Global Institute of Technology, Jaipur

ITS-1, IT Park, EPIP, Sitapura Jaipur 302022 (Rajasthan)

Solution III sem University Examination 2019

IFY1-06 I Sem / I Year

RTU Paper Solution

Branch –Computer Science Engineering

Subject Name –Programming for Problem Solving

Paper Code – 1FY1-06

Date of Exam – December 2019

Global Institute of Technology, Jaipur

ITS-1, IT Park, EPIP, Sitapura Jaipur 302022 (Rajasthan)

Solution III sem University Examination 2019

IFY1-06 I Sem / I Year

1E2406

Roll No. _____ Total No of Pages: 4

1E2406

B. Tech. I - Sem. (Main/Back) Exam., Dec. 2019

1FY1 – 06 Programming for Problem Solving

Time: 2 Hours **Maximum Marks: 80**

Min. Passing Marks: 28

Instructions to Candidates:

Attempt all five questions from Part A, four questions out of six questions from Part B and two questions out of three from Part C.

Schematic diagrams must be shown wherever necessary. Any data you feel missing may suitably be assumed and stated clearly. Units of quantities used /calculated must be stated clearly.

Use of following supporting material is permitted during examination. (Mentioned in form No. 205)

1. NIL 2. NIL

PART – A

(Answer should be given up to 25 words only) **[5×2=10]**

All questions are compulsory

Q.1 Draw the block diagram of a Computer. Explain its components. Differentiate between primary & secondary storage.

Q.2 Draw a flowchart to find the maximum among the three numbers.

Q.3 What is an algorithm? Write an algorithm to print even numbers from 2 to 100.

Q.4 Write the format of the following functions -

(a) fseek

(b) fopen

(c) getch

(d) getchar

Q.5 Differentiate between Array, Structures & Unions.

PART – B

(Analytical/Problem solving questions) **[4×10=40]**

Attempt any four questions

Q.1 (a) Write a program in C to print inverted half pyramid –

```
* * * * *
* * * *
* * *
* *
*
```

(b) Write a program in C to check Armstrong number of n digits.

Q.2 (a) What will be the output of the following "C" code?

```
#include

void main () {

int i = -1,  j = -1, k = 0, l = 2, m;

m = i++ &&j++ &&k++ || l++;

printf ("%d%d%d%d%d", i,j,k,l,m);

}
```

(b) Explain the terms Identifiers and Data Types in C. Explain the rules of Identifier declaration in C language. List all the Data Types of C with their storage size and storage format.

Q.3 Write a Pseudo Code to Multiply 2[3×3] Matrices and Transpose the output of Multiplication.

Q.4 Write a program to swap two values using functions, use 2 cases -

(a) call by value and

(b) call by reference

Q.5 Explain the following conditional statements with proper syntax and example –

(a) If- else statement

(b) Switch Case

(c) While Loop

(d) Do-While Loop

Q.6 Solve the following –

(a) Convert 253.64 from base 10 to base 8

(b) The given hexadecimal number (1E.53) base 16 is equivalent to (_____) base 8

(c) Convert the binary number (01011.1011)2 into decimal

(d) Find r's and (r-1)'s complement of 5308

(e) Solve the following (10110.01)2 – (11010.10)2, using (r-1)'s compliment

PART – C

(Descriptive/Analytical/Problem Solving/Design Questions) **[2×15=30]**

Attempt any two questions

Q.1 Write a C program to find the sum of the series –

$$S = 1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^n}{n!}$$

Q.2 Programming on structures/pointers –

(a) Define a structure data type called time_struct containing 3 members called hour, minute and second. Develop a program that would assign values to the individual members and display the time in the form 16:40:30

(b) Write a program in C to Swap Three (3) numbers using pointers

Q.3 Explain the concept of File handling in C language. Write a Program in C language to copy the data from source file to destination file and display the same on console.

Global Institute of Technology, Jaipur

ITS-1, IT Park, EPIP, Sitapura Jaipur 302022 (Rajasthan)

