SYSTEM PROGRAMMING LAB

LAB PRACTICALS RECORD

(CSX - 326)

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR – 144011, PUNJAB (INDIA)

Submitted To:

Ms. Rupali

Asst. Professor

Department of CSE

Submitted By:

Aman Garg

13103050

6th Semester

INDEX

PROGRAM -1

SEARCHING- LINEAR | BINARY

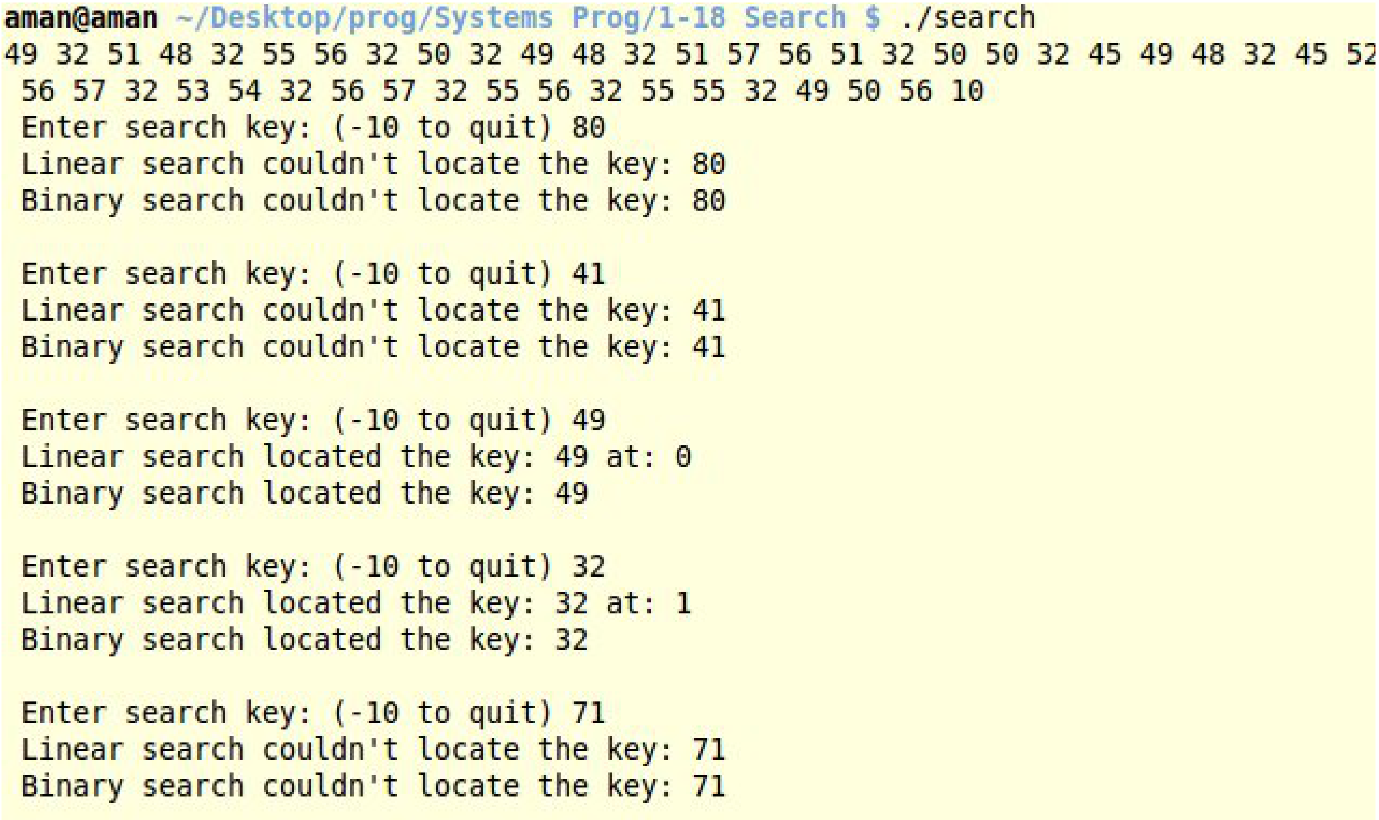
Description:

The binary search algorithm begins by comparing the target value to the value of the middle element of the sorted array. If the target value is equal to the middle element's value, then the position is returned and the search is finished. If the target value is less than the middle element's value, then the search continues on the lower half of the array; or if the target value is greater than the middle element's value, then the search continues on the upper half of the array. This process continues, eliminating half of the elements, and comparing the target value to the value of the middle element of the remaining elements - until the target value is either found (and its associated element position is returned), or until the entire array has been searched (and "not found" is returned).

Program:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

void linearSearch(vector<int> & input, const int & key){
   // Given a key and an array, it linearly  searches for the input key
   vector<int> :: iterator it = find(input.begin(), input.end(), key);
   if (it == input.end())  //we have reached end of the iterator
      cout << " Linear search couldn't locate the key: "<< key << endl;
   else
      cout << " Linear search located the key: "<< key <<" at: "<<int(it - input.begin())<<endl;
}


void binarySearch(const vector<int> & input, const int & key){
   // Given a key and a sorted array, it searches for the input key by dividing into intervals
   vector<int> newInput (input);
   sort(newInput.begin(), newInput.end());

   int l = 0, r = newInput.size() - 1, mid;
   bool foundState = false;

   while (l <= r){
      mid = (l + r)/2 ;

      if (newInput[mid] == key){
         foundState = true;
         break;
      }
```

```
        else if (newInput [mid] > key) //Key < [mid]. So, move to left interval
            r = mid - 1;
        else                //Key > [mid]. So, move to right interval
            l = mid + 1;
    }

    if (foundState == true)
        cout << " Binary search located the key: "<< key << endl;
    else
        cout << " Binary search couldn't locate the key: "<< key << endl;
}

int main(){

    // Read test data
    ifstream inf("testFile");
    if (!inf){

        fprintf(stderr,"\nError opening test file\n");
        return -1;
    }

    vector<int> input;
    int searchKey;
    char c;

    while( (c = inf.get()) != EOF)
                        input.push_back(int(c));

    inf.close();

    for (auto elem : input)
        cout << elem <<" ";

    while (true){
        cout <<"\n Enter search key: (-10 to quit) ";
        cin >> searchKey;
        if (searchKey == -10) break;

        linearSearch(input, searchKey);
        binarySearch(input, searchKey);
    }

    return 0;
}
```



```
aman@aman ~/Desktop/prog/Systems Prog/1-18 Search $ ./search
49 32 51 48 32 55 56 32 50 32 49 48 32 51 57 56 51 32 50 50 32 45 49 48 32 45 52
 56 57 32 53 54 32 56 57 32 55 56 32 55 55 32 49 50 56 10
 Enter search key: (-10 to quit) 80
 Linear search couldn't locate the key: 80
 Binary search couldn't locate the key: 80

 Enter search key: (-10 to quit) 41
 Linear search couldn't locate the key: 41
 Binary search couldn't locate the key: 41

 Enter search key: (-10 to quit) 49
 Linear search located the key: 49 at: 0
 Binary search located the key: 49

 Enter search key: (-10 to quit) 32
 Linear search located the key: 32 at: 1
 Binary search located the key: 32

 Enter search key: (-10 to quit) 71
 Linear search couldn't locate the key: 71
 Binary search couldn't locate the key: 71
```

PROGRAM - 2

SORTING ALGORITHMS

Description:

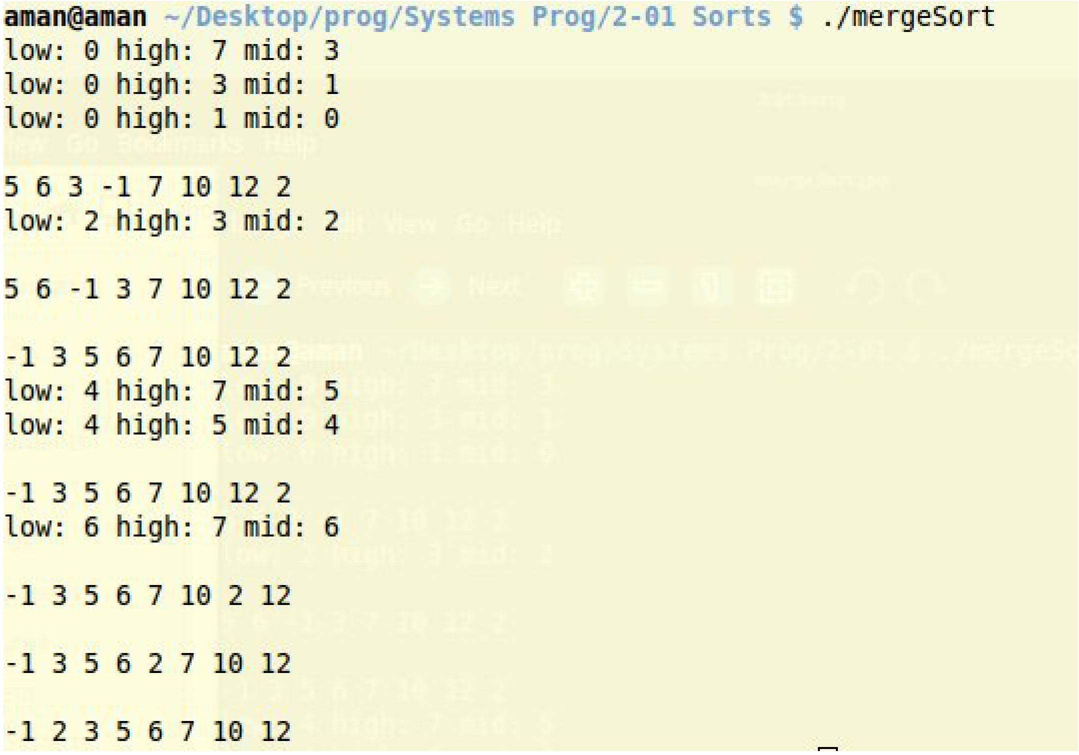
Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires O(N) extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. Comparing average complexity we find that both type of sorts have O(NlogN) average complexity but the constants differ. For arrays, merge sort loses due to the use of extra O(N) storage space.

Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of O(nLogn). The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.

Program (MERGESORT) :

