Solution/Model Answer to IAT-3
Programming in C & data Structure
2nd Sem I & J Section
-Dr. P. N. Singh, Professor(CSE)

---

1 a) Develop(write) a C program to display contents of a text file. (File name is given by user. Display appropriate error message if the file does not exist.) 05m

Ans:

```
/* dikhao.c - To display the contents of a text file */
#include <stdio.h>
int main()
{
   FILE *fp;
   char fname[64],c;
   printf("Enter file name : ");
   scanf("%s",fname);
   fp=fopen(fname,"r");
   if(!fp)
   {
       printf("File %s does not exist.\n",fname);
       return (99);
   }
   printf("Contents of %s file:\n");
   while(!feof(fp))
   {
     c=fgetc(fp);
     putchar(c);
   }
   fclose(fp);
   return (0);
}
```

1 b) Write a C program to copy contents of a file to another file. 05

```
/* kopy.c */
#include <stdio.h>
int main()
{
  FILE *fp1, *fp2;
  char sfname[64],tfname[64],c;
  printf("Enter source file name : ");
  scanf("%s",sfname);
  fp1=fopen(sfname,"r"); /*opening source file with read permission.*/
  if(!fp1){
      printf("File %s does not exist\n",sfname);
      return (99);
  }
```

```
  printf("Enter target file name : ");
  scanf("%s",tfname);
  fp2=fopen(tfname,"w"); /* opening target file with write permission*/
  while(!feof(fp1))
  {
     c=fgetc(fp1);  /* copying character by character in target file*/
     fputc(c,fp2);
  }
  fclose(fp1);
  fclose(fp2);
  return (0);
}
```

2. Define file handling functions fgetc(), fputc(), fprintf(), fscanf(), fopen() with syntax and examples. 10

Ans:

fgetc( ) – reads a character from a file and moved the file pointer one character forward:
example:
c=fgetc(fp);
Reads a character from file pointed by file pointer fp and stores in character variable c

fputc( ) – Writes a character to a file and moved the file pointer one character forward:
example:
fputc(c, fp);
Writes character of c variable in file pointed by fp and moves the file pointer for next
Reads a character from file pointed by file pointer fp and stores in character variable c

fprintf( ) – formatted output to a file
Syntax: fprintf (fp, "control string", var1,var2,..varn);
file pointer variable,constants, strings
Example: fprintf(f1, "%s %d %f\n ", name, age, 7.8);

fscanf( ) – formatted input from file
Syntax: fscanf(fp, "control string", &var1, &var2,….&varn);
Example: fscanf(f2, " %s %d", item, &qtty);

fopen( ) – opens file different modes
Syntax:
FILE *fp;
fopen(filename,"mode")
Example:
fp=fopen("student.txt","r");
File opening modes:

r-read only
w-write
a-append

3 a)

What are following declarations:

i. int *a[5];
ii. int (*p)[5];
iii. int *func( );
iv. int (*func)( ); 5m

Ans:

i. int *a[5];
/* **Array of pointers** - an array having all 5 element addresses only */
ii. int (*p)[5];
/***Pointer to arrays** – Here p is pointer to 2-d array where 5 is number of columns */
iii. int *func( );
/***Pointer to return type of function** – The function func() will return address */
iv. int (*func)( );
/* **Pointer to function** – here func pointer will keep address of other function */

3 b) Implement call by reference in a C program using pointers. 5 m

```
/* Call by reference using pointers */
#include <stdio.h>
void swap(int *p, int *q) /* here p and q are receiving addresses */
{
      int temp = *p;
      *p = *q;
      *q = temp;
}

int main()
{
      int num1, num2;
      printf("Enter number 1 : ");
      scanf("%d", &num1);
      printf("Enter number 2 : ");
      scanf("%d", &num2);
      swap(&num1, &num2); /& call by reference */
      printf("Numbers swapped: num1 = %d num2 = %d\n", num1, num2);
      return (0);
}
```

4) Explain dynamic memory allocation & de-allocation by built-in functions 10m

Ans:

Memory is allocated/de-allocated dynamically at run time as and when required. Dynamic memory management refers to manual memory management. This allows a programmer to obtain more memory when required and release it when not necessary.

In C there are 4 library functions defined under stdlib.h or alloc.h for the same

malloc( ) - Allocates a block of memory of specified size in bytes and return a pointer of type void which can be casted into pointer of any form
Syntax:
ptr = (cast-type*) malloc(byte-size)
Example for p is 2-d array:
p= malloc(sizeof(int)*rows*cols)

If p is cast type then one example
p = (int *)malloc(sizeof(int)*100);

calloc() – Contiguous allocation – Allocates multiple blocks of same size – requires two arguments
calloc(number_of_blocks, size_in_bytes_for_each_block)
ptr = (cast-type*)calloc(n, element-size);
ptr = (float*) calloc(25, sizeof(float));

realloc( ) : changes/reallocates the size of previously allocated space by malloc() or calloc()
Syntax:
ptr = realloc(ptr, newsize);

if size1 is allocated by malloc: ptr = (int*) malloc(size1 * sizeof(int));
then to rallocate for size2
ptr = realloc(ptr, size2);

free() – To deallocate the previously allocated space
Syntax/example:
free(ptr)
This statement frees the space allocated in the memory pointed by ptr.

5. Explain any 5 pre-processor directives with example. Write 1 example program for macros. 10m

Ans:

A pre-processor represents pre-compilation process. These directives are preceded with # (**and there is no semicolon at the end**).