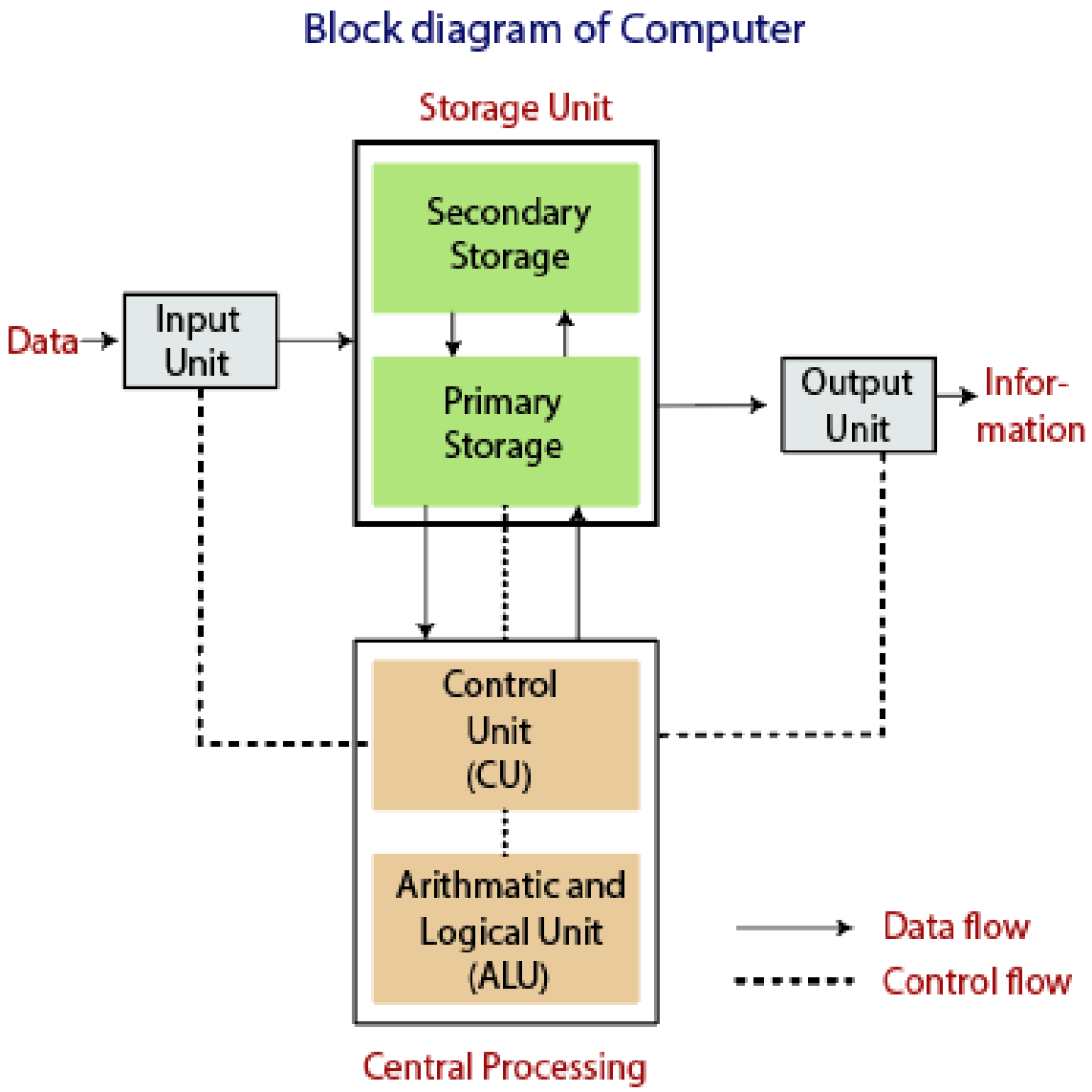
Solution III sem University Examination 2019

IFY1-06 I Sem / I Year

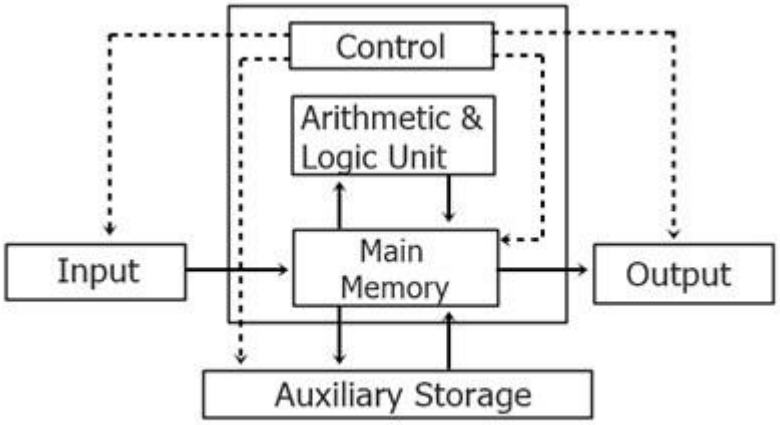
Part A

1 Block Diagram of Computer

Block diagram of a computer gives you the pictorial representation of a computer that how it works inside. Or you can say that, in computer's block diagram, we will see how computer works from feeding the data to getting the result.



In the diagram, both control (control unit or CU) and arithmetic & logic unit (ALU) combinely called as Central Processing Unit (CPU).



Block Diagram of Computer

Let's describe about all the parts as included in the above diagram one by one.

The Processor Unit (CPU)
It is the brain of the computer system.

All major calculation and comparisons are made inside the CPU and it is also responsible for activation and controlling the operation of other unit.

This unit consists of two major components, that are arithmetic logic unit (ALU) and control unit (CU).

Arithmetic Logic Unit (ALU)
Here arithmetic logic unit performs all arithmetic operations such as addition, subtraction, multiplication and division. It also uses logic operation for comparison.

Control Unit (CU)
And the control unit of a CPU controls the entire operation of the computer. It also controls all devices such as memory, input/output devices connected to the CPU.

CU fetches instructions from memory, decodes the instruction, interprets the instruction to know what the task are to be performed and sends suitable control signals to the other components to perform for the necessary steps to executes the instruction.

Input/Output Unit
The input/output unit consists of devices used to transmit information between the external world and computer memory.

