Ramaiah Institute of Technology
Department of Computer Science and Engineering

COURSE DESIGN, DELIVERY AND ASSESMENT

Semester: VI

Course Code: CSL66

Course Name: Unix System Programming & Compiler Design Laboratory

Course Faculty: Parkavi.A, Sini Anna Alex, Aparna.R , Chandrika P

Sl#	Section	Course Faculty Name	Date
1	A	Sini Anna Alex, Aparna.R	10/1/2020
2	B	Sini Anna Alex, Aparna.R	10/1/2020
3	C	Dr.Parkavi A, CHNDRIKA.P	10/1/2020

Course Coordinator	Date
Dr.Parkavi.A	10/1/2020

Program Coordinator	Date
Dr. Annapurna P Patil	10/1/2020

Head of Department (Sign & Date)

COURSE DESIGN, DELIVERY AND ASSESMENT

Course code and Title : CSL66, Unix System Programming & Compiler Design Laboratory	Course Credits :0:0:1
CIE : 50 Marks	SEE : 50 Marks
Total No of Theory / Tutorial / Lab Hours : 0:0:14	
Prepared by : Parkavi.A	Date : 10/1/2020
Reviewed by : Jagadish S Kallimani	Date : 10/1/2020

Prerequisites: NIL

Course Contents:

Part A

1.Basic file I/O functions & properties of a file.
2.File Types, File access permission and File links.
3.Creating the process and process accounting.
4.Feature provided by different signal implementation.
5.Coding rules and Characteristics of Daemon Process

Part B

1.Tokenization of input
2.Validating the syntax of the input
3.Performing syntax directed translation
4.Verification of semantic
5.Generation of intermediate code
6.Optimization of code
7.Generation of assembly language code

Course Outcomes:

At the end of the course students should be able to:

1. Build analyzers to recognize tokens and errors.(PO-1,2,4,PSO-2,3)
2. Construct generators for intermediate codes, optimizations and assembly language codes(PO-1,2,4,PSO-2,3)
3. Demonstrate advanced UNIX operating system concepts and terminology.(PO-1,2,4,PSO- 2,3)

Text Books:

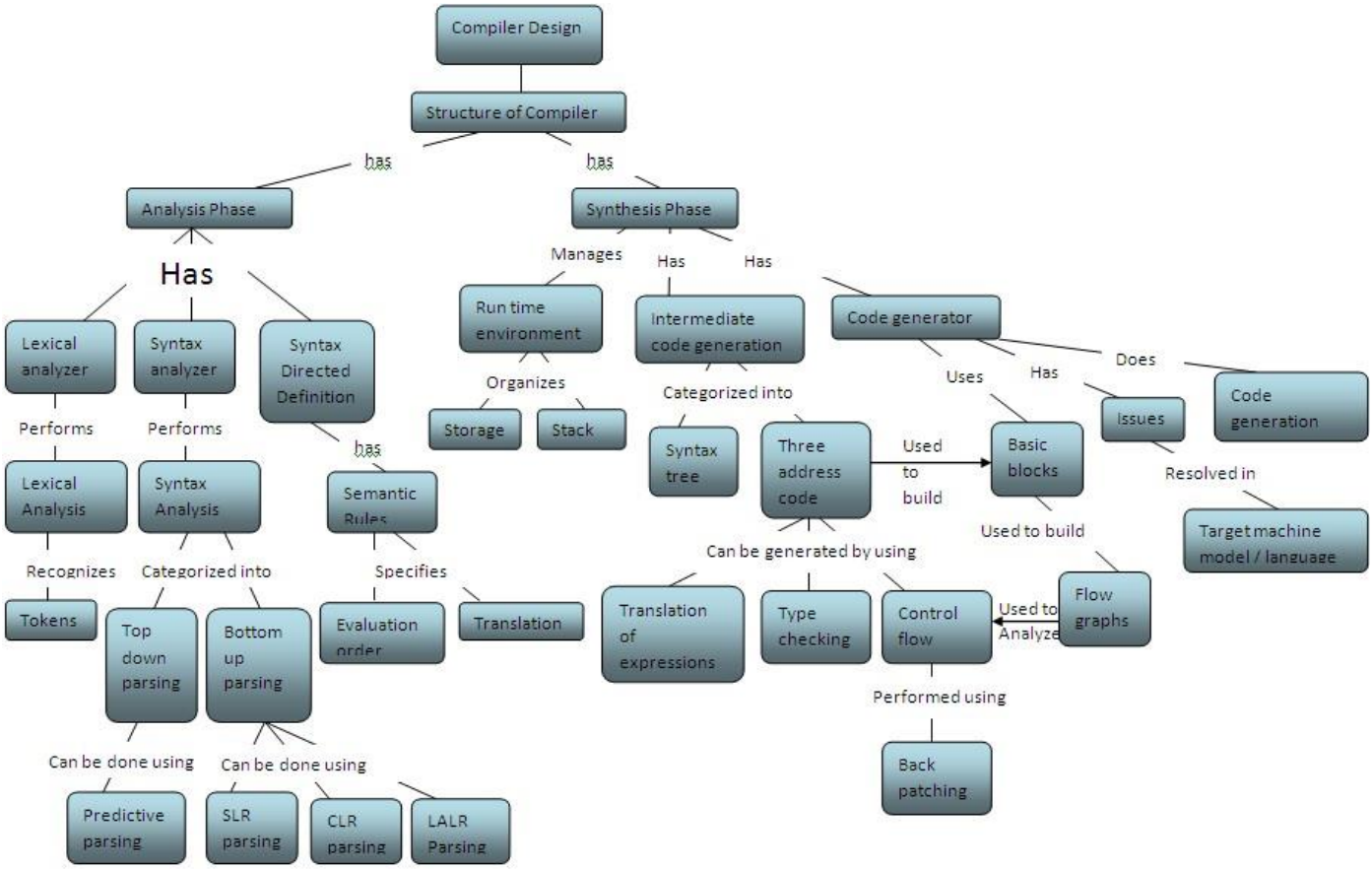
1. Alfred V Aho, Monica S. Lam, Ravi Sethi, Jeffrey D Ullman: Compilers- Principles, Techniques and Tools, 2nd Edition, Addison-Wesley, 2007.
2. W. Richard Stevens: Advanced Programming in the UNIX Environment, Second Edition, Pearson education, 2011.

Reference Books:

1. Kenneth C Louden: Compiler Construction - Principles & Practice, Brooks/Cole, CENGAGE learning, 1997.
2. Andrew W Appel: Modern Compiler Implementation in C, Cambridge University Press, 1999.
3. Terrence Chan: UNIX System Programming Using C++, First edition, Prentice Hall India, 2011.
4. Kay A Robbins and Steve Robbins: Unix Systems Programming, First Edition, Pearson Education, 2009.
5. Marc J. Rochkind: Advanced UNIX Programming, 2nd Edition, Pearson Education, 2009.

