Example 4: Given a stack, print the Next Greater Element (NGE) for every element.

- For any stack the bottom most element always has next greater element as -1
- For a stack which is sorted in decreasing order, all elements have next greater element as -1.
- For the input stack [4, 5, 2, 25], the next greater elements for each element are as follows.

- Element NGE

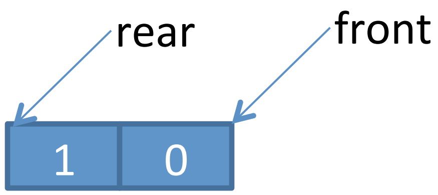
Element	NGE
4	5
5	25
2	25
25	-1

Example 5: Implement a Stack using a Linked List

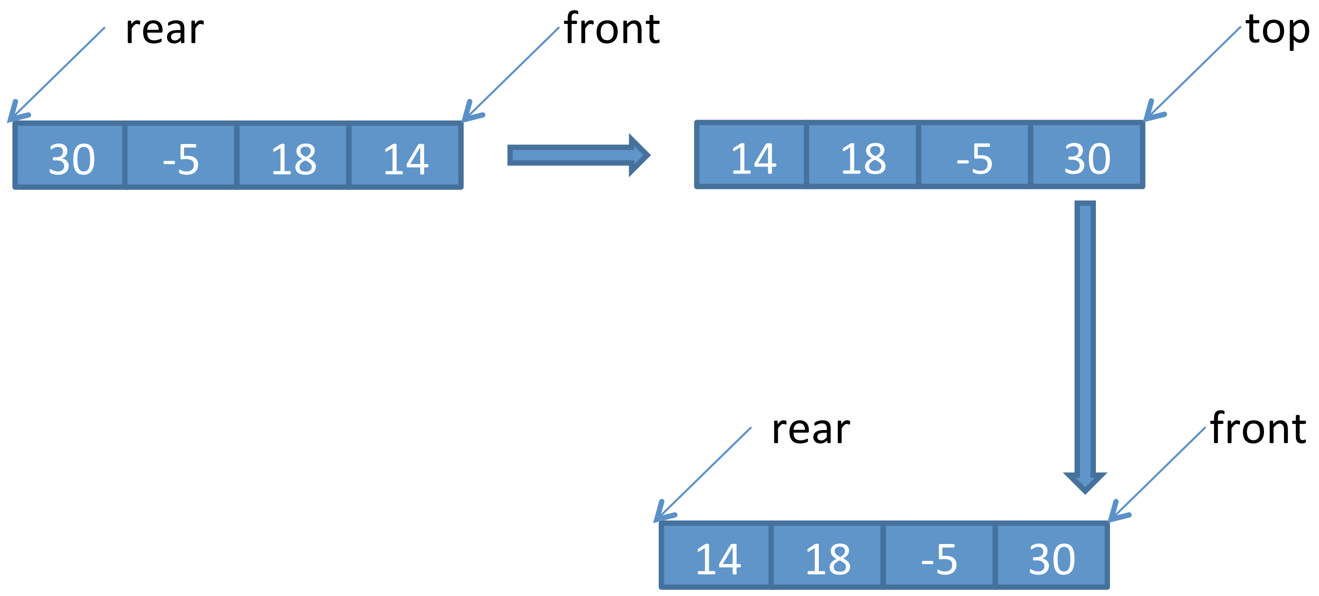


Example 6: Print first *N* Fibonacci Numbers using a Queue

The queue initially contains 0 and 1

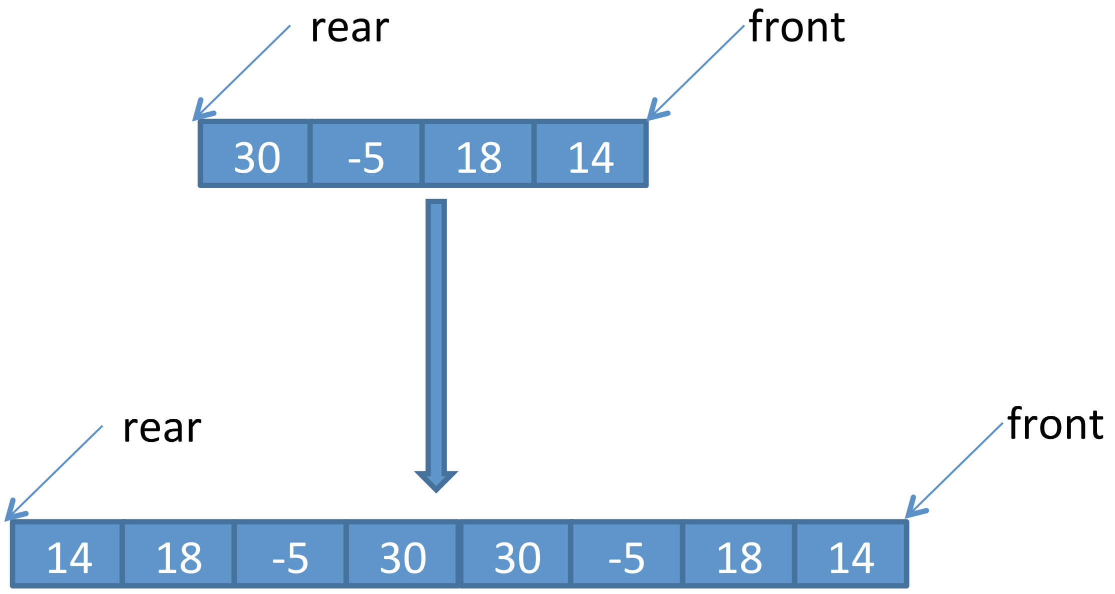


