Set A

Scoring Indicators

Course Name : Programming in C
Course Code : (2021)-3132 **QID : 2110220121**

Qn. No.	Scoring Indicator	Split score	Sub Total	Total score
I	**PART - A**			9

1	If (expression-1)
	Statement-1;
	else if (expression-2)
	Statement-2;
	else if (expression-n)
	Statement-n;
	else
	Default-statement;
		1	1	

2	Recursion is the mechanism by which a function calls itself	1	1	
3	Zero (0)	1	1	
4	Two dimensional	1	1	
5	strlen()	1	1	
6	A pointer is a variable that store memory address or that contains address of another variable	1	1	
7	float age[5], *ptr; ptr=age;	1	1	

8	union <union name>
	{
	datatype member1;
	datatype member2;
	};
		1	1	

9	Command-line arguments are parameters that are given on the system's command line, and the values of these arguments are passed to main() function during program execution.	1	1	
II	**PART - B**			24

1	While(condition)
	{
	Body of the loop
	}
	do
	{
	Body of the loop
	} while(condition);
		With explanation 1.5 + 1.5	3	

2	• It is a group of variables of similar data types referred to by a single name. • Its elements are stored in a contiguous memory location. • The size of the array should be mentioned while declaring it. • Array elements are always counted from zero (0) onward. • Array elements can be accessed using the position of the element (index) in the array. • Memory allocation is static	Any 4 x 0.75	3	

3	int main() {
	int a[10], Size, i, Smallest;
	printf("\nPlease Enter the size of an array \n");
	scanf("%d",&Size);
	printf("Enter elements of an array: \n");
	for(i=0; i<Size; i++) {
	scanf("%d",&a[i]); }
	Smallest = a[0];
		Array reading(1)+ Logic(2)
		
		** Only logic is important
			3	

	for(i=1; i<Size; i++) {
	if(Smallest > a[i]) {
	Smallest = a[i];
	} }
	printf("\nSmallest element in an Array = %d", Smallest);
	return 0; }			

4	**strcpy()** - This function is used to copying one string to another string. The syntax is: **strcpy(str1,str2);** copies str2 to str1 including the NULL character. Here str2 is the source string and str1 is the destination string. The old content of the destination string str1 are lost. **strcat()** – This function is used to append(combine/concatenate) a copy of a string at the end of the another string. The NULL character from str1 is moved and str2 is added at the end of str1. The 2nd string str2 remains unaffected. The syntax is: **strcat(str1,str2);**	Description (1+1) + syntax (2x0.5)	3	

5	The array_name contains the address of the first element. Need to pass only the name of the array in the function which is intended to accept an array. The array defined as the formal parameter will automatically refer to the array specified by the actual parameter.
	
	main() {
	int ar[10];
	……..
	sort(ar); // only name of array, no need of size and subscript operator
	…….. }
	
	sort(int xy[]){ // The subscript operator [] used with array name
	……..
	}
		Description(1) + example (2)	3	

6	int n1=25, n2=17, large, *p1,*p2;
	p1=&n1;
	p2=&n2;
	if(*p1>*p2)
	large=*p1;
	else
	large=*p2;
	printf("Largest is %d", large);
		Declaration and address assignment (1) + logic (2)
		
		** pointer can also be used for variable **large**
			3	

7	1. When pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value. 2. Any change made by the function using the pointer is permanently made at the address of passed variable. 3. No need to return values, any change in formal parameter automatically affects the value of actual parameter. 4. More than one value can be returned at the same time.	Any 3 x 1	3	
8	Declare using **char** type pointer. Initialize by assigning string value in double quotes to the pointer variable char *str; str="Computer Engineering"	Explanation (1) + example (2)	3	
9	Array of pointers is a collection of addresses (pointers) of same type. int *ptr[10]; It declares **ptr** as an array of 10 integer pointers. Thus, each element in **ptr**, holds a pointer to an **int** value.	Explanation (2) + example (1)	3	
10	These are used by the programmers to create their own data types and define what values the variables of these data types can hold. The keyword used to declare enumerated type is **enum**. **Syntax** enum tagname{ identifier1, identifier2,……,identifier n };	Statement (1) + syntax (1) + example (1)	3	