```
#include <bits/stdc++.h>
using namespace std;

void merge(vector<int> & arr, int low, int mid, int high){
   int i = low, j = mid +1, k = low;
   vector<int> c(100);

   while (i <= mid && j <= high){
      if (arr[i] < arr[j]){
         c[k] = arr[i];
         k ++;i ++;
      }
      else{
         c[k] = arr[j];
         k ++, j ++;
      }
   }
   while (i <= mid){
      c[k] = arr[i];
      k ++;i ++;
   }
   while (j <= high){
      c[k] = arr[j];
      k++, j ++;
   }
   for(i = low; i < k; i++)
      arr[i] = c[i];
```

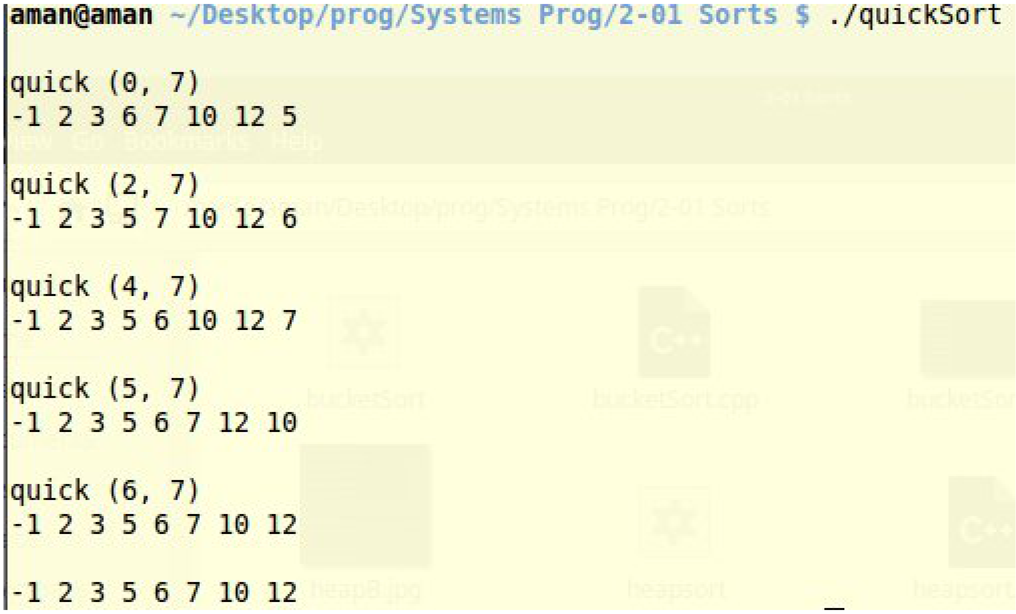
```
    std::cout <<  endl;
    for(auto a: arr)
        cout << a <<" ";
    cout << endl;
}
void mergeSort(vector<int> &arr, int low, int high){

    if (low < high){
        int mid = (low +high) /2;
        printf("low: %d high: %d mid: %d\n",low, high, mid );

        mergeSort(arr, low, mid);
        mergeSort(arr, mid +1, high);
        merge(arr, low, mid, high);
    }
    return;
}

int main(){

    vector<int> arr {6, 5, 3, -1, 7, 10, 12, 2};
    mergeSort(arr, 0, arr.size() -1);
    return 0;
}
```



```
aman@aman ~/Desktop/prog/Systems Prog/2-01 Sorts $ ./mergeSort
low: 0 high: 7 mid: 3
low: 0 high: 3 mid: 1
low: 0 high: 1 mid: 0

5 6 3 -1 7 10 12 2
low: 2 high: 3 mid: 2

5 6 -1 3 7 10 12 2

-1 3 5 6 7 10 12 2
low: 4 high: 7 mid: 5
low: 4 high: 5 mid: 4

-1 3 5 6 7 10 12 2
low: 6 high: 7 mid: 6

-1 3 5 6 7 10 2 12

-1 3 5 6 2 7 10 12

-1 2 3 5 6 7 10 12
```

Program (QUICKSORT) :

```
#include <bits/stdc++.h>
using namespace std;

int partition(vector<int> &A, int low, int high){
    int pivot = A[high];
    int pivotIndex = low;

    for (int i = low; i < high; i++){
        if (A[i] <= pivot){
            swap(A[i], A[pivotIndex]);
            pivotIndex ++;
        }
    }
    swap(A[high], A[pivotIndex]);
    for (auto a : A)
    cout << a <<" ";
    cout << endl <<endl;
    return pivotIndex;
}


void quickSort(vector<int> &A, int low, int high){
    if (low < high){
        printf("quick (%d, %d)\n",low, high);
        int pivotIndex = partition(A, low, high);
        quickSort(A, low, pivotIndex -1);
        quickSort(A, pivotIndex +1, high);
    }
}


int main(){

    vector<int> A {6, 5, 3, -1, 7, 10, 12, 2};

    cout << endl;
    quickSort(A, 0, A.size() -1);
    for (auto a : A)
    cout << a <<" ";
    cout << endl;
    return 0;
}
```



```
aman@aman ~/Desktop/prog/Systems Prog/2-01 Sorts $ ./quickSort

quick (0, 7)
-1 2 3 6 7 10 12 5

quick (2, 7)
-1 2 3 5 7 10 12 6

quick (4, 7)
-1 2 3 5 6 10 12 7

quick (5, 7)
-1 2 3 5 6 7 12 10

quick (6, 7)
-1 2 3 5 6 7 10 12

-1 2 3 5 6 7 10 12
```

BUCKETSORT

Bucket sort, or bin sort, is a sorting algorithm that works by distributing the elements of an array into a number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm. It is a distribution sort, and is a cousin of radix sortin the most to least significant digit flavour. Bucket sort is a generalization of pigeonhole sort. Bucket sort can be implemented with comparisons and therefore can also be considered a comparison sort algorithm. The computational complexity estimates involve the number of buckets. Bucket sort works as follows:

1.Set up an array of initially empty "buckets".

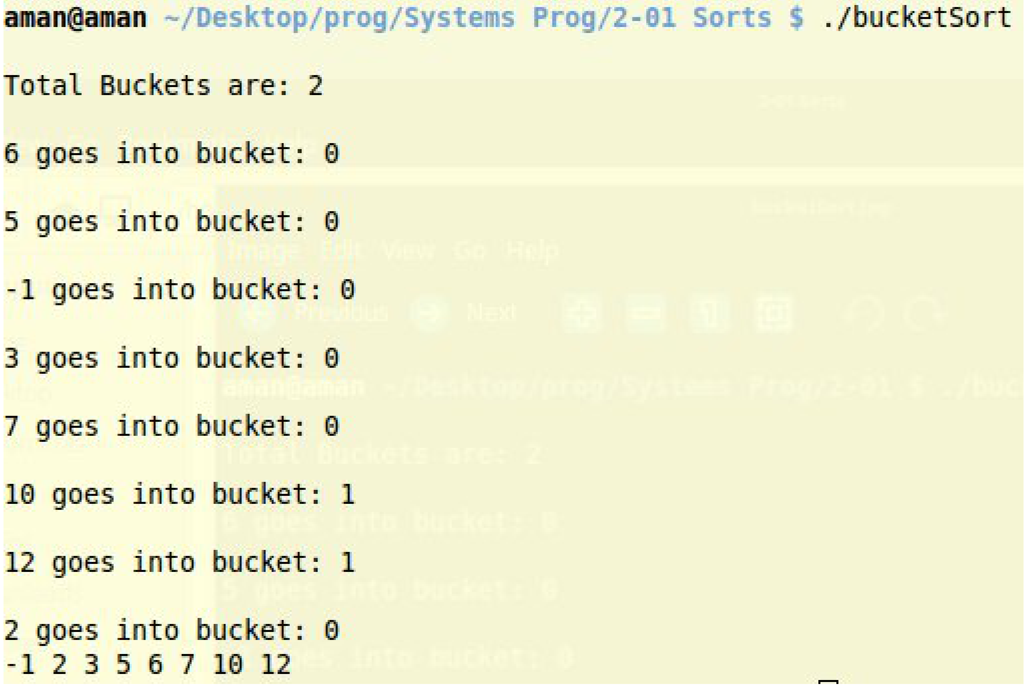
2.Scatter: Go over the original array, putting each object in its bucket.

3.Sort each non-empty bucket.

4.Gather: Visit the buckets in order and put all elements back into the original array.

Program (BUCKETSORT) :

```
#include <bits/stdc++.h>
using namespace std;

void bucketSort(vector<int> &A){
    int n = A.size();
    int minm = (*min_element(A.begin(), A.end()) / 10) * 10;
    int maxm = (*max_element(A.begin(), A.end()) / 10) * 10 + 10;
    int rangeM = (maxm - minm)/10;
    vector<vector<int> > buckets(rangeM);
    for(int i = 0; i < n; i++){
        int c = (A[i] / 10);
        buckets[c].push_back(A[i]);
    }

    for (int i = 0; i < buckets.size(); i++)
        sort(buckets[i].begin(), buckets[i].end());

    int index = 0;
    for (int i = 0; i < buckets.size(); i++){
        for(int j = 0; j < buckets[i].size(); j++)
            A[index++] = buckets[i][j];
    }
}

int main(){
    vector<int> A {6, 5, -1, 3, 7, 10, 12, 2};
    bucketSort(A);
    for(auto a : A)
        cout << a << " ";
    cout << endl;
    return 0;
}
```



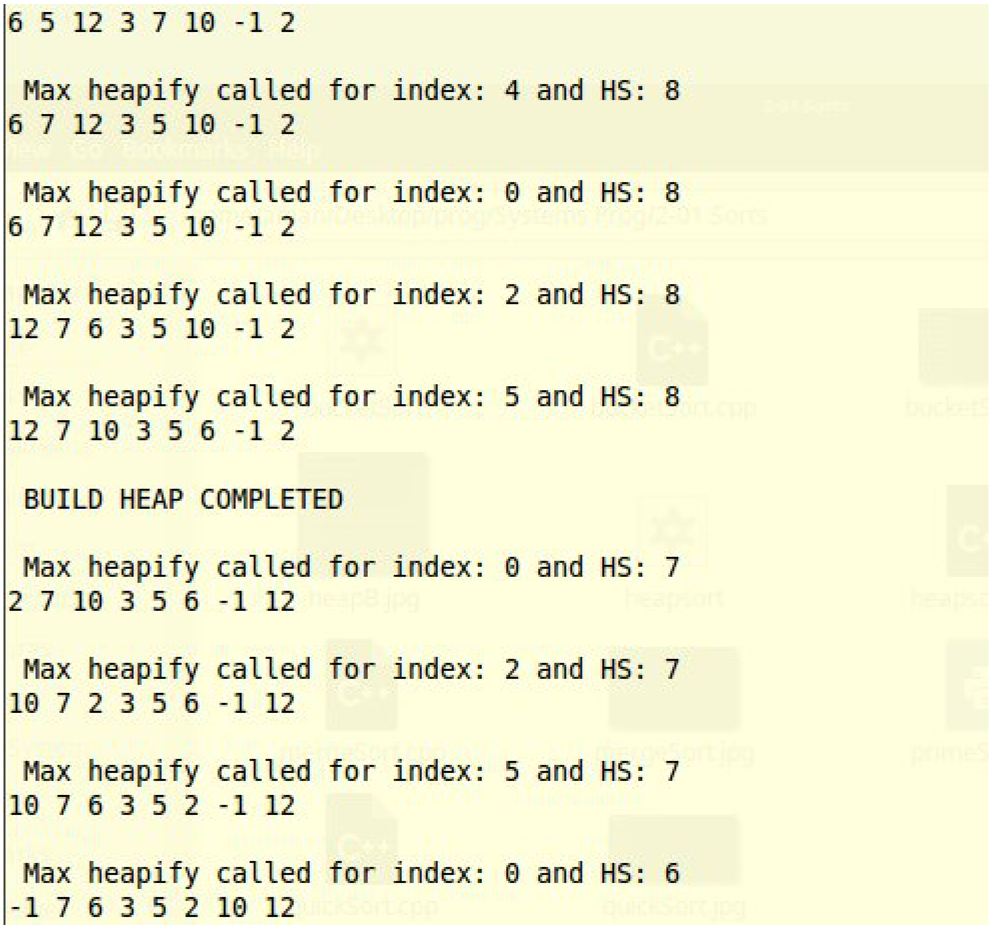
```
aman@aman ~/Desktop/prog/Systems Prog/2-01 Sorts $ ./bucketSort