Example 7: Use a Stack to reverse a Queue



Example 8: Create a new Queue with given elements appended at the end of the Queue in a reverse order

* Hint- You can use a stack in order to achieve the outcome



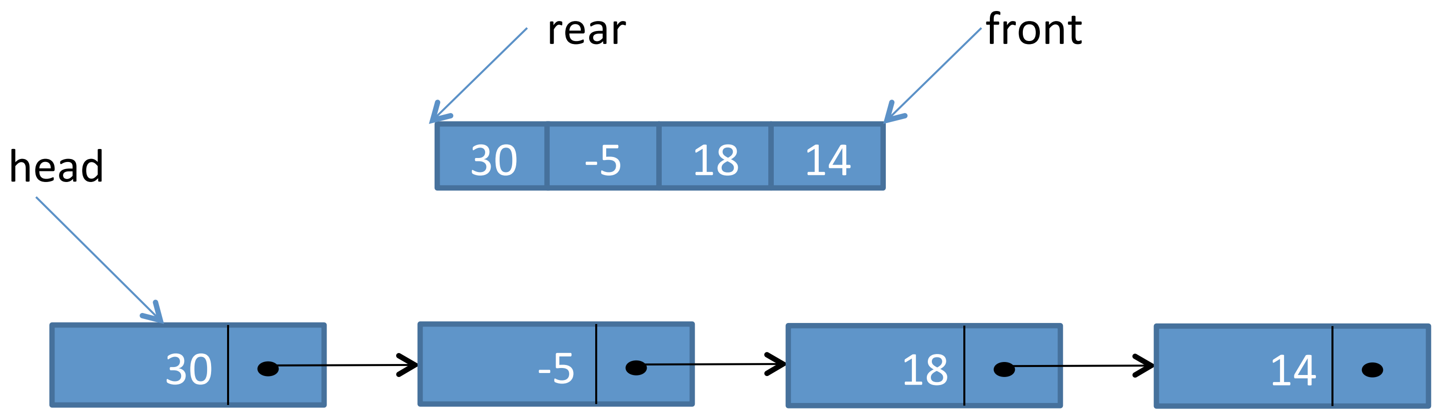
Example 9: Implement a Stack using a Queue data structure

For a given stack create a same size array which you are going to use as a Queue.

Push and pop operation of stack’s should be emulated with the Enqueue and Dequeue operation.

You can use an intermediate Queue for the above implementation.

Example 10: Implement a Queue using a Linked List


rear
front
30
-5
18
14
head
30
-5
18
14