The information fed through the input unit is stored in computer's memory for processing and the final result stored in memory can be recorded or display on the output medium.

Memory Unit
Memory unit is an essential component of a digital computer. It is where all data

intermediate and find results are stored.

The data read from the main storage or an input unit are transferred to the computer's memory where they are available for processing.

This memory unit is used to hold the instructions to be executed and data to be processes.

Disk Storage Unit
Data and instruction enters into a computer system through input device have to stored inside the computer before actual processing start.

Two types of storage unit are primary and secondary storage unit.

Primary Storage Unit
Primary memory has direct link with input unit and output unit. It stores the input data, calculation result.

Secondary Storage Unit
The primary storage is not able to store data permanently for future use. So some other types of storage technology is required to store the data permanently for long time, it is called secondary or auxiliary storage.

Primary Memory and Secondary Storage

Primary memory is the main memory of the computer which can be directly accessed by the central processing unit, whereas secondary memory refers to the external storage device which can be used to store data or information permanently.
Difference between Primary and Secondary Memory

Basics of Primary and Secondary Memory
Memory plays a critical part in computers to store and retrieve data. Computer memory is categorized into primary and secondary memory. While primary memory is the main memory of the computer which is used to store data or information temporarily, whereas secondary memory refers to external storage devices that are used to store data or information permanently.

Access of Primary and Secondary Memory
Primary memory holds only those data or instructions which the computer is currently processing allowing the processor to access running applications and services that are stored temporarily in a specific memory address. Secondary memory, on the other hand, is persistent in nature which means instructions are transferred to the main memory first and then re-routed to the central processing unit.

Data in Primary and Secondary Memory
In primary memory, data is directly accessed by the processing unit and it resides in the main memory until processing. Information and data are stored in semiconductor chips so they have a limited storage capacity. In secondary memory, information is stored in external storage devices and they cannot be directly accessed by the processing unit.

Global Institute of Technology, Jaipur

ITS-1, IT Park, EPIP, Sitapura Jaipur 302022 (Rajasthan)

Solution III sem University Examination 2019

IFY1-06 I Sem / I Year

Nature of Primary and Secondary Memory

Primary memory is volatile in nature which means data or information stored in the main memory is temporarily which may lead to loss of data in case of power failure and it cannot be retained. On the contrary, secondary memory is non-volatile in nature which means information is stored permanently with no data loss in case of power failure. Data is intact unless the user erases it intentionally.

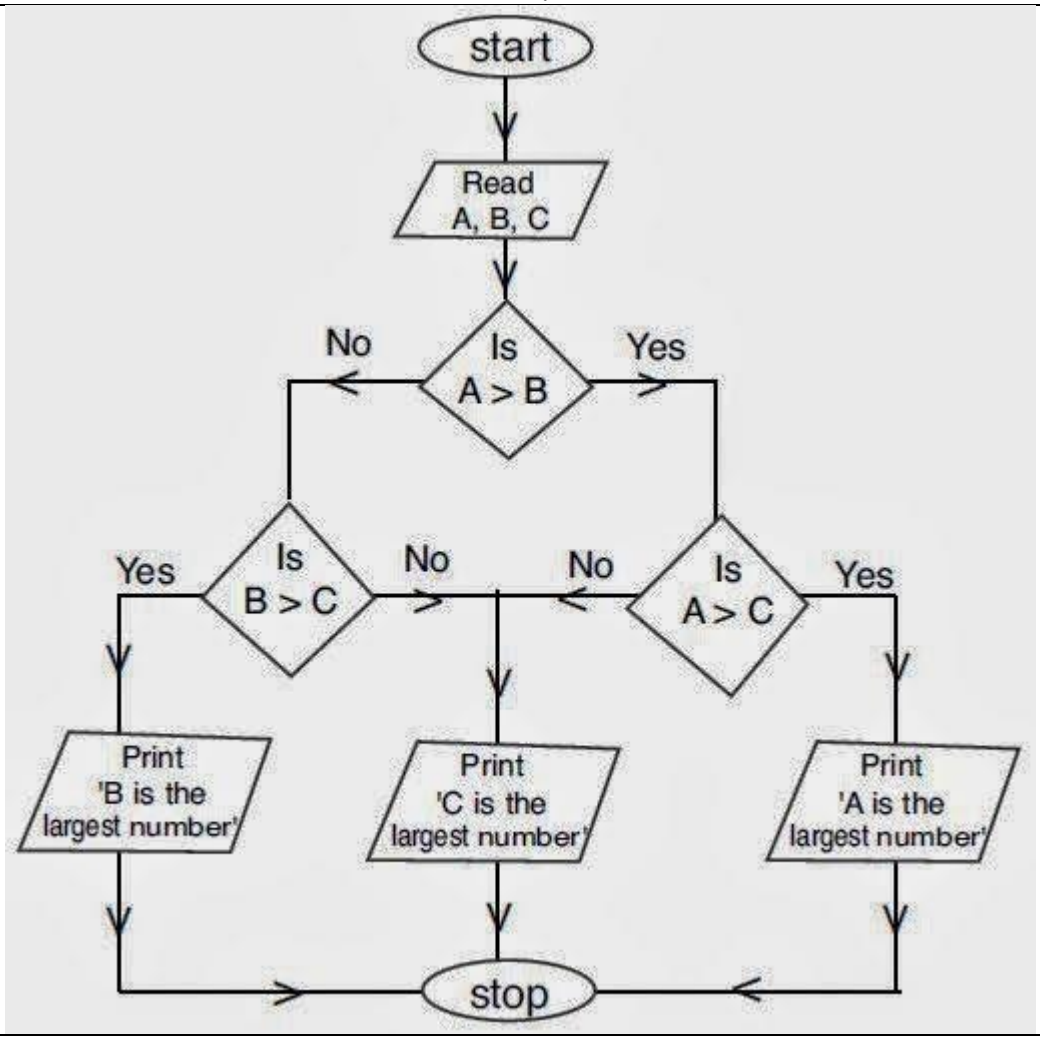
Devices for of Primary and Secondary Memory