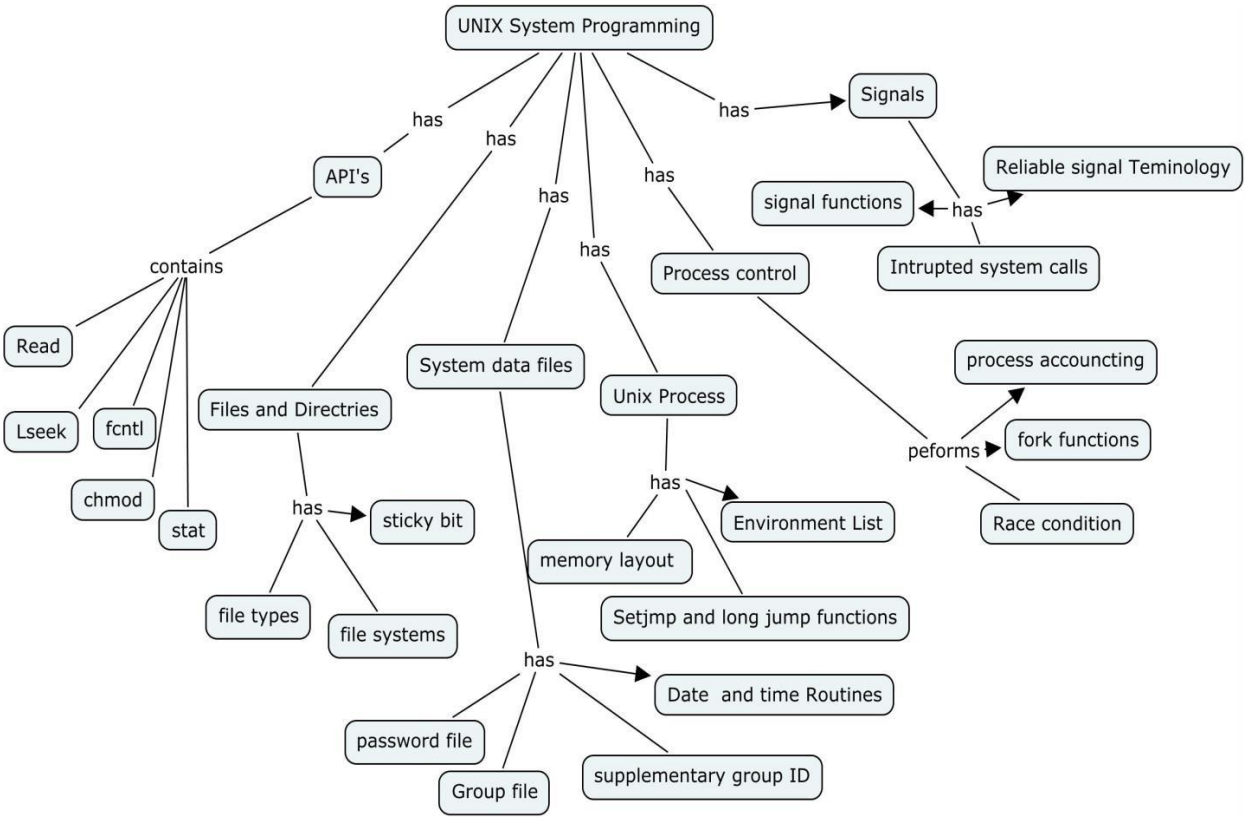
Concept map

Part A: Compiler Design Lab



Course Map:

Part B-Unix System Programming lab



Compiler design Laboratory Lesson Plan

Lesson No/Session No	Topics	No. of hours
1.	Programs on Lexical Analyzer operations	2
2.	Programs on Lexical Analyzer operations	2
3	Programs on Parser operations	2
4	Programs on Parser operations	2
5	Programs on Semantic analyzer operations	2
6	Programs on Semantic analyzer operations	2
7.	Programs on Intermediate code generator operations	2

Unix system programming Laboratory Lesson Plan

Lesson No/Session No	Topics	No. of hours
1.	C programs to handle files, command line arguments, structures and Unix commands usage , C programs to handle standard input and standard output, file handling functions – open, close, read, write, lseek	2
2	C programs to handle file attributes using fcntl, stat, umask and chmod functions , directory files, symbolic links	2
3	C program to illustrate effect of setjmp and longjmp functions on register and volatile variables, a program to setup a timer and setup handler for the timer using fork (or exec) to check who gets the timer signal,	2
4	C programs to demonstrate fork, exec functions , zombies, orphan processes, on daemons, co-operating processes , to use signal system calls	2
5	C programs to use signal system calls in various use cases, programs to implement client - server communication using pipes. (FIFO)	2
6	C programs to demonstrate inter process communication using Message Queue, to demonstrate inter process communication using shared memory	2
7	C program to demonstrate process synchronization by using semaphores to implement reader-writer problem.	2

Course Delivery:

This course uses assigned readings, lectures, and homework to enable the students to: apply the compiler design and Unix System programming concepts while programming, explore the knowledge by applying on suitable applications.

Mapping Course Outcomes with Program Outcomes:

Course Outcome	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO10	PO11	PO12	PSO 1	PSO 2	PSO 3
CO1	2	2		2										2	2
CO2	2	2		2										2	2
CO3	2	2		2										2	2

Course Assessment and Evaluation:

	What		To Whom	When/ Where (Frequency in the course)	Max Marks	Evidence Collected	Contribution to Course Outcomes
Direct & Indirect Assessment Methods	C I E	Internal Assessment Test	Students	Twice in Laboratory(Sum of the marks will be computed)	50	Answer scripts	1,2,3
							1,2,3
	S E E	Standard Examination		End of Course (Answering 1 question out of 14 questions)	50	Answer scripts	1,2,3
	End of Course Survey			End of the Course	-	Questionnaire	1,2,3 Delivery of the course

Course Assessment and Evaluation:

Questions for CIE and SEE will be designed to evaluate the various educational components (Bloom's taxonomy) such as:

CIE and SEE evaluation

S.No	Bloom's Category	Compiler design Lab Test	Unix system programming Test	Semester-End Examination
1	Remember	1	1	3
2	Understand	1	1	3
3	Apply	1	1	3
4	Analyze	1	1	3
5	Evaluate	1	1	3
6	Create	20	20	35

Ramaiah Institute of Technology
Department of Computer Science and Engineering

USP & CD Laboratory-CSL66

Part A – USP Lab

USP Lab – Session 1

1. Execute the following commands in unix with different options and note down the output and comments.

    1. ls
    2. ps
    3. cat
    4. df
    5. du
    6. grep
    7. find
    8. gcc
2. Differentiate between internal and external commands of unix.
3. Write a C program to read from keyboard and display on monitor screen.
4. Write a C program to simulate copy command that copies contents of one file into another.
5. Write a C program to simulate a calculator reading the operation and operands from command line.

USP Lab – Session 2

1. Write a C program to read from standard input and display on standard output.

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<fcntl.h>  #define
BUFFSIZE 100 int main()
{
        int n;
        char buf[BUFFSIZE];
        while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0) if
                (write(STDOUT_FILENO, buf, n) != n)
                printf("write error");
        if (n < 0) printf("read
                error");
       exit(0);

}
```

2. Write a program to read n characters from a file and append them back to the same file using dup2 function.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main()
{
    int fd1=0,fd2=0;
    char buf[50];
      if((fd1=open("t12.txt",O_RDWR,0))<0)
            printf("file open error");

      fd2=dup(fd1);
    printf("%d %d \n",fd1, fd2);
    read(fd1,buf,10);  lseek(fd2,
0L, SEEK_END);
    write(fd2, buf, 10);
    printf("%s\n",outbuf);    return
0;
}
```

3. Write a program
    a. to read first 20 characters from a file
    b. seek to 10th byte from the beginning and display 20 characters from there
    c. seek 10 bytes ahead from the current file offset and display 20 characters

d. display the file size

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h> #include<sys/types.h>
int main()
{
      int file=0, n;      char
buffer[25];
 if((file=open("testfile.txt",O_RDONLY))      <      -1)
printf("file open error\n");    if(read(file,buffer,20) !=
20)   printf("file read operation failed\n");
          else
                write(STDOUT_FILENO, buffer, 20);
          printf("\n");
          if(lseek(file,10,SEEK_SET) < 0)
                printf("lseek operation to beginning of file failed\n");
          if(read(file,buffer,20) != 20)
                printf("file read operation failed\n");
          else
                write(STDOUT_FILENO, buffer, 20);
          printf("\n");

          if(lseek(file,10,SEEK_CUR) < 0)  printf("lseek operation to
                beginning of file failed\n");
          if(read(file,buffer,20) != 20)
                printf("file read operation failed\n");
          else
                write(STDOUT_FILENO, buffer, 20);
          printf("\n");

          if((n = lseek(file,0,SEEK_END)) <0)
                printf("lseek operation to end of file failed\n");
      printf("size of file is %d bytes\n",n);
      close(file);        return 0;

}
```

4. Write a program to display the file content in reverse order using lseek system call.