Total Buckets are: 2

6 goes into bucket: 0

5 goes into bucket: 0

-1 goes into bucket: 0

3 goes into bucket: 0

7 goes into bucket: 0

10 goes into bucket: 1

12 goes into bucket: 1

2 goes into bucket: 0
-1 2 3 5 6 7 10 12
```

HEAPSORT

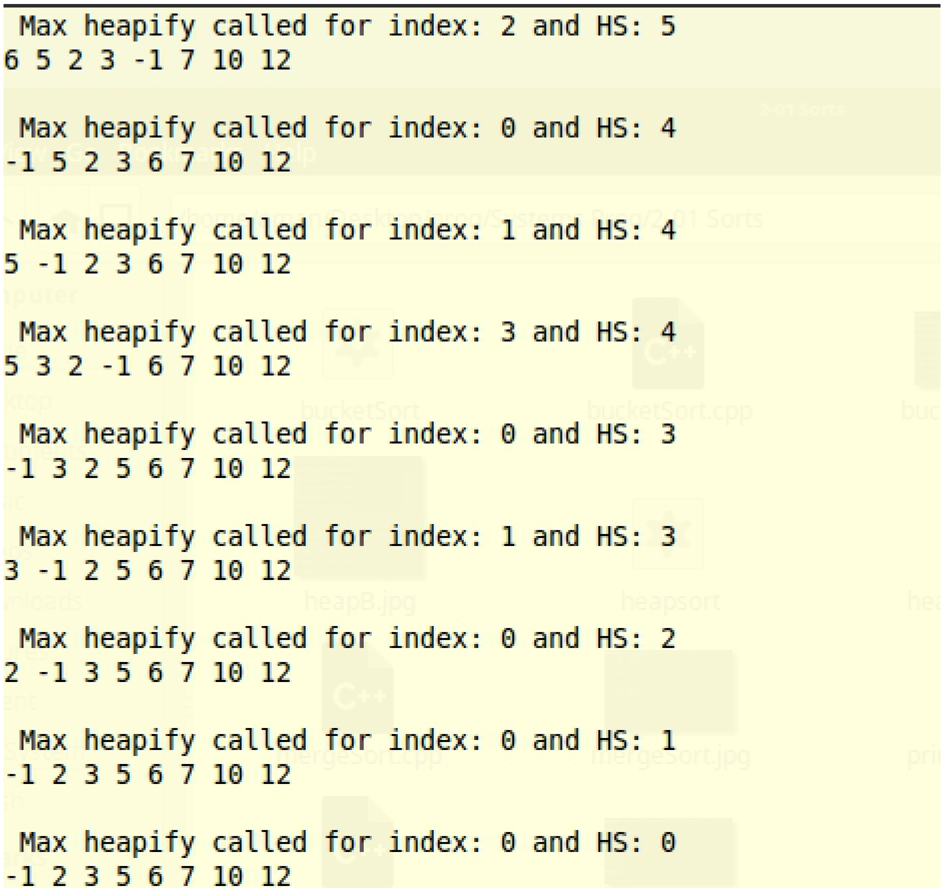
Heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the maximum. Although somewhat slower in practice on most machines than a well-implemented quicksort, it has the advantage of a more favorable worst-case O(n log n) runtime. Heapsort is an in-place algorithm, but it is not a stable sort.

Program (HEAPSORT) :

```
#include <bits/stdc++.h>
using namespace std;

int temp;
int left(int i){
    // get left child 2i + 1
     return 2*i +1;
}
int right(int i){
    // get right child 2*i +2
     return 2*i + 2;
}
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void maxHeapify(vector<int> & arr, int i, int & heapSize){
    cout <<"\n Max heapify called for index: "<<i<<" and HS: "<<heapSize<<endl;
    for (auto a : arr)
        cout << a << " " ;
    cout << endl;
    int l = left(i), r = right(i);
    int largest = i;

    if (l < heapSize && arr[l] > arr[i])
        largest = l;
    if (r < heapSize && arr[r] > arr[largest])
        largest = r;
    if (largest == i)   //we are fine. no heapify needed
        return;
    // else swap arr[i] with arr[largest]
    swap(&arr[largest], &arr[i]);
    maxHeapify(arr, largest, heapSize); }

void buildHeap(vector<int> & arr, int & heapSize){
    int mid = (arr.size() -1) /2;
    for (int i = mid; i >= 0; i --)
        maxHeapify(arr, i, heapSize );
    cout <<"\n BUILD HEAP COMPLETED\n";
}

void heapSort(vector<int> &arr){
    int heapSize = arr.size();
    buildHeap(arr, heapSize);
    for (int i = arr.size() -1; i >= 0; i--){
        swap(&arr[0], &arr[i]);
        heapSize -= 1;
        maxHeapify(arr, 0, heapSize);
    }
}

int main(){
    vector<int> arr {6, 5, -1, 3, 7, 10, 12, 2};
    heapSort(arr);
    return 0;
}
```



```
6 5 12 3 7 10 -1 2

 Max heapify called for index: 4 and HS: 8
6 7 12 3 5 10 -1 2

 Max heapify called for index: 0 and HS: 8
6 7 12 3 5 10 -1 2

 Max heapify called for index: 2 and HS: 8
12 7 6 3 5 10 -1 2

 Max heapify called for index: 5 and HS: 8
12 7 10 3 5 6 -1 2

 BUILD HEAP COMPLETED

 Max heapify called for index: 0 and HS: 7
2 7 10 3 5 6 -1 12

 Max heapify called for index: 2 and HS: 7
10 7 2 3 5 6 -1 12

 Max heapify called for index: 5 and HS: 7
10 7 6 3 5 2 -1 12

 Max heapify called for index: 0 and HS: 6
-1 7 6 3 5 2 10 12
```



```
 Max heapify called for index: 2 and HS: 5
6 5 2 3 -1 7 10 12

 Max heapify called for index: 0 and HS: 4
-1 5 2 3 6 7 10 12

 Max heapify called for index: 1 and HS: 4
5 -1 2 3 6 7 10 12

 Max heapify called for index: 3 and HS: 4
5 3 2 -1 6 7 10 12

 Max heapify called for index: 0 and HS: 3
-1 3 2 5 6 7 10 12

 Max heapify called for index: 1 and HS: 3
3 -1 2 5 6 7 10 12

 Max heapify called for index: 0 and HS: 2
2 -1 3 5 6 7 10 12

 Max heapify called for index: 0 and HS: 1
-1 2 3 5 6 7 10 12

 Max heapify called for index: 0 and HS: 0
-1 2 3 5 6 7 10 12
```

PROGRAM – 3

TWO PASS ASSEMBLER

Description:

Heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like

Program (Python) :

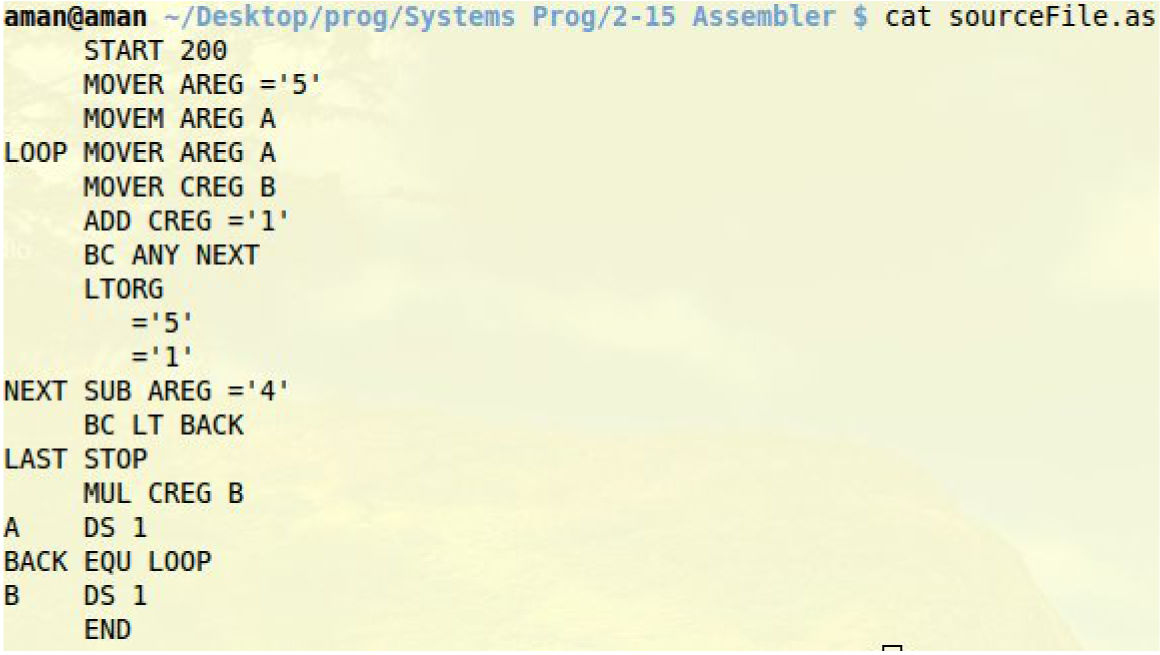
```
from __future__ import print_function
import re

# --------------Source Assembly Files---------------
sourceCode = "sourceFile.as"
mcOpTableFile = "mcOpTable"
passOneOutput = "pass I"
passTwoOutput = "pass II"
# --------------Source Assembly Files---------------


# --------------instruction Classes---------------
imperativeInstructions = ['MOVEM', 'MOVER',
                          'ADD', 'SUB', 'MUL', 'BC', 'LTORG']

ASS_DIRECTIVES = ['START', 'END', 'ORIGIN',  'EQU', 'LTORG',
                  'PURGE', 'USING', 'SEGMENT', 'END', 'ASSUME',
                  'PUBLIC', 'EXTERN', 'BALR']

REGISTER_LIST = ['AREG', 'BREG', 'CREG', 'DREG']

declarativeInstructions = ['DS', 'DC']
# --------------instruction Classes---------------

# location Counter defaults to 0
LC = 0
patternLiteral = re.compile("='(\w)'")

# PASS 1 uses OPTAB, SYMTAB, LITTAB, POOLTAB

OPTAB = {}
# --------------Pass I O/P---------------
SYMTAB = {}
LITTAB = {}
# --------------Pass I O/P---------------
```

```
def startswithAssDirective(line):
    """Determines whether the line starts not with a label"""

    for i in ASS_DIRECTIVES + imperativeInstructions + declarativeInstructions:
        if line[0] == i:
            return True
    return False


def getAddressFromSymTab(reqdSymbol):
    """Gets the required address of the symbol from SYMTABLE"""
    try:
        return SYMTAB[reqdSymbol]
    except KeyError:
        return None


def updateSymTab(reqdSymbol, reqdAdd):
    """Updates the address of the symbol with the reqdAdd"""
    SYMTAB[reqdSymbol] = reqdAdd


def imperativeStatement(line):
    """Returns true if the line is an imperative statement"""
    for i in imperativeInstructions:
        if i in line:
            return True
    return False


def declarativeStatement(line):
    """Returns true if line is a declarative statement"""
    for i in declarativeInstructions:
        if i in line:
            return True
    return False


def getSizeFromMOT(line):
    """Given an opcode and its type, return its machine size"""
    for i in line:
        if i in OPTAB:
            return OPTAB[i][1]


def getCodeFromMOT(line):
    """Given an opcode and its type, return its machine code"""
    for i in line:
        if i in OPTAB:
            return OPTAB[i][0]

def isLiteral(s):
    """Returns true if a string contains a literal"""
    p = patternLiteral.search(s)
    if p is None:
        return (False, None)
    return (True, p.groups()[0])

def literalImmediate(line):
    """Returns true if  a line contains a literal string"""
    literalList = []
    for i in line:
        res = isLiteral(i)
        if res[0]:
            literalList.append(res[1])
    return literalList

def passOne(fileHandle):
    """Generates SYMTAB LITTAB POOLTAB given a source file"""
    global LC
    LTORG_SET = False