Primary memory can also be referred to as RAM, short for Random Access Memory, because of the random selection of memory addresses. RAM holds data in a uniform manner and it can be lost when power fails. Secondary memory refers to external storage devices such as hard disk, optical disk, compact disk, flash drives, magnetic tapes, etc. They are high-storage devices with substantial storage capacities, in the range of gigabytes to terabytes.

Speed of Primary and Secondary Memory

In primary memory, applications and instructions are stored in the main memory which makes them relatively faster to access via data bus. Processor is able to retrieve data faster than it does with secondary memory, which acts more like a backup memory to store data in external storage devices.

2



4 **(a)**

The C library function int fseek(FILE *stream, long int offset, int whence) sets the file position of the stream to the given offset.

Declaration
Following is the declaration for fseek() function.

int fseek(FILE *stream, long int offset, int whence)
Parameters
stream − This is the pointer to a FILE object that identifies the stream.

offset − This is the number of bytes to offset from whence.

whence − This is the position from where offset is added. It is specified by one of the following constants −

Sr.No. Constant & Description
1
SEEK_SET

Beginning of file

2
SEEK_CUR

Current position of the file pointer

3
SEEK_END

End of file

Return Value
This function returns zero if successful, or else it returns a non-zero value.

Example
The following example shows the usage of fseek() function.

```
#include <stdio.h>

int main () {
   FILE *fp;

   fp = fopen("file.txt","w+");
   fputs("This is tutorialspoint.com", fp);

   fseek( fp, 7, SEEK_SET );
   fputs(" C Programming Language", fp);
   fclose(fp);

   return(0);
}
```

Let us compile and run the above program that will create a file file.txt with the following content. Initially program creates the file and writes This is tutorialspoint.com but later we had reset the write pointer at 7th position from the beginning and used puts() statement which over-write the file with the following content −

This is C Programming Language
Now let's see the content of the above file using the following program −

```
#include <stdio.h>

int main () {
   FILE *fp;
   int c;
```

```
   fp = fopen("file.txt","r");
   while(1) {
      c = fgetc(fp);
      if( feof(fp) ) {
         break;
      }
      printf("%c", c);
   }
   fclose(fp);
   return(0);
}
```

Let us compile and run the above program to produce the following result −

This is C Programming Language

(b)

The C library function FILE *fopen(const char *filename, const char *mode) opens the filename pointed to, by filename using the given mode.

Declaration
Following is the declaration for fopen() function.

FILE *fopen(const char *filename, const char *mode)
Parameters
filename − This is the C string containing the name of the file to be opened.

mode − This is the C string containing a file access mode. It includes −

Sr.No.	Mode & Description
1	"r" Opens a file for reading. The file must exist.
2	"w" Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
3	"a" Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.

4
"r+"

Opens a file to update both reading and writing. The file must exist.

5
"w+"

Creates an empty file for both reading and writing.

6
"a+"

Opens a file for reading and appending.

Return Value
This function returns a FILE pointer. Otherwise, NULL is returned and the global variable errno is set to indicate the error.

Example
The following example shows the usage of fopen() function.

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

int main () {
   FILE * fp;

   fp = fopen ("file.txt", "w+");
   fprintf(fp, "%s %s %s %d", "We", "are", "in", 2012);

   fclose(fp);

   return(0);
}
```

Let us compile and run the above program that will create a file file.txt with the following content −

We are in 2012
Now let us see the content of the above file using the following program −

```
#include <stdio.h>

int main () {
   FILE *fp;
   int c;
```

```
  fp = fopen("file.txt","r");
  while(1) {
    c = fgetc(fp);
    if( feof(fp) ) {
      break ;
    }
    printf("%c", c);
  }
  fclose(fp);

  return(0);
}
```

Let us compile and run the above program to produce the following result −

We are in 2012

(c)

getch() is a nonstandard function and is present in conio.h header file which is mostly used by MS-DOS compilers like Turbo C. It is not part of the C standard library or ISO C, nor is it defined by POSIX.

Like these functions, getch() also reads a single character from the keyboard. But it does not use any buffer, so the entered character is immediately returned without waiting for the enter key.

Syntax:

int getch(void);
Parameters: This method does not accept any parameters.

Return value: This method returns the ASCII value of the key pressed.

Example:

filter_none
edit
play_arrow

brightness_4