```
#include<stdlib.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
#include<sys/stat.h>
#include<unistd.h>

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

```
{     int source, dest,
n;     char buf;     int
filesize;     int i;

   if (argc != 3) {
      fprintf(stderr, "usage %s <source> <dest>", argv[0]);
exit(-1);
   }

   if ((source = open(argv[1], O_RDONLY)) < 0)
{   fprintf(stderr, "can't open source\n");
      exit(-1);
   }

   if ((dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC)) < 0)
   {   fprintf(stderr, "can't create dest\n");
      exit(-1);
   }

   filesize = lseek(source, (off_t) 0, SEEK_END);
   printf("Source file size is %d\n", filesize);

   for (i = filesize - 1; i >= 0; i--)
   {
      lseek(source, (off_t) i, SEEK_SET);

      if ((n = read(source, &buf, 1)) != 1) {
fprintf(stderr, "can't read 1 byte");
         exit(-1);
      }

      if ((n = write(dest, &buf, 1)) != 1) {
fprintf(stderr, "can't write 1 byte");
         exit(-1);
      }

   }
   write(STDOUT_FILENO, "DONE\n", 5);
   close(source);
   close(dest);

return 0;
}
```

USP Lab – Session 3

1. Program to print file types

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>

int main(int argc, char *argv[])
{ int i; struct
stat buf; char
*ptr;
for (i = 1; i < argc; i++)
{
        printf("%s: ", argv[i]);    if
(lstat(argv[i], &buf) < 0)
            {
                    err_ret("lstat error");
                    continue;
            }
            if (S_ISREG(buf.st_mode))
                    ptr = "regular";
        else if (S_ISDIR(buf.st_mode))
                ptr = "directory";
 else if (S_ISCHR(buf.st_mode))   ptr =
"character special";
 else if (S_ISBLK(buf.st_mode))   ptr =
"block special";
            else if (S_ISFIFO(buf.st_mode))
                    ptr = "fifo";
            else if (S_ISLNK(buf.st_mode))
                ptr = "symbolic link";    else
if (S_ISSOCK(buf.st_mode))
        ptr = "socket";   else
                ptr = "** unknown mode **";    printf("%s\n",
ptr);
} exit(0);
}
```

2. Program to print file access permissions.(Write a program to display various details of a file using stat structure(At least 5 fields)

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(int argc, char **argv)
```

```
{
   if(argc != 2)
return 1;

   struct stat fileStat;
   if(stat(argv[1],&fileStat) < 0)
      return 1;

   printf("Information for %s\n",argv[1]);     printf("--
-------------------------\n");     printf("File Size: \t\t%d
bytes\n",fileStat.st_size);     printf("Number of Links:
\t%d\n",fileStat.st_nlink);     printf("File inode:
\t\t%d\n",fileStat.st_ino);

   printf("File Permissions: \t");     printf(
(S_ISDIR(fileStat.st_mode)) ? "d" : "-");     printf(
(fileStat.st_mode & S_IRUSR) ? "r" : "-");     printf(
(fileStat.st_mode & S_IWUSR) ? "w" : "-");     printf(
(fileStat.st_mode & S_IXUSR) ? "x" : "-");     printf(
(fileStat.st_mode & S_IRGRP) ? "r" : "-");     printf(
(fileStat.st_mode & S_IWGRP) ? "w" : "-");     printf(
(fileStat.st_mode & S_IXGRP) ? "x" : "-");     printf(
(fileStat.st_mode & S_IROTH) ? "r" : "-");     printf(
(fileStat.st_mode & S_IWOTH) ? "w" : "-");
printf( (fileStat.st_mode & S_IXOTH) ? "x" : "-");
printf("\n\n");

   printf("The file %s a symbolic link\n", (S_ISLNK(fileStat.st_mode)) ? "is" : "is not");

   return 0;
}
```

3. Programs to demonstrate usage of access, umask and chmod functions.

```
#include <fcntl.h>
int main(int argc, char *argv[])
{ if (argc !=
2)
      printf("usage: argv[0] <pathname>"); if
(access(argv[1], R_OK) < 0)              printf
("access error for %s", argv[1]); else
      printf("read access OK\n"); if
(open(argv[1], O_RDONLY) < 0)
      printf ("open error for %s", argv[1]); else
      printf("open for reading OK\n"); exit(0);
}
```

Example of umask function

```
#include <fcntl.h>
```

```
#define RWRWRW (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)

int main() {
umask(0);
if
(creat("foo
",
RWRWR
W) < 0)
      printf(
"creat error
for foo");
umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
if (creat("bar", RWRWRW) < 0)
      printf("creat error for bar"); exit(0);
}
```

Example of chmod function

```
int main() {
struct stat statbuf;
/* turn on set-group-ID and turn off group-execute */
if (stat("foo", &statbuf) < 0)
      err_sys("stat error for foo");
if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0)
          err_sys("chmod error for foo");
/* set absolute mode to "rw-r--r--" */
if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) <
0)    err_sys("chmod error for bar"); exit(0);
}
```

USP Lab – Session 4

1. Program to demonstrate the creation of hard links and the various properties of hard links

```
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h> int
main(int argc, char*argv[])
{
      if(argc==3)
      {
      printf("Hard linking %s and %s",argv[1],argv[2]);
if(link(argv[1],argv[2])==0)              printf("\nHard
link created");
      else
            printf("\nLink not created");
      }
      else if(argc==4)
```

```
    {
    printf("Soft linking %s and %s",argv[1],argv[2]);
if(symlink(argv[1],argv[2])==0)
printf("\nSoft link created");        else
            printf("\nLink not created");
    }
}
```

2. Program to demonstrate the creation of soft links and the various properties of hard links

3. Write a program to implement ls –li command which list the files in a specified directory. Your program should Print 5 attributes of files.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include  <dirent.h>  #include
<time.h>        int        main(int
argc,char* argv[])
{ struct dirent *dir;
struct stat mystat; DIR
*dp; dp = opendir(".");
if(dp) { while(dir =
readdir(dp))
{
stat(dir->d_name,&mystat); //
inode mode uid guid access_time
printf("%ld %o %d %d %s %s\n",
mystat.st_ino,mystat.st_mode,mystat.st_uid,mystat.st_gid,ctime(&mystat.st_atime),dir->d_name);
}
}
            }
```

4. Write a program to remove empty files from the given directory.

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

```
#include
<unistd.h>
#include
<dirent.h> int
main() {
DIR *dp;
struct dirent *dir; int fd,n; dp =
opendir("."); //open current directory
if(dp) { while((dir = readdir(dp)) !=
NULL)
{
fd = open(dir->d_name,O_RDWR,0777);
n = lseek(fd,0,SEEK_END);
if(!n) { unlink(dir-
>d_name);
}
}
}
}
```

5. Write a program to Copy access and modification time of a file to another file using utime function.