    # Open source file for reading
    f = file(sourceCode)
    literalPending = []

    for i in f.readlines():
        line = i.split()

        if 'END' in line:
            # process remaining literals in the literal pool
            for i in literalPending:
                LITTAB[i] = LC
                literalPending.remove(i)
                LC += 1
            break

        LC += 1
        literalPending.extend(literalImmediate(line))

        # if symbol is present in lable field
        if not startswithAssDirective(line) and len(line) != 1:
            # An only literal is not a label
            SYMTAB[line[0]] = LC
        for lit in literalPending:
            # if there's a literal in the line
            # Check entry in littable whether it has been mapped to a loc
            # if no entry, create a new entry as None.
            if lit not in LITTAB:
                LITTAB[lit] = None

        if LTORG_SET:
            # Then revisit the mapping to the current location and increase LC
            if not isLiteral(line[0]):
                LTORG_SET = False
                break
            for i in literalPending:
                LITTAB[i] = LC
                literalPending.remove(i)
                LC += 1

        if 'LTORG' in line:
            LTORG_SET = True

        # if a start statement
        elif 'START' in line:
            # Update LC to denote main program
            LC = int(line[1])
            SYMTAB[line[0]] = LC

        elif 'EQU' in line:
            # place the address of the third value as that of the first
            currAdd = getAddressFromSymTab(line[2])
            updateSymTab(line[0], currAdd)

        elif imperativeStatement(line) or declarativeStatement(line):
            size = getSizeFromMOT(line)
            LC += size

    # Close source file
    f.close()

    # ------------OUTPUT OF PASS-1 ----------------
    fileHandle.write('\nSYMBOL TABLE: ')
    for i in SYMTAB.iteritems():
        fileHandle.writelines(('\n' + str(i[0]) + '\t' + str(i[1])))

    fileHandle.write('\n\nLITERAL TABLE ')
    for i in LITTAB.iteritems():
        fileHandle.writelines(('\n' + str(i[0]) + '\t' + str(i[1])))
    # -----------OUTPUT OF PASS 2 -------------------
```

```
def passTwo(fileHandle):
    """Generates final machine code using symbol table, literal table"""
    global LC

    # Open source file for reading
    f = file(sourceCode)

    for i in f.readlines():
        line = i.split()
        if 'END' in line:
            break

        if 'START' in line:
            LC = SYMTAB['START']

        elif imperativeStatement(line) or declarativeStatement(line):
            # Process Operands carefully
            operands = []

            for possibleOp in line:
                if possibleOp in REGISTER_LIST:
                    operands.append(REGISTER_LIST.index(possibleOp) + 1)

                elif possibleOp in SYMTAB:
                    operands.append(SYMTAB[possibleOp])

                elif possibleOp in LITTAB:
                    operands.append(LITTAB[possibleOp])

            opcode = getCodeFromMOT(line)
            length = getSizeFromMOT(line)
            fileHandle.write(('\n' + str(LC) + '\t'))
            for i in operands:
                fileHandle.write(str(i) + '\t')
            fileHandle.write(('\t' + str(opcode) + '\t' + str(length)))
            LC += length

    f.close()

def main():
    # Read input source Files, mcOPtable and pseudoOptable
    fOp = file(mcOpTableFile)
    for i in fOp.readlines():
        l = i.split()
        OPTAB[l[0]] = (l[1], int(l[2]))
    fOp.close()
    pA = file(passOneOutput, 'w')
    pA.writelines('\n-----------OUTPUT OF PASS I------------\n')
    passOne(pA)
    pA.writelines('\n\n-----------OUTPUT OF PASS I-----------\n')

    # Read symbol table, literal table and machine opcode table and produce final code
    pB = file(passTwoOutput, 'w')
    pB.write('\n-----------OUTPUT OF PASS II-----------\n')
    passTwo(pB)
    pB.write('\n\n-----------OUTPUT OF PASS II-----------\n')

if __name__ == '__main__':
    main()
```

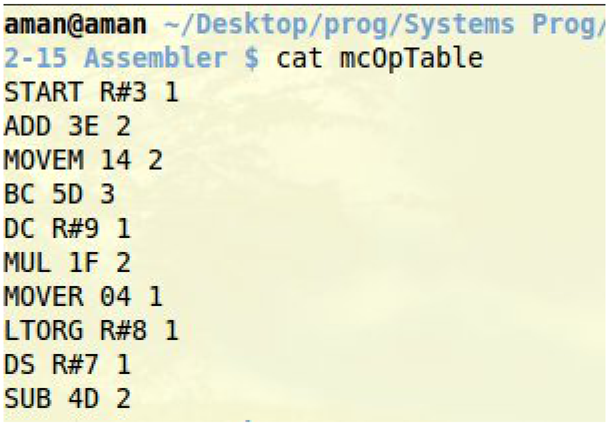
INPUT:

Source File



```
aman@aman ~/Desktop/prog/Systems Prog/2-15 Assembler $ cat sourceFile.as
     START 200
     MOVER AREG ='5'
     MOVEM AREG A
LOOP MOVER AREG A
     MOVER CREG B
     ADD CREG ='1'
     BC ANY NEXT
     LTORG
        ='5'
        ='1'
NEXT SUB AREG ='4'
     BC LT BACK
LAST STOP
     MUL CREG B
A    DS 1
BACK EQU LOOP
B    DS 1
     END
```

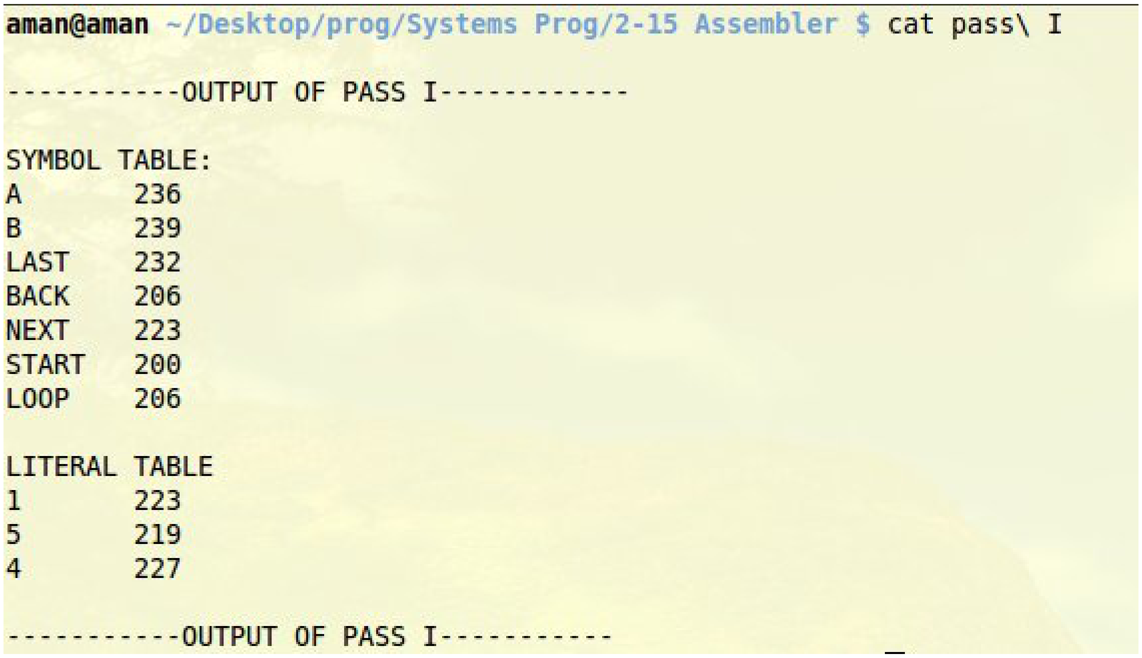
Machine Opcode Table



```
aman@aman ~/Desktop/prog/Systems Prog/
2-15 Assembler $ cat mcOpTable
START R#3 1
ADD 3E 2
MOVEM 14 2
BC 5D 3
DC R#9 1
MUL 1F 2
MOVER 04 1
LTORG R#8 1
DS R#7 1
SUB 4D 2
```

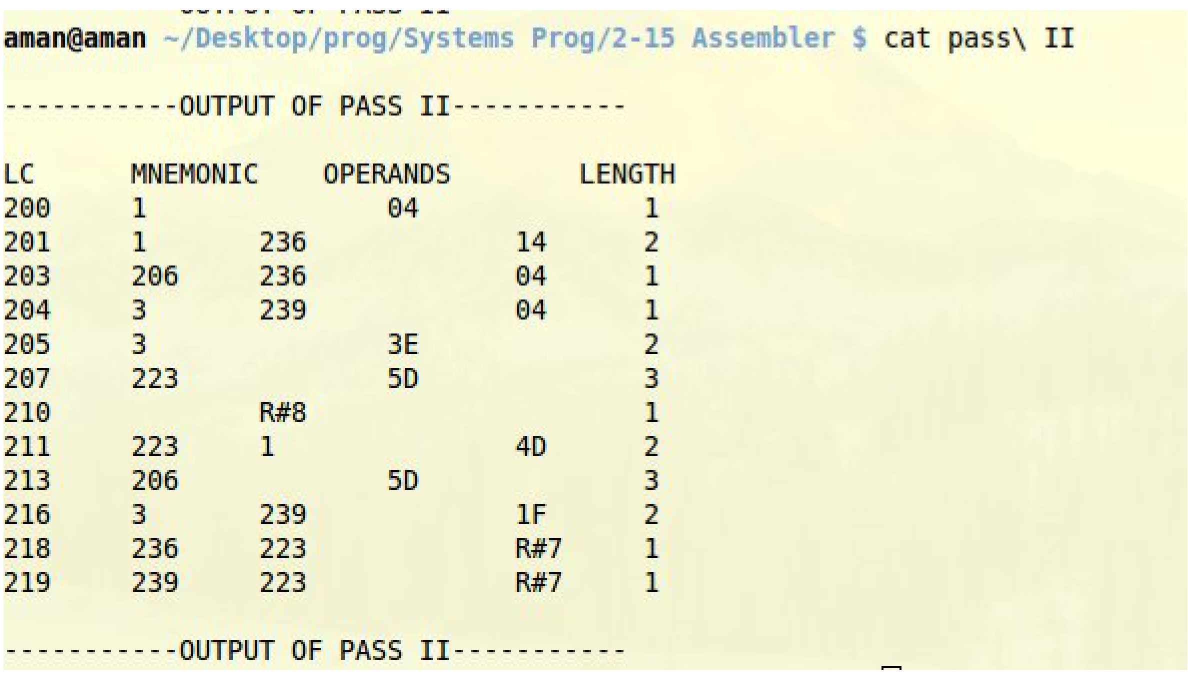
OUTPUTS:

Pass I Output File



```
aman@aman ~/Desktop/prog/Systems Prog/2-15 Assembler $ cat pass\ I

-----------OUTPUT OF PASS I------------

SYMBOL TABLE:
A       236
B       239
LAST    232
BACK    206
NEXT    223
START   200
LOOP    206

LITERAL TABLE
1       223
5       219
4       227

-----------OUTPUT OF PASS I-----------
```

Pass II Output File



```
aman@aman ~/Desktop/prog/Systems Prog/2-15 Assembler $ cat pass\ II

-----------OUTPUT OF PASS II-----------

LC      MNEMONIC    OPERANDS        LENGTH
200     1               04              1
201     1       236             14      2
203     206     236             04      1
204     3       239             04      1
205     3               3E              2
207     223             5D              3
210             R#8                     1
211     223     1               4D      2
213     206             5D              3
216     3       239             1F      2
218     236     223             R#7     1
219     239     223             R#7     1