C has the special feature of pre-processor directives. The pre-processor processes the C source code before it passes to the compiler. Pre-processor directives are executed before the C source code passes through the compiler.

Example of two pre-processor directives:

i) #include

This pre-processor is used to include another file. File name is written in angle brackets or within double quotes.

```
#include <stdio.h>
#include "SORT.H"
```

File name enclosed within angle brackets are searched in standard directories. File name enclosed within double quotes are searched first in the current directory and if not found then it is searched in the standard directories.

ii) #define

#define directive is used to define the symbolic constant or macro name with macro expression.

#define symconstant value

#define macroname macroexpression

Example

```
#define     PI      3.141
```

Here the C pre-processor searches for the symbolic constant PI the source code and substitutes 3.141 each time it is encountered.

As the function, a macro can have argument. The argument of the macro name is replaced with actual argument of the macro name, which is encountered in the program.

Examples:

```
#define     MAX(x,y)     ( (x) > (y) ? (x) : (y) )
#define     CUBE(x)      (x)*(x)*(x)
```

iii) #ifdef

#ifdef preprocessor directives checks whether a macro is defined by #define. If yes, it executes the code:

Example:

```
#ifdef PI
        Printf("PI symbolic constant is already defined\n");
#endif
```

iV) #ifndef

#ifndef preprocessor directives checks whether a macro is not defined by #define. If yes, it executes the code:

Example:

```
#ifndef PI
        #define PI 3.14
#endif
```

V) #pragma

#pragma preprocessor directives schedules the execution of function before and after main() function by clauses startup and exit:

Example:

#pragma startup func2

#pragma exit func1

Here func2() function will execute before main() function and func1() function will execute after main() function. There is no need to call these function from main()

Example program for macros:

```
#include <stdio.h>
#define cube(x) x*x*x
int main()
{
   printf("%d", cube(5+2))
   return (0);
}
```

It gives the output 27 and not cube of 7 i.e. 343 because a macro is expanded in which way it is defined 5+2 * 5+2 * 5+2 and it gives 27. We can say it is blunder definition of macro. It should be defined as:

#define cube(x) (x)*(x)*(x)

Then it will give correct result

6. Explain push() and pop() operations of stack data structure. 10m

Stack is a LIFO (Last In First Out) data structure. Items can be inserted or deleted from one end(top) only

```
/*stack push() operation given in a function */
void push(int stack[], int item)
{
     if(top==size-1)
          print("Stack is full-overflow.\n");
      else
      {
          top++;
          stack[top]=item;
      }
}

/*stack push() operation given in a function */
int pop(int stack[])
{
      int d;
```

```
      if(top==-1)
      {
           printf("Stack is empty-underflow\n");
           return (NULL);
      }
      else
      {
           d=stack[top];
           stack[top]=NULL;
           top--;
           return (d);
      }
}
```

7 a) What do you mean by a linked list? Define structure of a linked list. 5M

Ans:
A linked list is a dynamic data structure where member of a structure is pointing to structure itself. It is also known as self referential structure.

New nodes can be added as and when required allocating the memory at run time. It requires extra space for address of the linked/next node:

| data | next---|---->| data | next---|---->| data | next---|----> NULL

```
#structure definition of a linked list
struct LIST
{
     int data;
     struct LIST *next;
};
typedef struct LIST node;
```

7 b) Explain primitive data types & derived data types. 5m

Ans:

A data type is a classification of data, which can store a specific type of information.

Primitive data types are primary, fundamental or predefined types of data, which are supported by the programming language. In C primitive data types are:

- char
- int

- float
- double

Programmers can use these data types when creating variables in their programs. For example, a programmer may create a variable called marks and define it as a int data type type.

Examples:

```
int marks;
double num = 0.00;
char c;
float sum;
```

Derived data types are non-primitive data types or composed data types from primary data types. Non-primitive data types are not defined by the programming language, but are instead created by the programmer. Arrays, pointers, structures are examples of derived data type.

Examples of derived/non-primitive data types:

```
int a[10]; /* array of 10 integers */
struct student
{
        int roll;
        char name[20];
        int marks;
};
struct student s1,s2; /* s1 and s2 are structure variables or objects */
int *p; /* p is pointer variable to keep address of integer data */
```

8. Write a C program to count frequency of vowels & total number of consonants in a sentence (given by user) 10m

Ans:

```
/* to count frequency of vowels and total count of consonants */

#include <stdio.h>
#include <ctype.h>
int main()
{
   char sent[50],c;
   int x,a,e,i,o,u,conso;
   printf("Enter sentence : ");
   scanf(" %[^\n]",sent); /* to read string with blanks */
   a=e=i=o=u=conso=0;
```

```
  for(x=0;sent[x]!='\0';x++)
  {
    if(isalpha(sent[x]))  /* checks alphabets only */
    {
      c=tolower(sent[x]);
      if(c=='a') a++;
      else if(c=='e') e++;
      else if(c=='i') i++;
      else if(c=='o') o++;
      else if(c=='u') u++;
      else conso++;
    }
  }
  printf("a=%d,e=%d,i=%d,o=%d,u=%d,consonants=%d\n",a,e,i,o,u,conso);
  return (0);
}
```

Expected output:
Enter sentence : you are a good boy
a=2,e=1,i=0,o=4,u=1,consonants=6