```
// Example for getch() in C

#include <stdio.h>
```

```
// Library where getch() is stored
#include <conio.h>

int main()
{
   printf("%c", getch());
   return 0;
}
```

Input: g (Without enter key)
Output: Program terminates immediately.
But when you use DOS shell in Turbo C,
it shows a single g, i.e., 'g'

Important Points regarding getch() method:

getch() method pauses the Output Console untill a key is pressed.
It does not use any buffer to store the input character.
The entered character is immediately returned without waiting for the enter key.
The entered character does not show up on the console.
The getch() method can be used to accept hidden inputs like password, ATM pin numbers, etc.
Example: To accept hidden passwords using getch()

Note: Below code won't run on Online compilers, but on MS-DOS compilers like Turbo IDE.

filter_none
brightness_4

```
// C code to illustrate working of
// getch() to accept hidden inputs

#include <conio.h>
#include <dos.h> // delay()
#include <stdio.h>
#include <string.h>

void main()
{

   // Taking the password of 8 characters
   char pwd[9];
   int i;

   // To clear the screen
   clrscr();

   printf("Enter Password: ");
   for (i = 0; i < 8; i++) {
```

```
        // Get the hidden input
        // using getch() method
        pwd[i] = getch();

        // Print * to show that
        // a character is entered
        printf("*");
    }
    pwd[i] = '\0';
    printf("\n");

    // Now the hidden input is stored in pwd[]
    // So any operation can be done on it

    // Here we are just printing
    printf("Entered password: ");
    for (i = 0; pwd[i] != '\0'; i++)
        printf("%c", pwd[i]);

    // Now the console will wait
    // for a key to be pressed
    getch();
}
```

Output:
Abcd1234
Output:

Enter Password: ********
Entered password: Abcd1234

(d)

The C library function int getchar(void) gets a character (an unsigned char) from stdin. This is equivalent to getc with stdin as its argument.

Declaration
Following is the declaration for getchar() function.

int getchar(void)
Parameters
NA

Return Value
This function returns the character read as an unsigned char cast to an int or EOF on end of file or error.

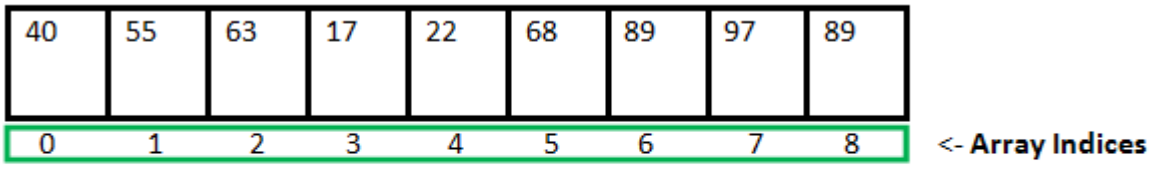
Example
The following example shows the usage of getchar() function.

```
#include <stdio.h>

int main () {
   char c;

   printf("Enter character: ");
   c = getchar();

   printf("Character entered: ");
   putchar(c);

   return(0);
}
```

Let us compile and run the above program that will produce the following result −