```
#include "apue.h"
#include <fcntl.h> #include
<utime.h>
int main(int argc, char *argv[])
{
        int i, fd; struct stat statbuf; struct utimbuf timebuf; for (i = 1; i <
        argc; i++) { if (stat(argv[i], &statbuf) < 0) { /* fetch current
        times */ printf("%s: stat error", argv[i]);
                        continue;
                } if ((fd = open(argv[i], O_RDWR | O_TRUNC)) < 0) { /*
        truncate */ printf ("%s: open error", argv[i]); continue;
        } close(fd);
timebuf.actime = statbuf.st_atime;
timebuf.modtime = statbuf.st_mtime;
if (utime(argv[i], &timebuf) < 0)
```

```
{ /* reset times */ printf("%s: utime
        error", argv[i]); continue;
} }
exit(0)
;
}
```

OR

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <utime.h>
#include <time.h> #include <fcntl.h> int main(int argc,char* argv[]) //copying
ctime and mtime of argv[2] to argv[1]
{ int fd; struct stat statbuf_1;
struct stat statbuf_2; struct
utimbuf times;
if(stat(argv[1],&statbuf_1)<0
) printf("Error!\n");
if(stat(argv[2],&statbuf_2)<0
)
printf("Error!\n");
printf("Before Copying ...\n");
printf("Access Time %s\nModification
Time%s\n",ctime(&statbuf_1.st_atime),ctime(&statbuf_1.st_mtime))
; times.modtime = statbuf_2.st_mtime; times.actime =
statbuf_2.st_mtime; if(utime(argv[1],&times)<0) printf("Error
copying time \n"); if(stat(argv[1],&statbuf_1)<0) printf("Error!\n");
printf("After Copying ...\n"); printf("Access Time %s\nModification
Time%s\n",ctime(&statbuf_1.st_atime),ctime(&statbuf_1.st_mtime));
}
```

Session 5

1. C program to simulate copy command by accepting the filenames from command line. Report all errors.

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h> #include<stdlib.h>
int main(int argc, char *argv[])
{char buf[100];  int
fd1,fd2; off_t size,ret,set;
ssize_t
readdata,writedata;
if(argc<3)
        printf("TOO FEW ARGUMENTS");

fd1=open(argv[1],O_RDONLY); //Open file 1 if(fd1==-1)
    printf("ERROR IN OPENING FILE: FILE DOES NOT EXIST \n"); else
        printf("FILE 1 OPENED SUCCESSFULLY \n");

fd2=open(argv[2],O_WRONLY | O_CREAT | O_TRUNC, 0666); //open file 2 in read-write mode, truncate its length to 0, create the file if it does not exist, 0666 is the access permission for the created file. order is important.
if(fd2==-1)
    printf("ERROR IN OPENING FILE"); else
        printf("FILE 2 OPENED SUCCESSFULLY \n");

size=lseek(fd1,0L,SEEK_END); //obtain the size of file 1 using lseek if(size==-
1)
    printf("ERROR: COULD NOT OBTAIN FILE SIZE \n"); else
        printf("FILE SIZE OF FILE 1 OBTAINED \n");

ret=lseek(fd1,0L,SEEK_SET); //change the current pointer to the beginning of the file if(ret==-1)
        printf("RETRACE FAILED \n");

readdata=read(fd1,buf,size); //read data equal to the size of the first file if(readdata==-1)
        printf("ERROR IN READING FILE CONTENTS \n");

writedata=write(fd2,buf,size);  //write the data to file 2 from buffer after read if(writedata!=size)
    printf("ERROR IN COPYING FILE"); else
    printf("FILE COPIED SUCCESSFULLY"); return 0;
}
```

2. C program to display environment variables using global variable environ.

```
#include<stdio.h>
#include<stdlib.h>
//extern char **environ;
```

```
int main() {
  int i = 1;

printf("test\n");
const char* s = getenv("PATH");
const char* p = getenv("PWD"); const
char* l = getenv("LOGNAME");
printf("PATH :%s\n",(s!=NULL)? s : "getenv returned NULL"); printf("PWD
:%s\n",(p!=NULL)? p : "getenv returned NULL"); printf("LOGNAME
:%s\n",(l!=NULL)? l : "getenv returned NULL");

printf("end test\n");

  return 0;
}
```

3. C program to display environment variables using APIs in C language.

4. C program to access the shell environment variable(SHELL). Print its value. Set it to /bin/csh and print the new value to standard output.

5. C program to illustrate effect of setjmp and longjmp functions on register, volatile and automatic variables.

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

static void f1(int, int, int, int);
static void f2(void);

static jmp_buf jmpbuffer; static
int globval;

int main(void)
{
        int autoval;
 register int regival;   volatile int
volaval;
        static int statval;

        globval = 1; autoval = 2; regival = 3; volaval = 4; statval = 5;
        if (setjmp(jmpbuffer) != 0)
        {
        printf("after longjmp:\n");
 printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n", globval, autoval,
regival, volaval, statval);
        exit(0);
        }
```

```
/*
* Change variables after setjmp, but before longjmp.
*/
                globval = 95; autoval = 96; regival = 97; volaval = 98;
                statval = 99;
        f1(autoval, regival, volaval, statval); /* never returns */
        exit(0);
}

static void f1(int i, int j, int k, int l)
{
                printf("in f1():\n");
 printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n", globval, i, j, k, l);
        globval=10000;
        j=10000;            f2();
}

static void f2(void)
{
                longjmp(jmpbuffer, 1);
}
```

6. C program to create a new process and demonstrate the working of fork function.

Session 6

1. C program to create a new process and demonstrate the working of fork function.

2. C program to create a new process and demonstrate the working of vfork function.

3. C program to demonstrate the working of wait function.

```
#include       "apue.h"
#include <sys/wait.h>
int main(void)
{
       pid_t pid;          int
status;          if ((pid =
fork()) < 0)
                     err_sys("fork error");
       else if (pid == 0) /* child */
       exit(7);

          if (wait(&status) != pid) /* wait for child */
                     err_sys("wait error");
          pr_exit(status); /* and print its status */

          if ((pid = fork()) < 0)
                     err_sys("fork error");
          else if (pid == 0) /* child */
                     abort(); /* generates SIGABRT */

          if (wait(&status) != pid) /* wait for child */
                     err_sys("wait error");
          pr_exit(status); /* and print its status */

          if ((pid = fork()) < 0)
                     err_sys("fork error");
       else if (pid == 0) /* child */                    status /= 0; /*
divide by 0 generates SIGFPE */        if (wait(&status) != pid)
/* wait for child */

                     err_sys("wait error");
       pr_exit(status); /* and print its status */   exit(0);
}
```

4. C program to avoid zombie status of a process.

```
#include<stdio.h>
#include<stdlib.h>
#include <sys/wait.h>
int
main(void) { pid_t
pid; if ((pid = fork())
< 0) { err_sys("fork
error");
```

```
}
else if (pid == 0) { /* first child */ if
((pid = fork()) < 0)
        err_sys("fork error");
else if (pid > 0)
        exit(0);
sleep(2);
printf("second child, parent pid = %ld\n",
(long)getppid()); exit(0); }
if (waitpid(pid, NULL, 0) != pid)
      err_sys("waitpid error");  exit(0);
}
```

5. C program to demonstrate race condition among parent and child processes.

6. C program to create a new process and demonstrate the working of exec function.

```
#include<stdio.h>
#include<stdlib.h>
//extern char **environ;

int main() {
int i = 1;

printf("test\n");
const char* s = getenv("PATH"); const char* p = getenv("PWD");
const char* l = getenv("LOGNAME"); printf("PATH
:%s\n",(s!=NULL)? s : "getenv returned NULL"); printf("PWD
:%s\n",(p!=NULL)? p : "getenv returned NULL");
printf("LOGNAME :%s\n",(l!=NULL)? l : "getenv returned
NULL");

printf("end test\n");

  return 0;
}
```

Session 7

/*Write a program (use signal system call)
i. which calls a signal handler on SIGINT signal and then reset the default action of the SIGINT signal
ii. Which ignores SIGINT signal and then reset the default action of SIGINT signal*/

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

void callback()
{
        printf("Intrrupt Received !\n");
        (void)signal(SIGINT,SIG_DFL);
} int
main()
{
        int ch,i=0;
        printf("Enter choice\n");
        scanf("%d",&ch);

        switch(ch)
        {
                case 1 :    (void)signal(SIGINT,callback);
                                        break;

                case 2 :    (void)signal(SIGINT,SIG_IGN);
                                        break;