-----------OUTPUT OF PASS II-----------
```

Program - 4

To implement a Text Editor

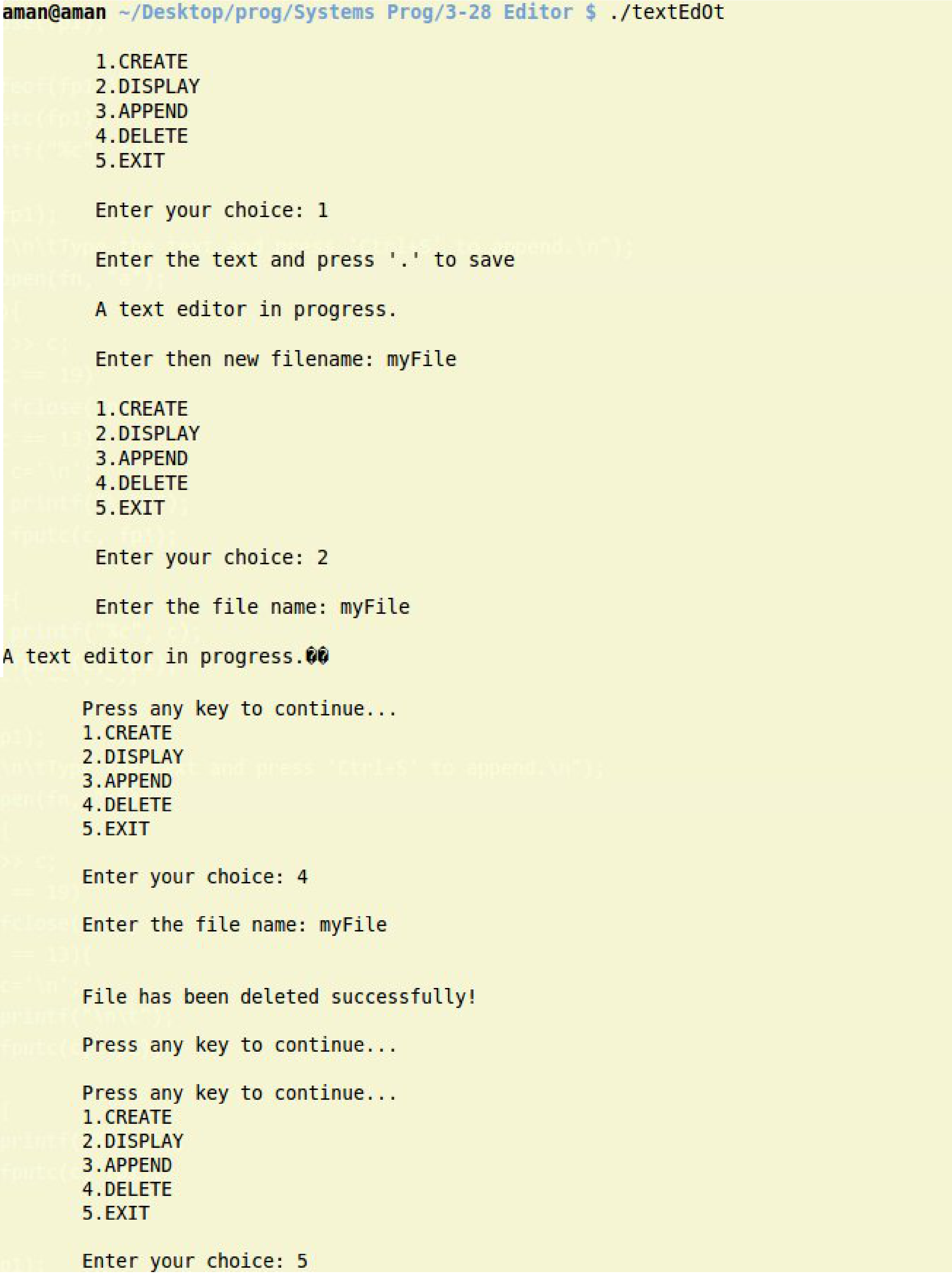
Description:

A text editor is a computer program that lets a user enter, change, store, and usually print text (characters and numbers, each encoded by the computer and its input and output devices, arranged to have meaning to users or to other programs). Typically, a text editor provides an "empty" display screen (or "scrollable page") with a fixed-line length and visible line numbers. One can then fill the lines in with text, line by line. A special command line lets you move to a new page, scroll forward or backward, make global changes in the document, save the document, and perform other actions. After saving a document, you can then print it or display it. Before printing or displaying it, you may be able to format it for some specific output device or class of output device. Text editors can be used to enter program language source statements or to create documents such as technical manuals.

Program:

```
#include<bits/stdc++.h>
using namespace std;

int i, j, ec, fg, ec2;
char fn[20], e,c;
FILE *fp1, *fp2,*fp;

void Create(){
    fp1=fopen("temp.txt", "w");
    printf("\n\tEnter the text and press '.' to save\n\n\t");

    while(1){
        c = getchar();
        fputc(c, fp1);
        if(c == '.'){
            fclose(fp1);
            printf("\n\tEnter then new filename: ");
            scanf("%s", fn);
            fp1 = fopen("temp.txt",  "r");
            fp2 = fopen(fn,  "w");

            while(!feof(fp1)){
                c = getc(fp1);
                putc(c, fp2);
            }
            fclose(fp2);
            break;
        }
    }
}

void Display(){
    printf("\n\tEnter the file name: ");
    scanf("%s", fn);
    fp1 = fopen(fn, "r");
    if(fp1 == NULL){
        printf("\n\tFile not found!");
        fclose(fp1);
        printf("\n\n\tPress any key to continue...");
    }
    while(!feof(fp1)){
        c = getc(fp1);
        printf("%c", c);
    }
    fclose(fp1);
    printf("\n\n\tPress any key to continue...");
}

void Delete(){
    printf("\n\tEnter the file name: ");
    scanf("%s", fn);
    fp1 = fopen(fn, "r");
    if(fp1 == NULL){
        printf("\n\tFile not found!");
        printf("\n\n\tPress any key to continue...");
    }
    fclose(fp1);
    if(remove(fn) == 0){
        printf("\n\n\tFile has been deleted successfully!");
        printf("\n\n\tPress any key to continue...");
    }
    else
        printf("\n\tError!\n");
        printf("\n\n\tPress any key to continue...");
}

void Append(){
    printf("\n\tEnter the file name: ");
    scanf("%s", fn);
    fp1 = fopen(fn, "r");
    if(fp1 == NULL){
        printf("\n\tFile not found!");
        fclose(fp1);
    }
    while(!feof(fp1)){
        c=getc(fp1);
        printf("%c", c);
    }
    fclose(fp1);
    printf("\n\tType the text and press 'Ctrl+S' to append.\n");
    fp1 = fopen(fn, "a");
    while(1){
        cin >> c;
        if(c == 19)
            fclose(fp1);
        if(c == 13){
            c='\n';
            printf("\n\t");
            fputc(c, fp1);
        }
        else{
            printf("%c", c);
            fputc(c, fp1);
        }
    }
    fclose(fp1);
}

int main(){
    while(1){
        printf("\n\t1.CREATE\n\t2.DISPLAY\n\t3.APPEND\n\t4.DELETE\n\t5.EXIT\n");
        printf("\n\tEnter your choice: ");
        scanf("%d", &ec);
        switch(ec){
            case 1:
                Create();
                break;
            case 2:
                Display();
                break;
            case 3:
                Append();
                break;
            case 4:
                Delete();
                break;
            case 5:
                exit(0);
        }
    }
    return 0;
}
```

OUTPUT:



```
aman@aman ~/Desktop/prog/Systems Prog/3-28 Editor $ ./textEdOt

        1.CREATE
        2.DISPLAY
        3.APPEND
        4.DELETE
        5.EXIT

        Enter your choice: 1

        Enter the text and press '.' to save

        A text editor in progress.

        Enter then new filename: myFile

        1.CREATE
        2.DISPLAY
        3.APPEND
        4.DELETE
        5.EXIT

        Enter your choice: 2

        Enter the file name: myFile

A text editor in progress.��

        Press any key to continue...
        1.CREATE
        2.DISPLAY
        3.APPEND
        4.DELETE
        5.EXIT

        Enter your choice: 4

        Enter the file name: myFile


        File has been deleted successfully!

        Press any key to continue...

        Press any key to continue...
        1.CREATE
        2.DISPLAY
        3.APPEND
        4.DELETE
        5.EXIT

        Enter your choice: 5
```

PROGRAM – 5

LEXICAL ANALYZER

Description:

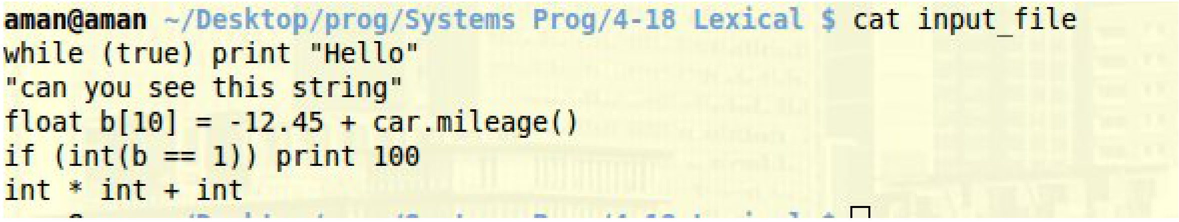
Lexical analysis is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings with an identified "meaning"). A program that performs lexical analysis may be called a lexer, tokenizer, or scanner (though "scanner" is also used to refer to the first stage of a lexer).

In this program, a file is read line by line and is analyzed for subscript operators, accessor methods, identifiers, restricted keywords, parenthesis, string literals, operators, numeric constants and so on. The program relies heavily on the use of regular expressions (regex) for each of the category mentioned above. The regex are specified in the order of decreasing precedence for each of the token. It is then matched accordingly, and a stream of token is generated.

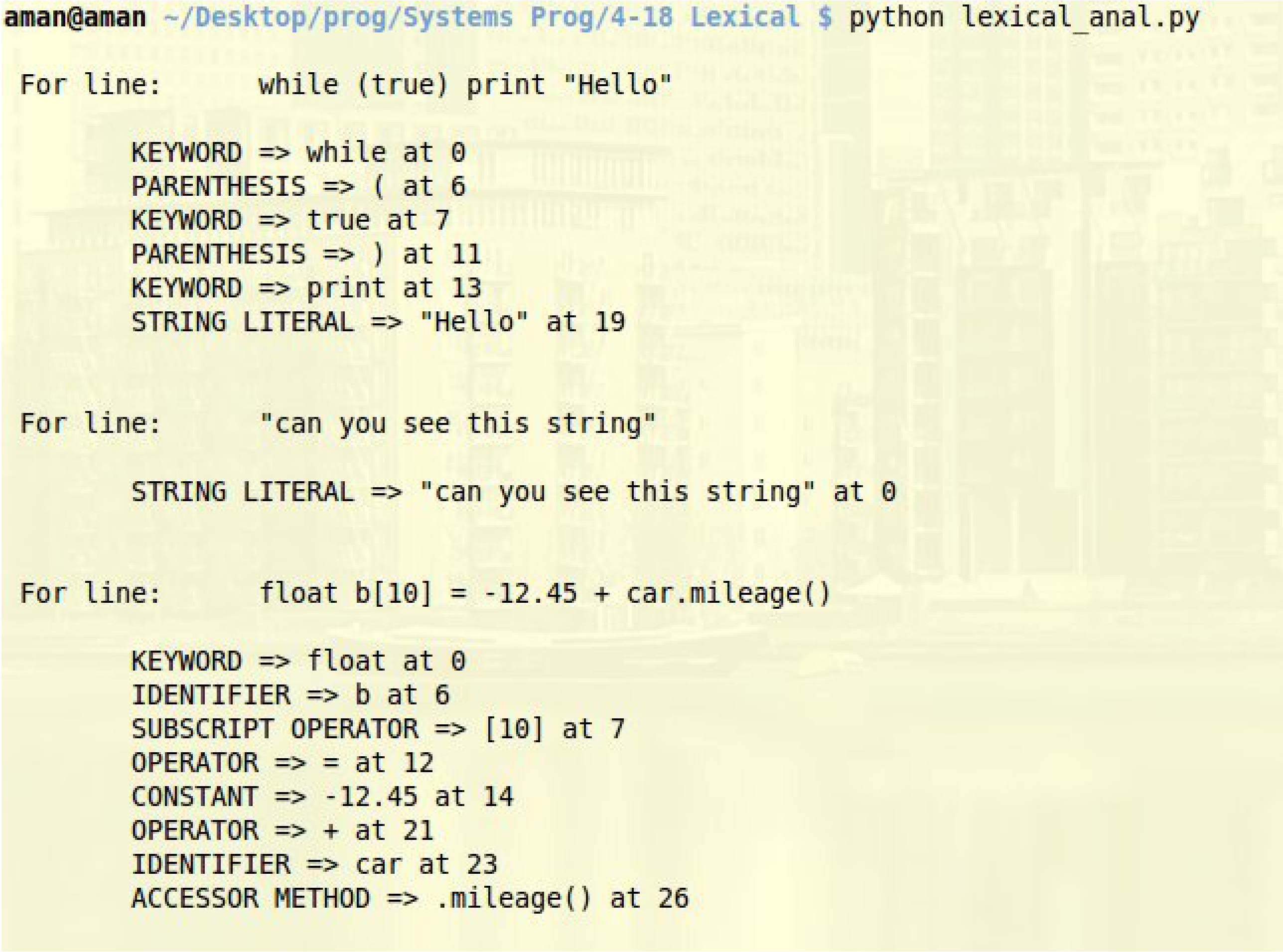
Program (Python) :