```
Enter character: a
Character entered: a
```

5 Array, structure and union are the container data type.
Array and structure both are the container data type. The major difference between an array and structure is that an “array” contains all the elements of “same data type” and the size of an array is defined during its declaration, which is written in number within square brackets, preceded by the array name.
The structure and union both are the container data types that can hold data of any “type”. The one major difference that distinguishes structure and union is that the structure has a separate memory location for each of its members whereas, the members of a union share the same memory location. Let’s understand the difference between structure and union, along with a comparison chart.



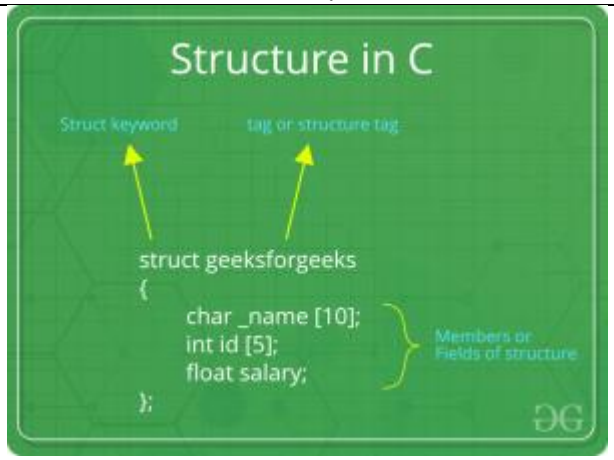
Array Length = 9
First Index = 0
Last Index = 8

Global Institute of Technology, Jaipur

ITS-1, IT Park, EPIP, Sitapura Jaipur 302022 (Rajasthan)

Solution III sem University Examination 2019

IFY1-06 I Sem / I Year



Key Differences Between Array and Structure

Where an Array is a collection of variables of the similar data type. On the other hand, Structure is a collection of variables of dissimilar data types.

Variable of an array are stored in a contiguous memory location whereas, the variables in a structure may or may not be stored in a contiguous memory location.

If you want to access any variable in an array you have to access it using its index number which shows its position in that array. If you want to access a variable in a structure then you have to access it using structure name followed by a dot followed by a variable name.

An operator used in Array is square bracket "[ ]", it is used while Array declaration and also while accessing an array variable. An operator used in the structure to access the structure variable is a dot operator.

An array name is a pointer, as the name of array points to the address of a first variable in that array. On the other hand, structure name does not point to its first element so a structure name is not a pointer.

We can not instantiate an array whereas, we can instantiate a structure.

All elements in an array have the same size because all elements are of the same datatype whereas, the structure contains elements of dissimilar datatype hence, all elements are of different size.

Bit-field can not be defined in an array whereas, the structure allows defining bit field in it.

Declaring array does not require any keyword. Declaring a structure requires a keyword struct.

An array is not a user defined data type whereas structure is a user-defined datatype.

An array can be accessed faster as compared to a structure.

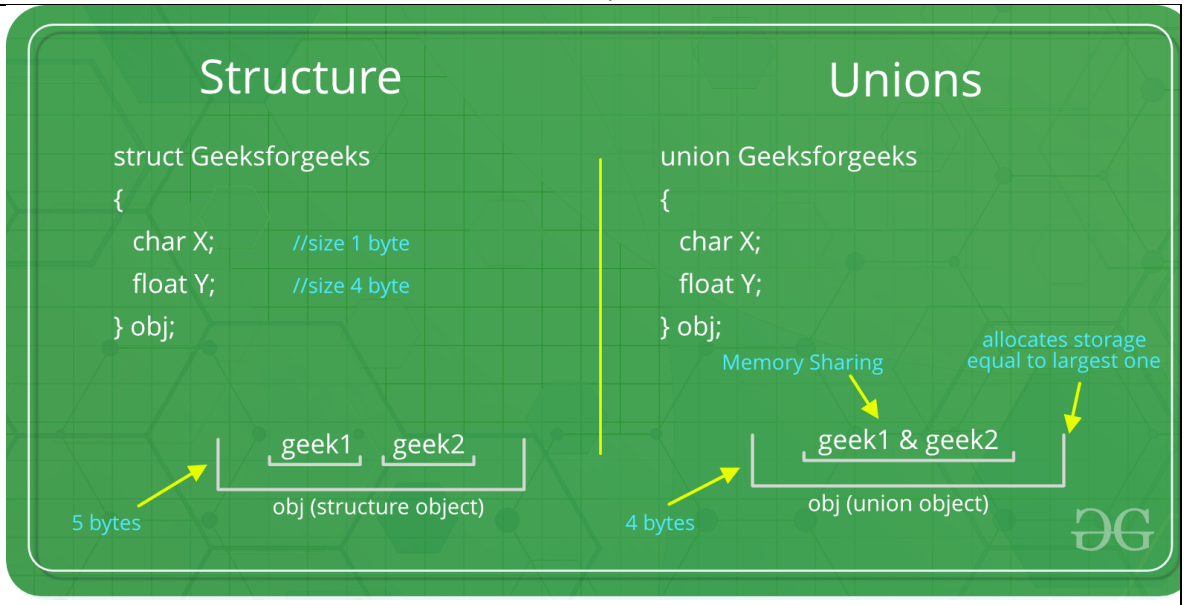
Searching an element in an array is faster as compared to a structure.

Global Institute of Technology, Jaipur

ITS-1, IT Park, EPIP, Sitapura Jaipur 302022 (Rajasthan)

Solution III sem University Examination 2019

IFY1-06 I Sem / I Year



	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Part B

1 **(a)**

```
#include <stdio.h>
int main() {
   int i, j, rows;
   printf("Enter the number of rows: ");
   scanf("%d", &rows);
   for (i = rows; i >= 1; --i) {
      for (j = 1; j <= i; ++j) {
```

```
        printf("* ");
      }
      printf("\n");
   }
   return 0;
}
```

(b)

```
#include <stdio.h>
int power(int, int);

int main()
{
  int n, sum = 0, t, remainder, digits = 0;

  printf("Input an integer\n");
  scanf("%d", &n);

  t = n;
  // Count number of digits
  while (t != 0) {
    digits++;
    t = t/10;
  }

  t = n;

  while (t != 0) {
    remainder = t%10;
    sum = sum + power(remainder, digits);
    t = t/10;
  }

  if (n == sum)
    printf("%d is an Armstrong number.\n", n);
  else
    printf("%d isn't an Armstrong number.\n", n);

  return 0;
}

int power(int n, int r) {
  int c, p = 1;