        }
        while(1)
        {
                sleep(1);
                printf("Press CTRL+C ...\n");
                i++;
                if(i == 10 && ch == 2)
                        (void) signal(SIGINT,SIG_DFL);
        }
        return 0;
}
```

Write a program to create a coprocess

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<fcntl.h>
```

```
#define BUFFSIZE 100
int main() {
int n;
char buf[BUFFSIZE];
while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0) if
(write(STDOUT_FILENO, buf, n) != n)
printf("write error");
if (n < 0)
printf("read error");
exit(0);
}
```

/*Write a program using sigaction system call which calls a signal handler on SIGINT signal and then reset the default action of the SIGINT signal*/

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

struct sigaction sig;

void handler(int val)
{
	printf("Interrupt Received!\n");  sig.sa_handler =
SIG_DFL;    sigaction(SIGINT,&sig,0);
}

int main()
{
	sig.sa_flags = 0;
	sigemptyset(&sig.sa_mask);
		sigaddset(&sig.sa_mask,SIGINT); // listen only for SIGNIT
		sig.sa_handler = handler;

		sigaction(SIGINT,&sig,0);

		while(1)
		{
			printf("Progress is Happiness!\n");
			sleep(1);
		}
}
```

Program to establish client server communication using pipes

Client program

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

```
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>
#include <fcntl.h>

#define MAX 80

int main(int argc,char* argv[])
{
        char buffer[MAX];
        int in,out;
        int n;

    in = open("server_to_client",O_RDWR,0777);    out
= open("client_to_server",O_RDWR,0777);

        printf("Sending Message\n");
    n = strlen(argv[1]);
    write(out,argv[1],n);

    read(in,buffer,n);
    buffer[n]='\0';
        printf("Message Received : %s\n",buffer);

    close(in);
    close(out);
}

server program #include
<stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>

#define MAX 80

int main()
{  char  buffer[MAX];     int  in,out,n;
mkfifo("server_to_client",0777);
mkfifo("client_to_server",0777);

        while(1)
    {                   in = open("client_to_server",O_RDWR,0777);
    out = open("server_to_client",O_RDWR,0777);

                memset(buffer,0,MAX);
            printf("Waiting for Message\n");
            n = read(in,buffer,MAX);
    printf("Message : %s\n",buffer);
```

```
        buffer[0] = toupper(buffer[0]);
        printf("Sending Reply\n");
                        write(out,buffer,n);

                    close(in);
        close(out);
            }
            return 0;
}
```

Part B- Compiler Design Lab- Question Bank

Session 1

1. Write a C / C++ program to accept a C program and do error detection & correction for the following. **(CO1)**

   a) Check for un-terminated string constant in the input C program. i.e A string constant begins with double quotes and extends for more than one line. Intimate the error line numbers and the corrective actions to user.

2. Write a C / C++ program to accept a C program and do error detection & correction for the following. **(CO1)**

   a) Check for un- terminated multi line comment statement in your C program.

Session 2

4. Write a Lex program to accept a C program and do error detection & correction for the following. **(CO1)**

   a) Check for un-terminated string constant in the input C program. i.e A string constant begins with double quotes and extends for more than one line. Intimate the error line numbers and the corrective actions to user.

   b) Check for valid arithmetic expressions in the input C program. Report the errors in the statements to user.

Session 3

5. Write a Lex program to accept a C program and do the following error detection & correction.(CO1)

   a) Check for the valid usages of numerical constants in the input C program. Intimate the invalid usages to user.

   b) Check for valid declarative statements in your program. Intimate the invalid statements along with their line numbers to users.

Session 4

6. Write a Lex program to accept a C program and do the following error detection & correction.(CO2)

   a) Check for the valid if statement in the input C program. Report the errors to users.

b) Check for un- terminated multi line comment statement in your C program.

Session 5

7. Write Yacc program to accept a statement and do the following error detection.(CO2)
   a) Check for valid arithmetic expressions in the input C statement. Report the errors in the statements to user. Evaluate the arithmetic expression.
8. Write Yacc program to accept a statement and do the following error detection.
   a)Check for valid declarative statement. Intimate the errors to users.(CO2)

Session 6

9. Write Yacc program to accept a statement and do the following error detection.
   a) Check for the valid relational expression and evaluate the expression(CO2)

10. Write Yacc program to accept a statement and do the following error detection.
    a) Check for the valid logical expression and evaluate the expression(CO2)

Session 7

11.Write Yacc programs for the following grammar. Form valid and invalid input strings manually and give them as input for your executable code of yacc program and validate the input strings.(CO2)

a) S->SS+ | SS*|a

b) S->L=R | R

L-> *R | id

R->L

12.Write Yacc programs for the following grammar. Form valid and invalid input strings manually and give them as input for your executable code of yacc program and validate the input strings.

a)D>TL

T->int | float
L->L,id | id

b) S-> L.L | L

L->LB | B B->0 | 1

Solution - Sample codes:

1.

```
#include<stdio.h>
#include<string.h> int
main() { FILE *fp; int
strcheck=0;
int i; int lineno=0; int
string=0; char line[100];
int open,close; clrscr();
fp=fopen("file.txt", "r");
if(fp==NULL) {
 printf("File cant be opened\n");  exit(0);
}        printf("File        opened        correctly!\n");
while(fgets(line,    sizeof(line),    fp)!=NULL)    {
lineno++;   strcheck=0;   string=0;   open=close=0;
for(i=0;i<strlen(line);i++)  {  if(line[i]=="")


  {
string=1;      if(open==1&&close==0)
close=1;      else if(open==0&&close==0)
open=1;      else if(open==1&&close==1)
close=0;
```

```
  }
 }
if(open==1 &&close==0)
{    printf("\n Unterminated string in line %d. String Has to be closed", lineno);  strcheck=1;  }  else if(string==1 && strcheck==0){  printf("\n String usage in line %d is validated!",lineno);
 } }
return 0;
} file.txt
#include<stdio.h>
#include<conio.h>
int s[35]="gh"; void main(){
int a; char
c[10]="msrit",f[]="lk;
strlen("hjkl); a=a+/*b;
}
```

2.

```
#include<stdio.h>
#include<string.h> int main()
{ FILE *fp; int
commentcheck=0;
int i;
int lineno=0; int comment=0; char line[100]; int
open=0,close=0,openlineno,closelineno;
clrscr(); fp=fopen("file2.txt", "r");
if(fp==NULL) {
 printf("File cant be opened\n");  exit(0);
} printf("File opened correctly!\n");
while(fgets(line, sizeof(line), fp)!=NULL)
{  lineno++;  getch();
 commentcheck=0;  comment=0;  if(open==1&&close==0)
   printf("\n%s",line);  if(strstr(line,"/*")&&open==0)
```

```
 {
open=1;close=0;    comment=1;
openlineno=lineno;
printf("\n%s",line);
 }
 if(strstr(line,"*/")&&close==0&&open==1)
 {      closelineno=lineno;
if(open==1&&close==0)
    {
close=1;       open=0;       printf("\n Comment is displayed above!\nComment opened in line no
%d and closed in line no %d",openlineno,closelineno);
     }
 }
}  if(open==1
&&close==0)
 {   printf("\n Unterminated comment in begin in line no %d. It Has to be closed",
openlineno);   commentcheck=1;
 }
 else if(comment==1 && commentcheck==0){  printf("\n
Comment usage in line %d is validated!",lineno);
 } return
0;
}
```

---

file2.txt

```
#include<stdio.h>
#include<conio.h>
/
*
/
*
d
f
g
d
```

```
f
g
d d
f
g
d
f
g
*
/ i
n
t
s[
3
5
]
=
"
g
h
";
v
o
i
d m
ai
n
()
{
i
n
t
a;
/
*
char c[10]="msrit",f[]="lk;
*/strlen("hjkl);
/*dgdfgdfg*/
a=a+b; /*
fsdgdgds
sdgfsd sdfsdf

}
```

3a.