	Example enum week{ mon, tue, wed, thu, fri, sat, sun };			
III	**PART - C**			42
1	#include - include the contents of a file during compilation, to include header files. #include "filename" **or** #include <filename> Eg:- #include<stdio.h> #define – to declare symbolic constants, to replace a token by an arbitrary sequence of characters, conditional compilation and macros with arguments. #define C_NAME value **or** #define C_NAME expression Eg:- #define pi 3.14	Listing (1) + Use (2) + Syntax(2) + examples(2)	7	
	OR			
2	**Automatic, Register, Static, External** Automatic Scope:-local to the block or function in which variable is defined. Life time:-Till the control remains within function or block in which it is defined. It terminates when function is released Register Scope :-local to the function or block in which it is defined. Life time :-till controls remains within function or blocks in which it is defined. Static Scope :- local to the block or function in which it is defined. Life time:- value of the variable persist or remain between different function call (as long as program execution remains it retains) External Scope :- global Life time:-as long as program execution remains it retains	Listing (1) + any 3 x 2	7	

3	void Even(int,int);
	int main() {
	int n;
	printf("Enter the limit: ");
	scanf("%d", &n);
	Even(2,n);
	return 0; }
	
	void Even(int cur, int n) {
	if(cur <=n) {
	printf("%d, ", cur);
	Even(cur + 2, n);}
	}
		Function declaration (1) + function calling in main() function (1) + function definition and logic (4) + recursive call (1)	7	

	OR			

4	int reverse(int);
	int main() {
	int n,rev;
	printf("Enter the number: ");
	scanf("%d", &n);
	rev=reverse(n);
	printf("The reverse number is: %d",rev);
	return 0; }
	
	int reverse(int n) {
	int rem, rno=0;
		Function declaration (1) + function calling in main() function (1) + function definition (2) + Logic(3)	7	

	while (n!=0) {
	rem=n%10;
	rno=rno*10+rem;
	n=n/10;
	}
	return (rno); }			

5	Step 1 - Read the search element from the user. Step 2 - Find the middle element in the sorted list. Step 3 - Compare the search element with the middle element in the sorted list. Step 4 - If both are matched, then display "Given element is found" and terminate the function. Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element. Step 6 - If the search element is smaller than middle element, repeat steps 2, 3, 4 and 5 for the left sublist of the middle element. Step 7 - If the search element is larger than middle element, repeat steps 2, 3, 4 and 5 for the right sublist of the middle element. Step 8 - Repeat the same process until we find the search element in the list or until sublist contains only one element. Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list" and terminate the function.	Algorithm/ logic/ steps/ program (7)	7	
	OR			

6	void main(){
	int size,i,j,temp,list[100];
	printf("Enter the size of the List: ");
	scanf("%d",&size);
	printf("Enter %d integer values: ",size);
	for(i=0; i<size; i++)
	scanf("%d",&list[i]);
	for(i=0; i<size; i++){
	for(j=i+1; j<size; j++){
	if(list[i] > list[j])
	{
	temp=list[i];
	list[i]=list[j];
	list[j]=temp;
	} } }
	printf("List after sorting is: ");
	for(i=0; i<size; i++)
	printf(" %d",list[i]);
	}
		Variable, Array declaration (1) + reading values (1) + sorting logic (4) + print (1)
		
		** another logic exchanges values after one complete inner loop iteration
			7	

7	#include<stdio.h>
	int main () {
	int number[10], *p;
	int i,sum=0;
	p = number;
	printf("Enter array element: ");
	for(i=0;i<10;i++)
	{
	scanf("%d", &(*(p+i)));
	}
	for(i=0;i<10;i++)
	{
	if((*(p+i))%2 !=0)
	sum=sum + (*(p+i));
	}
	printf("The sum of odd numbers is %d", sum);
	}
		Array and pointer declaration (1) + assigning array to pointer (1) + reading values (2) + sum calculation (2) + printing (1)	7	