  for (c = 1; c <= r; c++)
    p = p*n;
```

	return p; }
2	**(a)** Syntax Error at first line (Preprocessor directive). Its #include<stdio.h> **(b)** C Identifiers C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore. If the identifier is not used in the external linkage, then it is called as an internal identifier. If the identifier is used in the external linkage, then it is called as an external identifier. We can say that an identifier is a collection of alphanumeric characters that begins either with an alphabetical character or an underscore, which are used to represent various programming elements such as variables, functions, arrays, structures, unions, labels, etc. There are 52 alphabetical characters (uppercase and lowercase), underscore character, and ten numerical digits (0-9) that represent the identifiers. There is a total of 63 alphanumerical characters that represent the identifiers. Rules for constructing C identifiers The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore. It should not begin with any numerical digit. In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive. Commas or blank spaces cannot be specified within an identifier. Keywords cannot be represented as an identifier. The length of the identifiers should not be more than 31 characters. Identifiers should be written in such a way that it is meaningful, short, and easy to read. Data types in C Language Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we can use in our program. These datatypes have different storage capacities. C language supports 2 different type of data types: Primary data types: These are fundamental data types in C namely integer(int), floating point(float), character(char) and void. Derived data types:

Derived data types are nothing but primary datatypes but a little twisted or grouped together like array, stucture, union and pointer. These are discussed in details later.

Data type determines the type of data a variable will hold. If a variable x is declared as int. it means x can hold only integer values. Every variable which is used in the program must be declared as what data-type it is.

Integer type
Integers are used to store whole numbers.

Size and range of Integer type on 16-bit machine:

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Floating point type
Floating types are used to store real numbers.

Size and range of Integer type on 16-bit machine

Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Character type
Character types are used to store characters value.

Size and range of Integer type on 16-bit machine

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

void type
void type means no value. This is usually used to specify the type of functions which returns nothing. We will get acquainted to this datatype as we start learning more advanced topics in C language, like functions, pointers etc.

4 **(a)**

```
/**
 * C program to swap two numbers using call by value
```

```
 */

#include <stdio.h>

/* Swap function definition */
void swap(int num1, int num2)
{
   int temp;

   printf("In Function values before swapping: %d %d\n", num1, num2);

   temp = num1;
   num1 = num2;
   num2 = temp;

   printf("In Function values after swapping: %d %d\n\n", num1, num2);
}

/* main() function definition */
int main()
{
   int n1, n2;

   /* Input two integers from user */
   printf("Enter two numbers: ");
   scanf("%d%d", &n1, &n2);

   /* Print value of n1 and n2 in before swapping */
   printf("In Main values before swapping: %d %d\n\n", n1, n2);

   /* Function call to swap n1 and n2 */
   swap(n1, n2);
   printf("In Main values after swapping: %d %d", n1, n2);

   return 0;
}
```

(b)

```
/**
 * C program to swap two numbers using call by reference
 */

#include <stdio.h>

/**
 * *num1 - pointer variable to accept memory address
 * *num2 - pointer variable to accept memory address
```

```
 */
void swap(int * num1, int * num2)
{
   int temp;

   printf("In Function values before swapping: %d %d\n", *num1, *num2);

   temp  = *num1;
   *num1 = *num2;
   *num2 = temp;

   printf("In Function values after swapping: %d %d\n\n", *num1, *num2);
}

/* main() function declaration */
int main()
{
   int n1, n2;

   printf("Enter two numbers: ");
   scanf("%d%d", &n1, &n2);

   printf("In Main values before swapping: %d %d\n\n", n1, n2);

   /*
    * &n1 - & evaluate memory address of n1
    * &n2 - & evaluate memory address of n2
    */
   swap(&n1, &n2);

   printf("In Main values after swapping: %d %d", n1, n2);

   return 0;
}
```

5 **(a)**

C If else statement

Syntax of if else statement:

If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” are skipped.

If condition returns false then the statements inside the body of “if” are skipped and the statements in “else” are executed.

```
if(condition) {
  // Statements inside body of if
}
else {
```

```
  //Statements inside body of else
}
```

Example of if else statement

In this program user is asked to enter the age and based on the input, the if..else statement checks whether the entered age is greater than or equal to 18. If this condition meet then display message "You are eligible for voting", however if the condition doesn't meet then display a different message "You are not eligible for voting".

```
#include <stdio.h>
int main()
{
  int age;
  printf("Enter your age:");
  scanf("%d",&age);
  if(age >=18)
  {
        /* This statement will only execute if the
         * above condition (age>=18) returns true
         */
        printf("You are eligible for voting");
  }
  else
  {
        /* This statement will only execute if the
         * condition specified in the "if" returns false.
         */
        printf("You are not eligible for voting");
  }
  return 0;
}
```

(b)

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

Syntax

The syntax for a switch statement in C programming language is as follows −

```
switch(expression) {

  case constant-expression  :
    statement(s);
    break; /* optional */

  case constant-expression  :
```

```
    statement(s);
    break; /* optional */

  /* you can have any number of case statements */
  default : /* Optional */
  statement(s);
}
```

The following rules apply to a switch statement −

The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.

You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.

The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.

When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.

When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Example

```
#include <stdio.h>

int main () {

  /* local variable definition */
  char grade = 'B';

  switch(grade) {
    case 'A' :
      printf("Excellent!\n" );
      break;
    case 'B' :
    case 'C' :
```