```
%{
#include<stdio.h>
int c=0;
FILE *fp;
%}
%%
\n { c++; }
["][a-zA-Z0-9]*["] {ECHO; printf(" Valid String in line number %d\n ",c+1);} ["][a-zA-Z0-9]*
{ ECHO; printf(" InValid String in line number %d\n ",c+1);}
```

```
. ;
%%
main() {
yyin=fopen("source.txt","r"); yylex();
fclose(yyin); }
```

```
source.txt #include<stdio.h>
#include<conio.h>
#include<string.h>  void
main()
{
int a,b,h; a=a+b; char
d[20]="d",h[67]="yu
; char c[10]="msrit";
a=a+/b+h;
strlen("msrit");
strlen("msr);
strcpy(c,"Bang
alore); b=b+*; }
```

3b..

```
%{
#include<stdio.h> int c=0; FILE *fp; %}
operator [-+*/] identifier [a-zA-Z][a-zA-Z0-9-
]* number [0-9]+ expression
({identifier}|{number}){operator}
({identifier}|{number})
%%
\n { c++; }
^"#".+ ;
^("int "|"float "|"char ").+ ;
"void main()" ;
{identifier}"="({expression}+";") { printf("Valid expression in line no :
%d\t",c+1);ECHO;printf("\n");}
{identifier}"="({number}|{identifier}";") { printf("Valid expression in line no :
%d\t",c+1);ECHO;printf("\n");}
({number}|([0-9]*[a-zA-Z0-9-]+))"="{expression}+ { printf("InValid expression in line no : %d;
Lvalue should satisfy the identifier rules\n",c+1);ECHO;printf("\n");}
{identifier}"=;" { printf("InValid expression in line no : %d; R-value required; Expression is needed
at right hand side of assignment operation\n",c+1);ECHO;printf("\n");}
{operator}{operator}+ {printf(" Invalid expression in line no: %d;More than one operator can't be
used in expression consequetively",c+1);ECHO;printf("\n");}
.|\n
;
%
%
ma
in(
) {
yyin=fopen("source.txt","r"); yylex(); fclose(yyin);
```

} ------

source.txt

```
#include<stdio.h>
#include<conio.h>
#include<string.h> void main()
{ int
a=1s,b,h;
a=a+b;
a=a+/b+h;
1a=7+j-;
a=;
b=b+*; }
```

4a.

```
%{

#include<stdio.h>  int c=0; %} number [0-

9]+(".")?[0-9]* invalid

[0-9]+(".")[0-9]*((".")[0-9]*)+

%%

\n {c++;}

{number} {printf("\nValid number in line number %d : ",c+1);ECHO;printf("\n");}

{number}[a-zA-Z0-9_]+ {printf("\nInvalid number in line number %d: Number followed with alphabets is
invalid",c+1);ECHO;printf("\n");}

{invalid} {printf("\nInvalid number in line number %d: Number with more than one decimal point sis
invalid",c+1);ECHO;printf("\n");}

. ; %%  void

main()

{ yyin = fopen("source.txt","r"); yylex();

fclose(yyin);

}
```

source.txt

```
#include<stdio.h>
#include<conio.h>
#include<string.h> void
main() { int a=56;
a=1b; a=a+5h;
a=a+4.5+5.
6.6;
}
```

4b.

```
%{

#include<stdio.h> int

c=0;

%}

%s DECLARE VAR   identifier [a-zA-

Z][a-zA-Z0-9-]* number [0-9]+[.]?[0-

9]* string ("\"")([a-zA-Z0-9]+)("\"")

%%

\n {c++;}

"int "|"float " {BEGIN DECLARE;}

<DECLARE>{identifier}("="{number})? {BEGIN VAR;}

<DECLARE>{identifier}("="{string}) {BEGIN VAR; printf("\n Invalid variable declaration in line no %d; string can't be assigned to integer or float variable:",c+1);ECHO;printf("\n");}

<VAR>";" {BEGIN 0;}

<VAR>{identifier}("="{number})? {}

<VAR>{identifier}("="{string}) {printf("\n Invalid variable declaration in line no %d; string can't be assigned to integer or float variable:",c+1);ECHO;printf("\n");}

<VAR>\n {BEGIN 0; c++;}

<VAR>"," {BEGIN DECLARE;}

<VAR>[,][,]+ {printf("\n Invalid usage of more than one comma in declaration in line no %d",c+1);BEGIN DECLARE;ECHO;printf("\n");}
. ; %% void

main()

{ yyin = fopen("source.txt","r"); yylex();


fclose(yyin);

}
```

source.txt

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

```
#include<conio.h>
#include<string.h> void
main() {
int a,b=78,g="78",,;
float c=5.6,h="fg";
sa=5; a=a+b; printf("\n
"); }
```

5a.

```
%{

#include<stdio.h> int

c=0,bc=0,fc=0;

FILE *fp;

%}

%s IF OPENP CLOSEP OPENF

%%

\n { c++; }

"if" {BEGIN IF;ECHO;bc=0;}

<IF>\n {c++;ECHO;printf("\n");}

<IF>"(" {BEGIN OPENP;ECHO;bc++;}

<IF>")" {BEGIN CLOSEP;ECHO;bc--;}

<OPENP>")" {ECHO;bc--;BEGIN CLOSEP;}
<OPENP>"(" {ECHO;bc++;}

<OPENP>. {ECHO;}

<CLOSEP>"{" {if(bc==0) {printf("condn is valid in line no %d\n",c+1);}              else

printf("condn invalid in line no %d;Paranthesis mismatch in condn\n",c+1);

            BEGIN OPENF;ECHO;printf("\n");fc++;}

<CLOSEP>"(" {BEGIN OPENP;bc++;ECHO;}

<CLOSEP>")" {ECHO;bc--;}

<CLOSEP>. {ECHO;}

<CLOSEP>\n {ECHO;printf("\n");c++;}

<OPENF>"}" {fc--;if(fc==0) BEGIN 0;;ECHO;printf("\n");}
```

```
<OPENF>. {ECHO;}

<OPENF>\n {ECHO;c++;}

.|\n ; %% main() {

yyin=fopen("source.txt","r"); yylex();

fclose(yyin);

}
```

source.txt

```
#include<stdio.h>
#include<conio.h
>
#include<string.h
> void main() {
int a,b=78;
if((a<5&&j<9) {
a=a+h; g=6+7;
a=a+b; printf("\n
");
} if(a<n)
{ h=j+k;
} if(a<n))
{ g=h+k;
}
}
```

5b.

```
%{
#include<stdio.h> int

c=0,oc=0;

FILE *fp;

%}

%s

COMM

ENT

%%

\n {c++;}

"/*" {BEGIN COMMENT;printf("\n comment begins in line no : %d\n",c);ECHO;oc=1;}

<COMMENT>"*/" {BEGIN 0;ECHO;oc=0;printf(": Comment ends in line no %d\n",c);}
```

```
<COMMENT>\n {c++;printf("\n");ECHO;}

<COMMENT>. {ECHO;}

. ; %% main() { yyin=fopen("source.txt","r"); yylex(); fclose(yyin); if(oc==1)

{  printf("\n comment is not closed till the end of file!");

}

}
```

source.txt

```
#include<stdio.h>
#include<conio.h>
#include<string.h
> /*dfddf*/ void
main()

{
/*vbhfghfgh
dfhfgh
fghgfhfg
fghfh */ int
a,b=78;
if((a<5&&j
<9) { a=a+h;
g=6+7;
a=a+b;
printf("\n
"); } /*
if(a<n) {
h=j+k; }
if(a<n))
{ g=h+k;
}
}
```

6.

Yacc:

```
%{

#include<stdio.h

> int flag=1; %}

%token id num

%left '(' ')'
```

```
%left '+' '-'

%left '/' '*'