	OR			

8	The process of allocating memory at the time of execution or at the runtime, is called dynamic memory location. Allocation and release of memory space can be done with the help of some library function called dynamic memory allocation function.
	malloc() - This function use to allocate memory during run time, its declaration is
	void *malloc(size);
	malloc () - returns the pointer to the 1st byte and allocate memory, and its return type is void, which can be type cast such as:
	int *p=(datatype*)malloc(size);
	
	calloc() - Similar to malloc() only difference is that calloc() function use to allocate multiple block of memory. Two arguments are there, 1st argument specify number of blocks, 2nd argument specify size of each block.
	syntax:-
	int *p= (int*) calloc(number of blocks, size of each block);
	
	realloc() - The function realloc() use to change the size of the memory block and it alter the size of the memory block without loosing the old data, it is called reallocation of memory.
	It takes two argument such as;
	int*p=(int *)realloc(ptr, new size);
	
	free() - Function free() is used to release space allocated dynamically. The memory released by free() is made available again. It can be used for further purpose.
	Syntax :
	free(p); , where p is pointer.
		Statement (1) + any 3x2 (6)	7	

9	struct student {
	int register_number;
	char name[20];
	float cgpa;
	};
	
	main() {
	struct student st[50];
	int n,i;
	scanf("%d",&n);
	for(i=0;i<n;i++) {
	scanf("%d%s%f",&st[i].register_number,st[i].name,&st[i].cgpa);
	}
	for(i=0;i<n;i++) {
	printf("%d%s%f",st[i].register_number,st[i].name,st[i].cgpa);
	}
	}
		Structure definition (1) + structure array declaration (1) + reading (2) + printing (2) + logic (1)	7	

	OR			

10	****step 1 – Structure definition*********
	struct complex {
	int real;
	int imaginary;
	};
	
	****step 2 function declaration*********
	void fun(struct complex);
	
	****step 3 function definition with structure type argument***
	void fun(struct complex c){
		Structure definition (1) + function declaration (1) +function definition (2) + function calling (1) + example (2)	7	

	……
	}
	
	******step 4 main() function definition*****
	main(){
	struct complex c1;
	……
	Reading of structure
	******step 5 function calling by passing structure as argument*****
	fun(c1);
	}
		**** any example can be used		

11	struct complex {
	int real;
	int imaginary;
	};
	
	main() {
	struct complex c1,c2,c3;
	printf("Enter the real and imaginary part of first number");
	scanf("%d%d",&c1.real, &c1.imaginary);
	printf("Enter the real and imaginary part of second number");
	scanf("%d%d",&c2.real, &c2.imaginary);
	c3.real=c1.real+c2.real;
	c3.imaginary=c1.imaginary+c2.imaginary;
	printf("The resultant number is = %d +i %d",c3.real,c3.imaginary);
	}
		Structure definition (1) + structure variable declaration (1) + reading (2) + addition (2)+ printing (1)	7	

	OR			

12	**Opening a file:**
	Before performing any type of operation, a file must be opened and for this fopen() function is used.
	syntax:
	file-pointer=fopen("FILE NAME " ," Mode of open");
	example:
	FILE *fp=fopen("ar.c" ," r");
	If fopen() unable to open a file than it will return NULL to the file pointer
	Reading a character from a file
	fgetc() is used for reading a character from the file
	Syntax:
	character variable= fgetc(file pointer);
	Example:
	ch=fgetc(fp);
	Writing a character into a file
	fputc() is used to writing a character to a file
	Syntax:
	fputc(character,file_pointer);
	Example:
	fputc(ch,fp);
		3+2+2	7	