```
        printf("Well done\n" );
        break;
      case 'D' :
        printf("You passed\n" );
        break;
      case 'F' :
        printf("Better try again\n" );
        break;
      default :
        printf("Invalid grade\n" );
    }

    printf("Your grade is  %c\n", grade );

    return 0;
}
```

(c)

A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a while loop in C programming language is −

```
while(condition) {
   statement(s);
}
```

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
```

}

(d)

Unlike for and while loops, which test the loop condition at the top of the loop, the do...while loop in C programming checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

The syntax of a do...while loop in C programming language is −

```
do {
   statement(s);
} while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Example

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {
      printf("value of a: %d\n", a);
      a = a + 1;
   }while( a < 20 );

   return 0;
}
```

Part C

1

```
#include <stdio.h>

void main()
{
        float x,sum,no_row;
        int i,n;
```

```
        printf("Input the value of x :");
        scanf("%f",&x);
        printf("Input number of terms : ");
        scanf("%d",&n);
        sum =1; no_row = 1;
        for (i=1;i<n;i++)
        {
          no_row = no_row*x/(float)i;
          sum =sum+ no_row;
        }
        printf("\nThe sum  is : %f\n",sum);
}
```

2 **(a)**

```
#include<stdio.h>
struct time_struct
{
        int Hour;
        int Minute;
        int Second;
};
void main()
{
        struct time_struct T;
        printf("\nEnter the Hour (less than 24 hrs):");
        scanf("%d",&T.Hour);
        if(T.Hour>24)
        {
                printf("\nInvalid Hour !");
        }
        printf("\nEnter the Minute (less than 60 minutes):");
        scanf("%d",&T.Minute);
        if(T.Minute>60)
        {
                printf("\nInvalid Minute !");
        }
        printf("\nEnter the Second (less than 60 seconds):");
        scanf("%d",&T.Second);
        if(T.Second>60)
        {
                printf("\nInvalid Second !");
        }
        if(T.Hour<24 && T.Minute<60 && T.Second<60 )
        {
                printf("\n%d:%d:%d\n",T.Hour,T.Minute,T.Second);
        }
        else
        {
```

```
                printf("\nInvalid Input. \nThanks for using it.");
        }
        getch();
}
```

(b)

```
#include <stdio.h>
void cyclicSwap(int *a, int *b, int *c);
int main() {
   int a, b, c;

   printf("Enter a, b and c respectively: ");
   scanf("%d %d %d", &a, &b, &c);

   printf("Value before swapping:\n");
   printf("a = %d \nb = %d \nc = %d\n", a, b, c);

   cyclicSwap(&a, &b, &c);

   printf("Value after swapping:\n");
   printf("a = %d \nb = %d \nc = %d", a, b, c);

   return 0;
}

void cyclicSwap(int *n1, int *n2, int *n3) {
   int temp;
   // swapping in cyclic order
   temp = *n2;
   *n2 = *n1;
   *n1 = *n3;
   *n3 = temp;
}
```

3 File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

Creation of the new file
Opening an existing file
Reading from the file
Writing to the file
Deleting the file

Functions for file handling

There are many functions in the C library to open, read, write, search and close the file. A list of some file functions are given below:

Function
fopen()
fprintf()
fscanf()
fputc()
fgetc()
fclose()
fseek()
fputw()
fgetw()
ftell()
rewind()

Opening File: fopen()
We must open a file before it can be read, write, or update. The fopen() function is used to open a file. The syntax of the fopen() is given below.

FILE *fopen(const char * filename, const char * mode);
The fopen() function accepts two parameters:

The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like "c://some_folder/some_file.ext".
The mode in which the file is to be opened. It is a string.
We can use one of the following modes in the fopen() function.

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

The fopen function works in the following way.

Firstly, It searches the file to be opened.
Then, it loads the file from the disk and place it into the buffer. The buffer is used to provide efficiency for the read operations.

It sets up a character pointer which points to the first character of the file.

Closing File: fclose()
The fclose() function is used to close a file. The file must be closed after performing all the operations on it. The syntax of fclose() function is given below:

```
int fclose( FILE *fp );
```

Program

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

main()
{
  char ch, source_file[20], target_file[20];
  FILE *source, *target;

  printf("Enter name of file to copy\n");
  gets(source_file);

  source = fopen(source_file, "r");

  if( source == NULL )
  {
    printf("Press any key to exit...\n");
    exit(EXIT_FAILURE);
  }

  printf("Enter name of target file\n");
  gets(target_file);

  target = fopen(target_file, "w");

  if( target == NULL )
  {
    fclose(source);
    printf("Press any key to exit...\n");
    exit(EXIT_FAILURE);
  }

  while( ( ch = fgetc(source) ) != EOF )
    fputc(ch, target);

  printf("File copied successfully.\n");
```

```
//   fclose(source);
   fclose(target);

{
{
   FILE *fp;

   fp = fopen(target_file, "r"); // read mode

   if (fp == NULL)
   {
      perror("Error while opening the file.\n");
      exit(EXIT_FAILURE);
   }

   printf("The contents of %s file are:\n", target_file);

   while((ch = fgetc(fp)) != EOF)
      printf("%c", ch);

   fclose(fp);
   return 0;
}
}
   return 0;
}
```