%nonassoc UMINUS %% stmt: expression { printf("\n valid

exprn");}

; expression : '(' expression ')' | '(' expression

{printf("\n Syntax error:

Missing right paranthesis");}


| expression '+' expression {printf("\nplus recog!");$$=$1+$3;printf("\n %d",$ $);}

| expression '+' { printf ("\n Syntax error: Right operand is missing ");}

| expression '-' expression {printf("\nminus recog!");$$=$1-$3;printf("\n %d",$ $);}

| expression '-' { printf ("\n Syntax error: Right operand is missing ");}

| expression '*' expression {printf("\nMul recog!");$$=$1*$3;printf("\n %d",$ $);}
| expression '*' { printf ("\n Syntax error: Right operand is missing ");}

| expression '/' expression {printf("\ndivision recog!");if($3==0) printf("\ndivision cant be done, as divisor is
zero.");

                                                                    else {$$=$1+$3;printf("\n %d",$$);}}

| expression '/' { printf ("\n Syntax error: Right operand is missing ");} | expression '%' expression |

expression '%' { printf ("\n Syntax error: Right operand is missing ");}

| id

| num

; %% main() { printf(" Enter an arithmetic expression\n");

yyparse(); } yyerror() { printf(" Invalid arithmetic

Expression\n"); exit(1);

}
```

Lex:

```
%{

#include "y.tab.h"

#include<stdio.h>
```

```
#include<ctype.h

> extern int

yylval; int val;

%}

%%

[a-zA-Z][a-zA-Z0-9]* {printf("\n enter the value of variable
%s:",yytext);scanf("%d",&val);yylval=val;return id;}

[0-9]+[.]?[0-9]* {yylval=atoi(yytext);return num;}

[ \t] ;

\n {return 0;}

. {return yytext[0];}
%% int yywrap()

{ return

1;

}
```

7.

Lex:

```
%{

#include "y.tab.h"

#include<stdio.h>

int yylval;

%}

%%

"int"[ ]+ {return KEY;}

"float"[ ]+ {return KEY;}

"char"[ ]+ {return KEY;}

"double"[ ]+ {return KEY;}

"short"[ ]+ {return KEY;}
```

"long int"[ ]+ {return KEY;}

[a-zA_Z][a-zA-Z]*[0-9]* {return ID;}

[0-9]+ {return NUM;}

[\t] ;

[;] {return COLON;}

\n {return 0;}

. {return yytext[0];}

%%

int yywrap()

{
return 1;

}

Yacc:

%{

#include<stdio.h> int

flag=0;

%}

%token ID KEY COLON COMMA NUM

%%

stmt: list {printf("\n declration is validated!");}

;

list : KEY list

| list ',' list

| list ',',' {printf("Syntax error: consequtive commas used: invalid");exit(0);}

| list COL

| ID '[' NUM ']'

| ID '[' NUM '."]' { printf("\n float number cant be the size of an array");exit(0);}

| ID '[' ID ']' {printf("\n Size of an array should be an integer"); exit(0);}

| ID '[' ID { printf("\n close bracket missing in array declration");exit(0);}

| ID '[' {printf("\n size of array should be given");exit(0);}

| ID

;

COL: COLON

| COLON COL {printf("\n Syntax error: consequtive semicolon are used : invalid");exit(0);}

;

%%

main()

{
printf(" Enter valid

declaration\n"); yyparse(); }

yyerror() { printf(" Invalid

statement\n"); exit(1);

}

8.

Yacc:

%{

#include<stdi

o.h> int flag=1;

%}

%token id num

%%

stmt: expression { printf("\n valid relational exprn");}

;

expression : '(' expression ')'

| '(' expression {printf("\n Syntax error: Missing right paranthesis");}

| expression '<' expression {printf("\nless than recog!");($$=$1<$3);printf("\n %d",$$);}

| expression '<' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '>' expression {printf("\ngreater than recog!");($$=$1>$3);printf("\n %d",$$);}

| expression '>' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '<"=' expression {printf("\nless than or equal recog!");$$=($1<=$4);printf("\n %d",$ $);}

```
| expression '<''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '>''=' expression {printf("\ngreater than or equal!");$$=($1>=$4);printf("\n %d",$$);}

| expression '>''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '!''=' expression {printf("\nNot equal recog!");$$=($1!=$4);printf("\n %d",$$);}

| expression '!''=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}
| expression '=''=' expression {printf("\ndouble equal recog!");$$=($1==$4);printf("\n %d",$$);}

| expression '=''=' { printf ("\n Syntax error: Right operand is missing

");exit(0);} | id |

num

;

;

%%

m a

i

n

(

) { printf(" Enter relational

expression\n"); yyparse(); }


yyerror() { printf(" Invalid relational

expression\n"); exit(1);

}
```

Lex:

```
%{

#include "y.tab.h"

#include<stdio.h>

#include<ctype.h>

extern int yylval;  int

val;

%}
```

```
%%
[a-zA-Z][a-zA-Z0-9]* {printf("\n enter the value of variable
%s:",yytext);scanf("%d",&val);yylval=val;return id;}

[0-9]+[.]?[0-9]* {yylval=atoi(yytext);return num;}

[ \t] ;

\n {return 0;}

. {return yytext[0];}

%%

int

yywra

p() { return

1;

}
```

9.

Lex:

```
%{

#include "y.tab.h"

#include<stdio.h>

#include<ctype.h>

extern int yylval; int val;

%}

%%

[a-zA-Z][a-zA-Z0-9]* {printf("\n enter the value of variable
%s:",yytext);scanf("%d",&val);yylval=val;return id;}

[0-9]+[.]?[0-9]* {yylval=atoi(yytext);return num;}

[ \t] ;

\n {return 0;}

. {return yytext[0];}

%%
int

yywra
```

```
p() { return
1;
}
```

Yacc:

```
%{
#include<stdi
o.h> int flag=1;
%}
%token id num
%%
stmt: expression { printf("\n valid logical exprn : evaluated result is %d",$1);}
;
expression : '(' expression ')' { $$=$2;printf("\n value : %d",$$);}
| '(' expression {printf("\n Syntax error: Missing right paranthesis");exit(0);}
| expression '&"&' expression  {printf("\nlogical and recog!");$$=(($1)&&($4));printf("\n %d",$ $);}
| expression '&"&' {printf("Syntax error: Right operand is missing ");exit(0);}
| expression '|"|' expression  {printf("\nlogical or recog!");$$=($1||$4);printf("\n %d",$$);}
| expression '|"|'  {printf("Syntax error: Right operand is missing ");exit(0);}
| '!' expression  {printf("\nlogical not recog!");$$=!($2);printf("\n %d",$$);}
| '!'   {printf("Syntax error: Right operand is missing ");exit(0);}
| expression '<' expression {printf("\nless than recog!");$$=($1<$3);printf("\n %d",$$);}
| expression '<' { printf ("\n Syntax error: Right operand is missing ");exit(0);}
| expression '>' expression {printf("\ngreater than recog!");$$=($1>$3);printf("\n %d",$$);}
| expression '>' { printf ("\n Syntax error: Right operand is missing ");exit(0);}
| expression '<"=' expression {printf("\nless than or equal recog!");$$=($1<=$4);printf("\n %d",$ $);}
| expression '<"=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}
| expression '>"=' expression {printf("\ngreater than or equal!");$$=($1>=$4);printf("\n %d",$$);}
| expression '>"=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}
```

```
| expression '!'=' expression {printf("\nNot equal recog!");$$=($1!=$4);printf("\n %d",$$);}

| expression '!'=' { printf ("\n Syntax error: Right operand is missing ");exit(0);}

| expression '='=' expression {printf("\ndouble equal recog!");$$=($1==$4);printf("\n %d",$$);}

| expression '='=' { printf ("\n Syntax error: Right operand is missing
");exit(0);} | id |
num

;

;

%%

main() { printf(" Enter

logical expression\n");

yyparse(); }


yyerror() { printf(" Invalid logical
expression\n"); exit(1); }

10a.