```
import re
input_file = "input_file"
# Read an input file line by line and analyze it lexically by separating
# it into constants, keywords and identifiers

# Obviously, first the whole string will be matched for the largest regex
# After that the rules of precedence follow
keywords = 'int|float|double|real|bool|do|while|if|then|else|return|main|switch\
        |char|byte|static|void|printf|print|true|false|NULL|extern'

regexList = [
    (re.compile("[\(\)]"), 'PARENTHESIS'),
    (re.compile('\.(\S)+'), 'ACCESSOR METHOD'),
    (re.compile('\"(.*?)\"'), 'STRING LITERAL'),
    (re.compile('\[.*\]'), 'SUBSCRIPT OPERATOR'),
    (re.compile("[-+]?[0-9]*\.?[0-9]+"), 'CONSTANT'),
    (re.compile("(==)|=|\-\-|\-|\+\+|\+|\*|\/|%|&|\^"), 'OPERATOR'),
    (re.compile(keywords), 'KEYWORD'),
    (re.compile("(_|[a-zA-Z])\w*"), 'IDENTIFIER')
]

class Token:
    """define what a token actually is"""
    def __init__(self, type, val, pos):
        self.type = type
        self.val = val
        self.pos = pos

class UndefinedTokenError(Exception):
    """Throw error when an unidentifiable input is achieved"""
    def __init__(self, pos):
        self.pos = pos

class LexicalAnalyzer:
    """Create a lexical analyzer"""

    def __init__(self, regexList):
        """Initialize the constructor"""
        self.regexList = regexList
        self.non_whitespace = re.compile("\S")

    def input(self, line):
        """Feed a new line input to the lexical analyzer"""
        self.line = line
        self.pos = 0

    def tokenStream(self):
        """Generate tokens one by one"""
        if self.pos >= len(self.line):
            return None  # done analyzing the last symbol

        # else check if there exists any non whitespace character
        match = self.non_whitespace.search(self.line[self.pos:])
        if match:
            self.pos += match.start()
        else:
            return None

        # For all the pairs <regular expressions, type>, search for each
        # if match is found, then increment position uptill where the match is
        # found + 1 so that the scanning may continue later
        for regex, type_identifier in self.regexList:
            match = regex.match(self.line[self.pos:])

            if match:
                val = self.line[self.pos +
                                match.start(): self.pos + match.end()]
                curToken = Token(type_identifier, val, self.pos)
                self.pos += match.end()
                return curToken
        # not a single regular expressions matched the string
        raise UndefinedTokenError(self.pos)

    def tokens(self):
        """Generate tokens one by one"""
        while True:
            currentToken = self.tokenStream()
            if not currentToken:
                break
            yield currentToken

def main():
    """Reads an input file line by line and does lexical analysis"""

    f = file(input_file, 'r')
    lex = LexicalAnalyzer(regexList)

    for i in f.readlines():
        i = i.strip()
        print '\n For line: \t', i, '\n'
        lex.input(i)

        try:
            for tok in lex.tokens():
                print tok
        except UndefinedTokenError, unLex:
            print 'Error at position: ', unLex.pos
        print

    f.close()

if __name__ == '__main__':
    main()
```

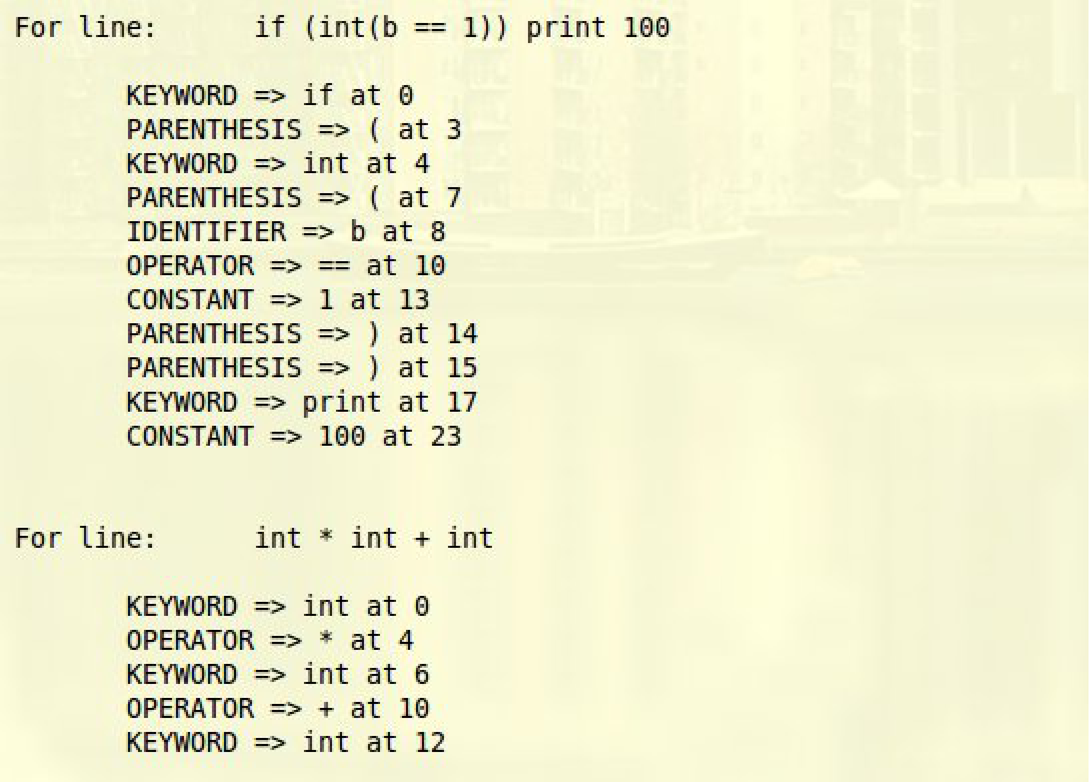
OUTPUT:



```
aman@aman ~/Desktop/prog/Systems Prog/4-18 Lexical $ cat input_file
while (true) print "Hello"
"can you see this string"
float b[10] = -12.45 + car.mileage()
if (int(b == 1)) print 100
int * int + int
```



```
aman@aman ~/Desktop/prog/Systems Prog/4-18 Lexical $ python lexical_anal.py

 For line:      while (true) print "Hello"

        KEYWORD => while at 0
        PARENTHESIS => ( at 6
        KEYWORD => true at 7
        PARENTHESIS => ) at 11
        KEYWORD => print at 13
        STRING LITERAL => "Hello" at 19


 For line:      "can you see this string"

        STRING LITERAL => "can you see this string" at 0


 For line:      float b[10] = -12.45 + car.mileage()

        KEYWORD => float at 0
        IDENTIFIER => b at 6
        SUBSCRIPT OPERATOR => [10] at 7
        OPERATOR => = at 12
        CONSTANT => -12.45 at 14
        OPERATOR => + at 21
        IDENTIFIER => car at 23
        ACCESSOR METHOD => .mileage() at 26
```



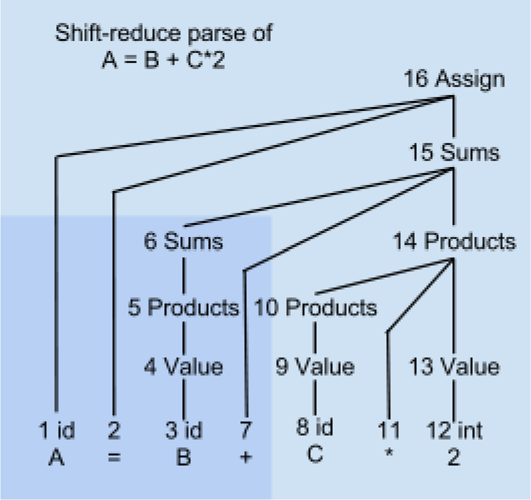
```
For line:      if (int(b == 1)) print 100

       KEYWORD => if at 0
       PARENTHESIS => ( at 3
       KEYWORD => int at 4
       PARENTHESIS => ( at 7
       IDENTIFIER => b at 8
       OPERATOR => == at 10
       CONSTANT => 1 at 13
       PARENTHESIS => ) at 14
       PARENTHESIS => ) at 15
       KEYWORD => print at 17
       CONSTANT => 100 at 23