LEX:

%{

#include<stdio.h>

#include"y.tab.h" int

yylval;

%}

%%

[a] {return ID;}

[\t]
. {return yytext[0];}

[\n] {return 0;}

[ ] {return 0;}

%%

int yywrap()
```

{ return

1; }

YACC

%{

#include<stdio.h>

%}

%token ID

%%

exp:exp exp '*'

exp:exp exp '+' exp:ID

%% m

a

i

n

(

) {

printf("enter the expression for the grammar \n S-->SS+ | SS* | a"); yyparse(); printf("valid experession\n"); } void yyerror() { printf("Invalid expression\n"); exit(1);

}

10b.

LEX:

%{

#include<stdi

o.h>

#include"y.ta

b.h" int yylval;

%}

%%

[a-zA-Z][a-zA-Z0-9]* {return ID;}

[\t]

. {return yytext[0];}

```
[\n] {return 0;}
[ ] {return 0;}
%%
int yywrap()
{ return 1;
}
YACC:
%{
#include<stdio.h>  int
flag=0;
%}
%token ID
%%
S:S '=' L|R {flag++;}
;
L: '*' R| ID;
R:L;
%% m
a
i
n
(
) {
printf("enter the expression for the grammar  \n S->S=L|R\nL->*R|ID\nR-
>L"); yyparse(); if(flag)
printf("valid experession\n");
else yyerror(); } void
yyerror() { printf("Invalid
expression\n"); exit(1);
}
```

11a.

LEX:

```
%{
#include<stdio.h>
#include"y.tab.h" int
yylval;
%}
%%
"int"|"float" {return type;}
[a-zA-Z][a-zA-Z0-9]* {return ID;}
[\t]
. {return yytext[0];}
[\n] {return 0;}
[ ] {return 0;}
%%
int yywrap() { return
1;
}
```

YACC:

```
%{
#include<stdi
o.h> int flag=0;
%}
%token type ID
%%
D: T ' ' L;
T: type;
L: L ',' ID| ID;
%%
```

```
main() { printf("enter the expression for the
grammar 4\n"); yyparse(); printf("valid
experession\n");
}  void
yyerror
()
{%{
#include<stdi
o.h> int flag=0;
%}
%token type ID
%%
D: T ' ' L;
T: type;
L: L ',' ID| ID;
%%
main() {
printf("enter the expression for the grammar 4\n");
yyparse(); printf("valid experession\n");
} void yyerror
() { printf("Invalid
expression\n"); exit(1); }
printf("Invalid expression\n");
exit(1);
}
```

11b.

LEX:

```
%{
#include<stdio.h>
#include"y.tab.h" int
yylval;
```

```
%}

%%

[0] {return ZERO;} [1]

{return ONE;}

[\t]

. {return yytext[0];}

[\n] {return 0;}
[ ] {return 0;}

%%

int

yywra

p() { return

1;

}

YACC:

%{

#include<stdio.h>

int flag=0;

%}

%token ZERO ONE

%%

S: L '.' L| L;

L: L B| B;

B: ZERO | ONE;

%%

main() { printf("enter the expression for the

grammar 5\n"); yyparse(); printf("valid

experession\n");

} void yyerror

() {
```

```
printf("Invalid expression\n"); exit(1);
}
```

RAMAIAH Institute of Technology

Department of Computer Science & Engineering

QUESTION BANK FOR VI SEMESTER (Term: Jan-May 2018)

Unix System Programming & Compiler Design Laboratory (CSL66)

I.A. Marks : 50 **Exam Hours: 03**

Credits: 0:0:1:0 **Exam Marks: 50**

Part A - Unix System Programming

Sl. No.	Question	Mapped CO
1.	Write a C program to display the file content in reverse order using lseek system call.	3
2.	Write a C program to implement ls –li command which list the files in a specified directory. Your program should Print 5 attributes of files.	3
3.	Write a C program to remove empty files from the given directory.	3
4.	Write a program to Copy access and modification time of a file to another file using utime function.	3

5.	Write a C program to illustrate effect of setjmp and longjmp functions on register and volatile variables.	3
6.	Write a C program to demonstrate race condition among parent and child processes.	3
7.	Write a C program to avoid zombie status of a process.	3
8.	Write a C program such that it initializes itself as a daemon Process.	3
9.	Write a C program using sigaction system call which calls a signal handler on SIGINT signal and then reset the default action of the SIGINT signal.	3
10	Write a C program (use signal system call) i. which calls a signal handler on SIGINT signal and then reset the default action of the SIGINT signal ii. Which ignores SIGINT signal and then reset the default action of SIGINT signal	3
11	Write and execute a C program to create co-process which performs Concatenation of two strings.	3
12	Write a C program to implement client - server communication using pipes. Client should send a string to the server and it should change the first character of the string to uppercase and sends it back to client.	3

Part B - Compiler Design

Sl. No.	Questions	Mapped CO
1.	Write a C / C++ program to accept a C program and perform error detection & correction for the following: a) Check for un-terminated string constant and single character constant in the input C program. i.e A string constant begins with double quotes and extends to more than one line. b) Report the error line numbers and the corrective actions to user.	1

2.	Write a C / C++ program to accept a C program and perform error detection & correction , indicate the user for the following : a) Check whether the multi-line comment statement is terminated correctly or not. b) Check whether the single line comment statement is existing in your C program and report the line numbers to the user.	1
3.	Write a Lex program to accept a C program and perform error detection & correction for the following: a) Check for valid arithmetic and relational expressions in the input C program b) Recognize increment and decrement operations also.	1
	c) Report the errors in the statements' to user.	
4.	Write a Lex program to accept a C program and perform the following error detection & correction: a) Check the validity of "*structure"* declarative statements in your program. b) Indicate the invalid statements along with their line numbers to users.	1
5.	Write a Lex program to accept a C program and perform the following error detection & correction: a) Check for the valid "*if ….else if…else"* statement in the input C program. b) Report the errors to users.	1
6.	Write Yacc and Lex programs to accept an arithmetic expression and perform the following error detection: a) Check the validity of the "*arithmetic expressions"* in the input C statement. b) Report the errors in the statements to user. c) Evaluate the arithmetic expression. d) Recognize increment and decrement operators involved in the expressions.	2
7.	Write Yacc and Lex programs to accept a declarative statement and perform the following error detection: a) Check the validity of the "*declarative"* statement. b) Recognize array declarations of any dimension. c) Report the errors to users.	2
8.	Write Yacc and Lex programs to accept a relational expression and perform the following error detection: a) Check the validity of the "*relational"* expression and evaluate the expression. Note: Relational expression can have arithmetic expressions embedded in it.	2

9.	Write Yacc and Lex programs to accept a logical expression and perform the following error detection: a) Check for the validity of the logical expression and evaluate it. Note: Logical expression can have relational and arithmetic expressions with in it.	2
10	Write Yacc and Lex programs for the following grammar: a) Test the executable code of Yacc program by giving valid and invalid strings as input.	2
	Grammar : S→SS+ | SS*| (S) |a	
11	Write Yacc and Lex programs for the following grammar: a) Test the executable code of Yacc program by giving valid and invalid strings as input. *Grammar :* S→L=R | R L→ *R | id |num R→L	2
12	Write Yacc and Lex programs for the following grammar: a) Test the executable code of Yacc program by giving valid and invalid strings as input. *Grammar :* D→TL T→int | float | long int | double | static int | register int L→L,id | id	2

Note:

1. Conduction, Execution and Result : 40 Marks

Part A : 20 Marks (Write-up : 5 marks, Execution : 15 marks)

Part B : 20 Marks (Write-up : 5 marks, Execution : 15 marks)

Viva-voce : 10 Marks

2. For Change of question

Part A OR Part B : -5 Marks

Part A AND Part B : -10 Marks