For line:      int * int + int

       KEYWORD => int at 0
       OPERATOR => * at 4
       KEYWORD => int at 6
       OPERATOR => + at 10
       KEYWORD => int at 12
```

PROGRAM – 6

SHIFT - REDUCE BOTTOM UP PARSER

Description:

Shift-reduce parsing attempts to construct a parse tree for an input string beginning at the leaves and working up towards the root. In other words, it is a process of "reducing" (opposite of deriving a symbol using a production rule) a string w to the start symbol of a grammar. At every (reduction) step, a particular substring matching the RHS of a production rule is replaced by the symbol on the LHS of the production.



For this program, the input is a grammar file which denotes all the possible productions for the given language, and an input string given by the user to be parsed by the SR Parser. The input string is converted into tokens by using the lexical analyser.

A shift-reduce parser works by doing some combination of Shift steps and Reduce steps, hence the name.

•A **Shift** step advances in the input stream by one symbol. That shifted symbol becomes a new single-node parse tree.

•A **Reduce** step applies a completed grammar rule to some of the recent parse trees, joining them together as one tree with a new root symbol.

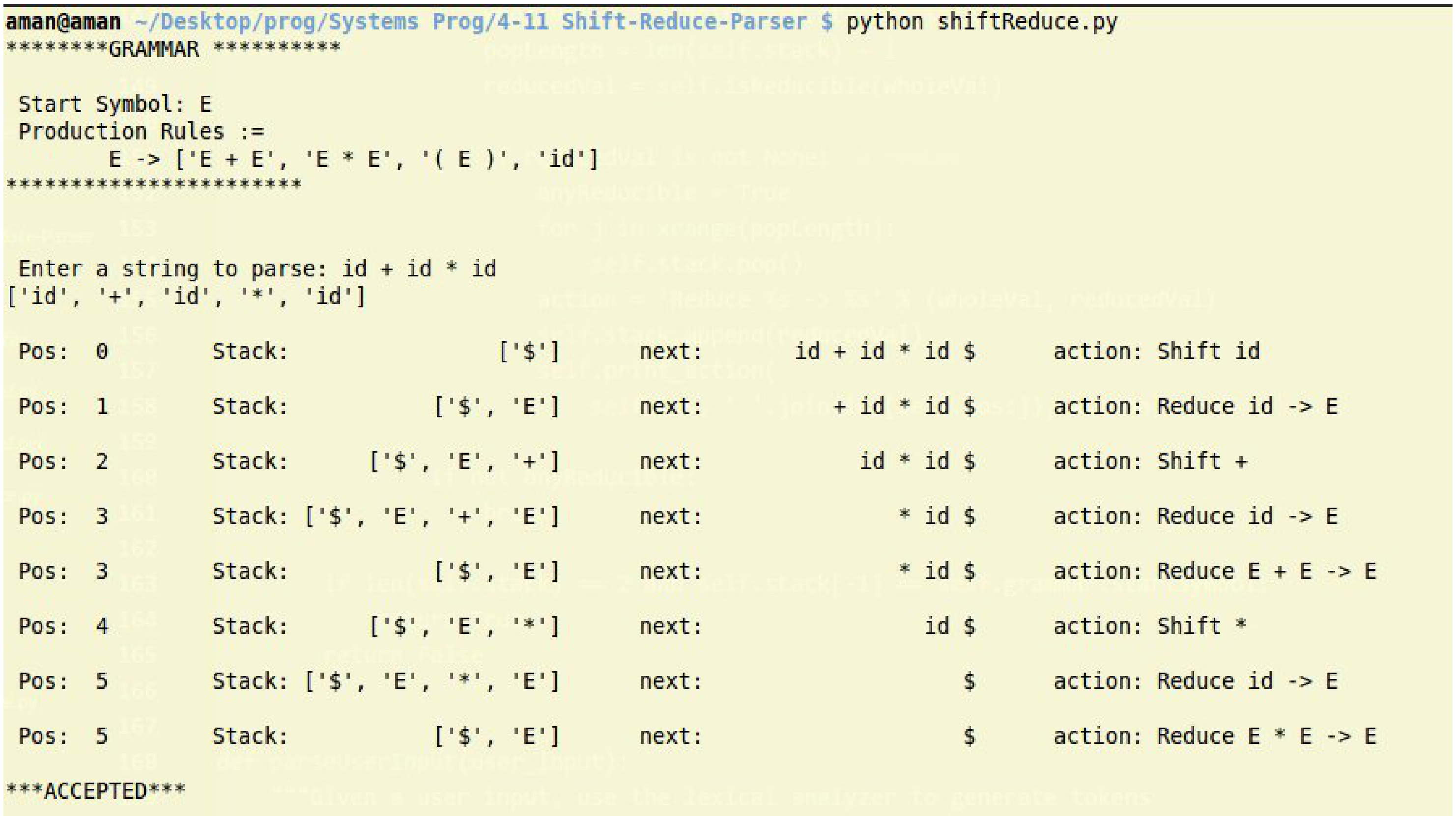
The parser continues with these steps until all of the input has been consumed and all of the parse trees have been reduced to a single tree representing an entire legal input.

At every parse step, the entire input text is divided into parse stack, current lookahead symbol, and remaining unscanned text. The parser's next action is determined by the rightmost stack symbol(s) and the lookahead symbol. The action is read from a table containing all syntactically valid combinations of stack and lookahead symbols.

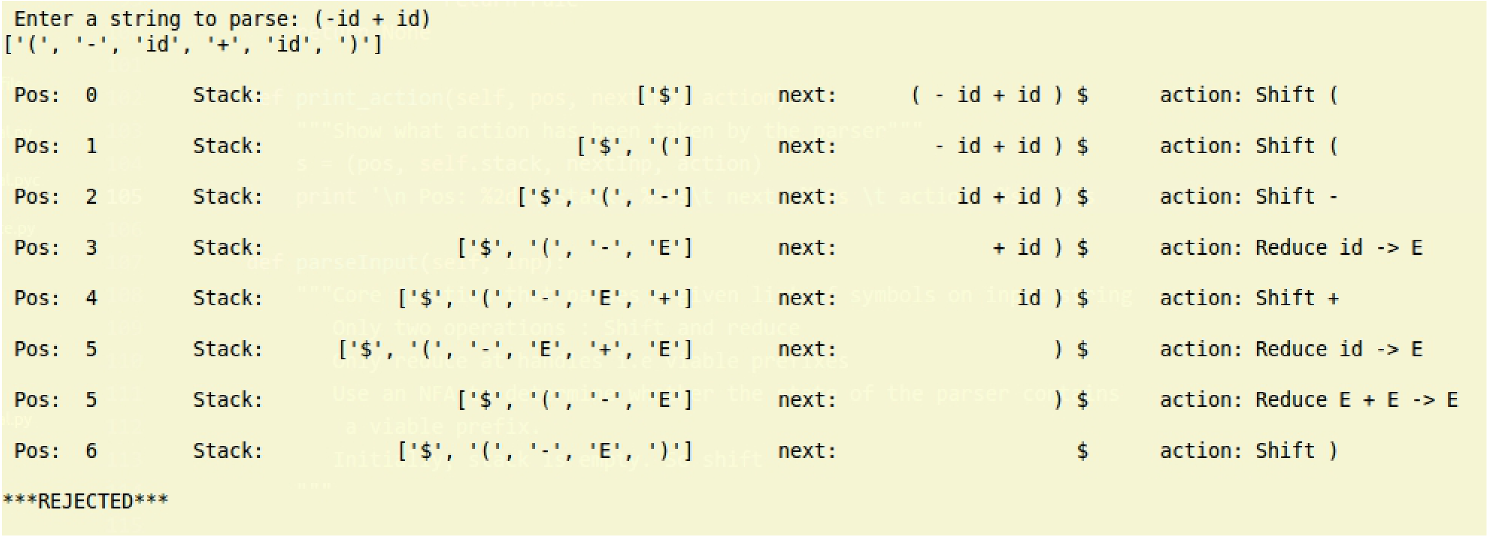
PROGRAM (PYTHON):

```
import lexical_anal as my_lexical
"""
A program to implement shift reduce parser which takes an input a context free
grammar G and an input string to test whether the input string lies within the
set of rules denotes by G.  Raise error accordingly if required.
"""
GRAMMAR_FILE = "grammar_file"

class IllegalProductionError(Exception):
    """raise an error when a production doesn't satisfy CFG requirements"""
    def __init__(self, line, msg):
        self.msg = msg
        self.line = line


class ParsingError(Exception):
    """Raise an error on unsuccessful parsing of a line"""
    def __init__(self, msg):
        self.msg = msg

class Grammar:
    """
    A grammar is a set of production rules of the form A -> B
    where,
        A is the set of non terminals
        B is the set of terminals and non terminals and is not empty
    Each production rule can be represented by a key value pair where key: A
    and value: list of individual subrules
    """
    def __init__(self):
        """Initialize a grammar"""
        self.startSymbol = ''
        self.productions = {}

    def __str__(self):
        """Allow grammar to be printed onto the screen"""
        string = '********GRAMMAR **********\n'
        string += '\n Start Symbol: %s' % (self.startSymbol)
        string += '\n Production Rules := '
        for i in self.productions:
            string += '\n\t%s -> %s' % (i, self.productions[i])

        string += '\n***********************\n'
        return string

    def addProduction(self, line):
        """Add a production rule as parsed from the gven line"""
        if not line:
            raise IllegalProductionError(line, 'Empty production')

        line = map(lambda x: x.strip(), line.split('->'))
        if len(line) != 2:
            raise IllegalProductionError(line, 'Wrong Non Terminal format')

        leftProd, rightProd = line[0], line[1]
        possibleProd = map(lambda x: x.strip(), rightProd.split('|'))

        if len(possibleProd) < 1:
            raise IllegalProductionError(
                line, 'Empty production values are not allowed')

        # set start symbol for the first valid production
        if len(self.productions) == 0:
            self.startSymbol = leftProd

        self.productions[leftProd] = possibleProd


class ShiftReduceParser:
    """
    A shift reduce parser that takes an input a grammar G according to which
    any string is matched. It initializes its structures, namely:
        * A stack for processing read terminals
        * A marker to indicate the current position in stack
    """
    def __init__(self, grammar):
        self.grammar = grammar  # A grammar defining the parsing stages
        self.stack = []  # A stack to maintain the currently read symbols
        self.pos = 0  # denotes the current position of the input symbol
        self.end = 0    # denotes the end of the input symbol

    def isReducible(self, string):
        """Find if a string occurs in rhs of some production"""
        for rule in self.grammar.productions:
            for RHS in self.grammar.productions[rule]:
                if RHS == string:
                    return rule
        return None
    def print_action(self, pos, nextInp, action):
        """Show what action has been taken by the parser"""
        s = (pos, self.stack, nextInp, action)
        print '\n Pos: %2d \tStack: %20s \t next: %20s \t action: %s ' % s

    def parseInput(self, inp):
        """Core function that parses a given list of symbols on input string
           Only two operations : Shift and reduce
           Only reduce at handles i.e viable prefixes. Initially, stack is empty. So shift
        """
        inp.append('$')     # Mark the end of the input
        self.stack = ['$']  # Mark the start of the stack
        self.pos, self.end = 0, len(inp) - 1
        action = ''

        if inp is None or len(inp) == 0:
            raise ParsingError('Can\'t parse an empty string')

        self.print_action(self.pos, ' '.join(
            inp), 'Shift %s' % (inp[self.pos]))

        while self.pos != self.end:
            # Check if the current stack symbol can be reduced
            # if not then shift the next symbol
            curVal = inp[self.pos]
            reducedVal = self.isReducible(curVal)

            if reducedVal is not None:  # reduce
                action = 'Reduce %s -> %s' % (curVal, reducedVal)
                self.stack.append(reducedVal)
            else:  # shift
                action = 'Shift %s' % (curVal)
                self.stack.append(curVal)

            self.pos += 1
            self.print_action(self.pos, ' '.join(inp[self.pos:]), action)
            # After this, check all reducible suffixes of the stack

            while True:
                anyReducible = False
                for i in xrange(1, len(self.stack)):
                    wholeVal = ' '.join(self.stack[i:self.pos + 1])
                    popLength = len(self.stack) - i
                    reducedVal = self.isReducible(wholeVal)
                    if reducedVal is not None:  # reduce
                        anyReducible = True
                        for j in xrange(popLength):
                            self.stack.pop()
                        action = 'Reduce %s -> %s' % (wholeVal, reducedVal)
                        self.stack.append(reducedVal)
                        self.print_action(
                            self.pos, ' '.join(inp[self.pos:]), action)

                if not anyReducible:
                    break

        if len(self.stack) == 2 and self.stack[-1] == self.grammar.startSymbol:
            return True
        return False

def parseUserInput(user_input):
    """Given a user input, use the lexical analyzer to generate tokens
       Return a list containing the value of these tokens"""
    lex = my_lexical.LexicalAnalyzer(my_lexical.regexList)
    lex.input(user_input)
    val = []
    try:
        for tok in lex.tokens():
            val.append(tok.val)
    except my_lexical.UndefinedTokenError:
        print 'Invalid user input: '
    return val

def main():
    """Read grammar from a file and user string
     and  Parseit accordingly"""
    g = Grammar()
    gFile = file(GRAMMAR_FILE, 'r')

    # Setup the grammar from the file
    map(lambda x: g.addProduction(x.strip()), gFile.readlines())
    print g
    srParser = ShiftReduceParser(g)

    user_input = parseUserInput(raw_input("\n Enter a string to parse: "))
    print user_input
    print '\n****ACCEPTED***' if srParser.parseInput(user_input) else '\n****REJECTED****'
    gFile.close()
```

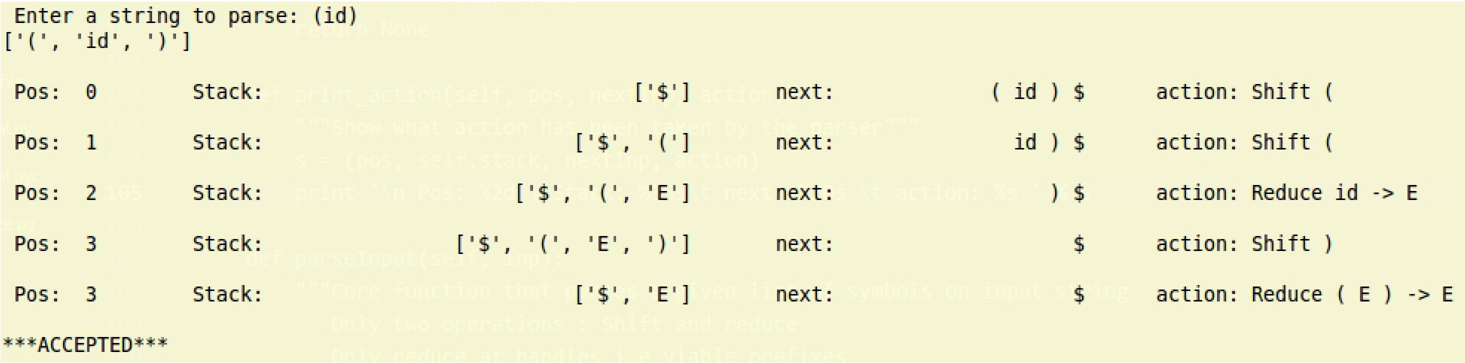
OUTPUT:



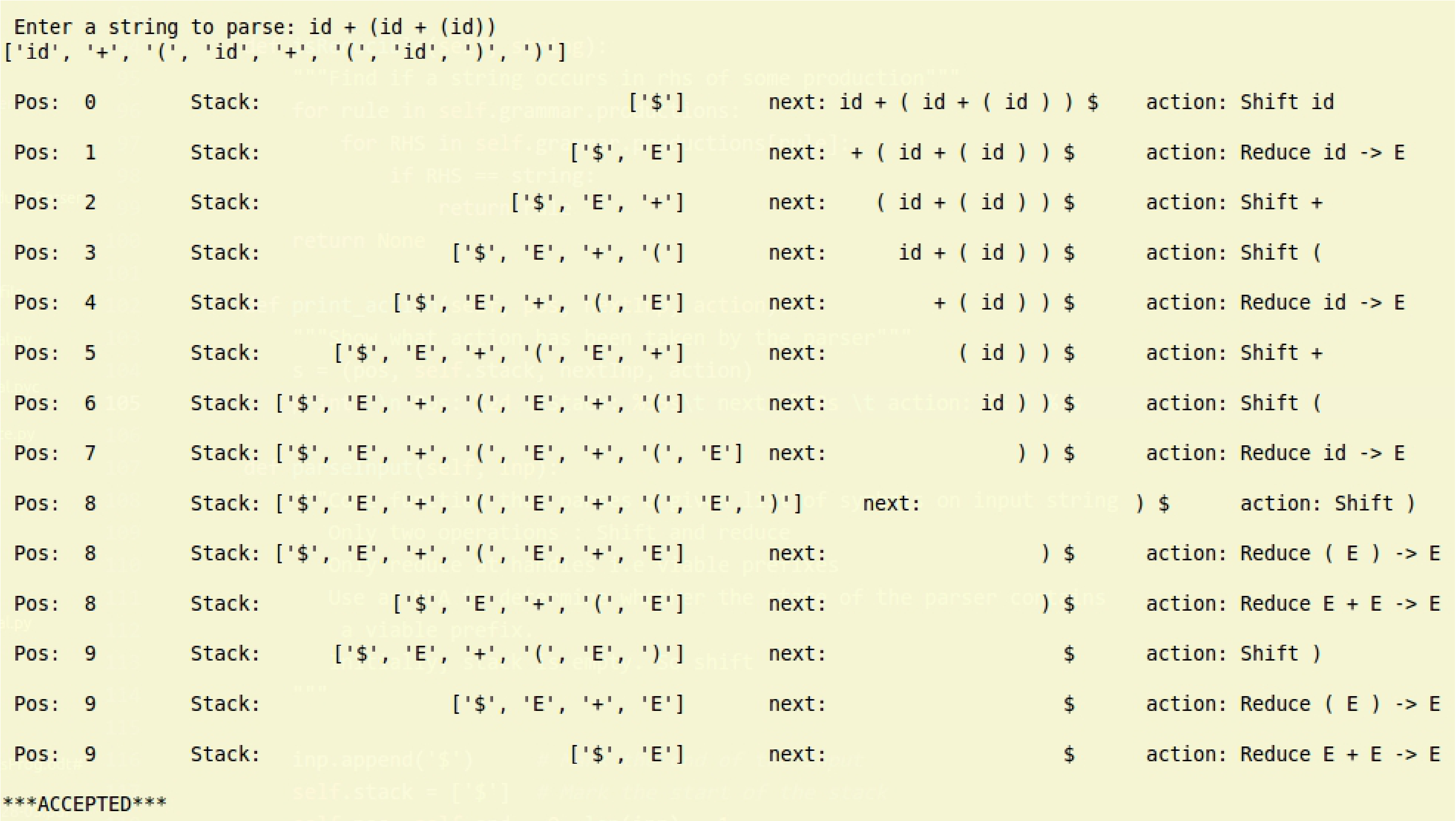
```
aman@aman ~/Desktop/prog/Systems Prog/4-11 Shift-Reduce-Parser $ python shiftReduce.py
********GRAMMAR **********

 Start Symbol: E
 Production Rules :=
        E -> ['E + E', 'E * E', '( E )', 'id']
***********************


 Enter a string to parse: id + id * id
['id', '+', 'id', '*', 'id']

 Pos:  0        Stack:                ['$']      next:       id + id * id $      action: Shift id

 Pos:  1        Stack:           ['$', 'E']      next:          + id * id $      action: Reduce id -> E

 Pos:  2        Stack:      ['$', 'E', '+']      next:            id * id $      action: Shift +

 Pos:  3        Stack: ['$', 'E', '+', 'E']      next:               * id $      action: Reduce id -> E

 Pos:  3        Stack:           ['$', 'E']      next:               * id $      action: Reduce E + E -> E

 Pos:  4        Stack:      ['$', 'E', '*']      next:                 id $      action: Shift *

 Pos:  5        Stack: ['$', 'E', '*', 'E']      next:                    $      action: Reduce id -> E

 Pos:  5        Stack:           ['$', 'E']      next:                    $      action: Reduce E * E -> E

***ACCEPTED***
```



```
 Enter a string to parse: (-id + id)
['(', '-', 'id', '+', 'id', ')']

 Pos:  0        Stack:                               ['$']       next:      ( - id + id ) $      action: Shift (

 Pos:  1        Stack:                          ['$', '(']       next:        - id + id ) $      action: Shift (

 Pos:  2        Stack:                     ['$', '(', '-']       next:          id + id ) $      action: Shift -

 Pos:  3        Stack:                ['$', '(', '-', 'E']       next:             + id ) $      action: Reduce id -> E

 Pos:  4        Stack:           ['$', '(', '-', 'E', '+']       next:               id ) $      action: Shift +

 Pos:  5        Stack:      ['$', '(', '-', 'E', '+', 'E']       next:                  ) $      action: Reduce id -> E

 Pos:  5        Stack:                ['$', '(', '-', 'E']       next:                  ) $      action: Reduce E + E -> E

 Pos:  6        Stack:           ['$', '(', '-', 'E', ')']       next:                    $      action: Shift )

***REJECTED***
```



```
 Enter a string to parse: (id)
['(', 'id', ')']

 Pos:  0       Stack:                    ['$']        next:           ( id ) $      action: Shift (

 Pos:  1       Stack:               ['$', '(']        next:             id ) $      action: Shift (

 Pos:  2       Stack:          ['$', '(', 'E']        next:                ) $      action: Reduce id -> E

 Pos:  3       Stack:     ['$', '(', 'E', ')']        next:                  $      action: Shift )

 Pos:  3       Stack:               ['$', 'E']        next:                  $      action: Reduce ( E ) -> E

***ACCEPTED***
```



```
 Enter a string to parse: id + (id + (id))
['id', '+', '(', 'id', '+', '(', 'id', ')', ')']

 Pos:  0       Stack:                              ['$']       next: id + ( id + ( id ) ) $    action: Shift id

 Pos:  1       Stack:                         ['$', 'E']       next:  + ( id + ( id ) ) $      action: Reduce id -> E

 Pos:  2       Stack:                    ['$', 'E', '+']       next:    ( id + ( id ) ) $      action: Shift +

 Pos:  3       Stack:               ['$', 'E', '+', '(']       next:      id + ( id ) ) $      action: Shift (

 Pos:  4       Stack:          ['$', 'E', '+', '(', 'E']       next:         + ( id ) ) $      action: Reduce id -> E

 Pos:  5       Stack:     ['$', 'E', '+', '(', 'E', '+']       next:           ( id ) ) $      action: Shift +

 Pos:  6       Stack: ['$', 'E', '+', '(', 'E', '+', '(']       next:             id ) ) $      action: Shift (

 Pos:  7       Stack: ['$', 'E', '+', '(', 'E', '+', '(', 'E']  next:                ) ) $      action: Reduce id -> E

 Pos:  8       Stack: ['$', 'E', '+', '(', 'E', '+', '(', 'E', ')']  next:              ) $      action: Shift )

 Pos:  8       Stack: ['$', 'E', '+', '(', 'E', '+', 'E']       next:                  ) $      action: Reduce ( E ) -> E

 Pos:  8       Stack:          ['$', 'E', '+', '(', 'E']       next:                  ) $      action: Reduce E + E -> E

 Pos:  9       Stack:     ['$', 'E', '+', '(', 'E', ')']       next:                    $      action: Shift )

 Pos:  9       Stack:               ['$', 'E', '+', 'E']       next:                    $      action: Reduce ( E ) -> E

 Pos:  9       Stack:                         ['$', 'E']       next:                    $      action: Reduce E + E -> E

***ACCEPTED***
```