2007 MAHARASHTRA STATE ICFAI UNIVERSITY M.C.A

Programming And Problem Solving In C (MC151) :

April 2006

Section A : Basic Concepts (30 Marks)

· This section consists of questions with serial number 1 - 30.

· Answer all questions.

· Each question carries one mark.

· Maximum time for answering Section A is 30 Minutes.

1.
What number will z in the sample code given below?

int z, x=5, y= -10, a=4, b=2;

z=x++ - --y *b/a;

(a) 5 (b) 6 (c) 10 (d) 11 (e) 12.
< Answer >

2.
Every C program consists of __________ function(s).

(a) Only one (b) Only two (c) One or two

(d) One or many (e) None of the above.
< Answer >

3.
What is the correct value to return to the operating system upon the successful completion of a program?

(a) 1 (b) 1 (c) 0

(d) Programs do not return a value (e) 2.
< Answer >

4.
What is the only function all C programs must contain?

(a) Start() (b) System() (c) Main()

(d) Program() (e) None of the above.
< Answer >

5.
Which of the following is not a correct variable type?

(a) Float (b) Real (c) Int (d) Double (e) Char.
< Answer >

6.
Which of the following correctly accesses the seventh element stored in foo, an array with 100 elements?

(a) Foo[6]; (b) Foo[7]; (c) Foo(7); (d) Foo; (e) Foo{ };
< Answer >

7.
Which of the following is a complete function?

(a) Int funct(); (b) Int funct(int x) {return x=x+1;}

(c) Void funct(int) {printf(“Hello");} (d) Void funct(x) {printf(“Hello");}

(e) None of the above.
< Answer >

8.
Which of the following is a not a keyword in C language?

(a) Void (b) Volatile (c) Sizeof (d) Getchar (e) Short.
< Answer >

9.
What is the return value of the following statement if it is placed in C program?

strcmp (“ABC”,”ABC”);

(a) 33 (b) –1 (c) 1 (d) Compilation error (e) 0.
< Answer >

10.

Suppose the return type of function f is void. Which one of the following statements is true?

(a) Function f does not contain a return statement

(b) Function f does not have parameters

(c) Function f returns a generic value

(d) Function f does not work correctly

(e) None of the above.
< Answer >

11.
Which command is used to skip the rest of a loop and carry on from the top of the loop again?

(a) Break; (b) Resume; (c) Continue;

(d) Skip (e) None of the above.
< Answer >

12.
How would you declare a constant of 5 called "MYCONST"?

(a) constant MYCONST = 5; (b) int myconst = 5;

(c) #define MYCONST 5 (d) constant = 5

(e) #define MYCONST=5.
< Answer >

13.
Which of the following is not a storage class in C?

(a) Auto (b) Struct (c) Extern (d) Static (e) Register.
< Answer >

14.
What is the fifth element of the array int a[3][4]={1,2,3,4,5,6,7,8,9,10,11}; ?

(a) 4 (b) 6 (c) 5 (d) 7 (e) 3.
< Answer >

15.

What is the output of the following program?

```
#include

int c [10]={1,2,3,4,5,6,7,8,9,10};

main ()

{ int a,b=0;

for(a=0;a<10;++a)

if(c[a]%2= =1)

b+=c[a];

printf (‘%d”, b);}
```

(a) 20 (b) 24 (c) 25

(d) 30 (e) None of the above.
< Answer >

16.
int a [5]={1,2,3} what is the value of a[4]?

(a) 3 (b) 1 (c) Garbage value (d) 0 (e) 2.
< Answer >

17.
Information will be passed to the function via special identifier is called

(a) Arguments (b) Parameters (c) Both (a) and (b) above

(d) Elements (e) Characters.
< Answer >

18.
Which of the following function returns multiple values?

(a) Printf( ) (b) Scanf( ) (c) Printf( ) & scanf( )

(d) Getch( ) (e) None of the above.
< Answer >

19.
What is the associatively of the conditional operator?

(a) Left to Right (b) Right to Left (c) Top to Bottom

(d) Bottom to Top (e) None of the above.
< Answer >

20.
What is the output of the following statement?

Printf(“%v”, 12);

(a) 12 (b) 12.0 (c) %v

(d) Garbage value (e) None of the above.
< Answer >

21.
What is the output of the following statements?

int I = 0;

printf(“%d %d ”, I, I++);

(a) 01 (b) 10 (c) 00 (d) 11 (e) Junk value.
< Answer >

22.
What is the output of the following statements?

for(I = 10; I++; I<15)

printf(“%d”, I);

(a) 1011121314 (b) 101112131415

(c) 910111213 (d) It will go to infinite loop

(e) None of the above.
< Answer >

23.

The size of an int must be greater than or equal to that of a _________

(a) Long int (b) Short int (c) Float (d) Double (e) Char.
< Answer >

24.
Which of the following is a LOOP statement of a C language?

(a) Repeat-Until (b) For (c) While-Do

(d) Do-while (e) Both (b) and (d) above.
< Answer >

25.
What is the output of the following statements?

```
int b = 5, c = 15, d = 8, e = 8,a;

a = b>c?c>d?12:d>e?13:14:15;

printf("%d", a);
```

(a) 13 (b) 14 (c) 15 (d) Garbage value (e) 12.
< Answer >

26.
What is the output of the following piece of code?

```
for (i = 0; i<10; i++);

printf("%d",i);
```

(a) 10 (b) 0123456789 (c) Syntax error (d) 0 (e) Infinite loop.
< Answer >

27.
What value does srarray [5][4][0] in the sample code below contain?

```
int srarray[2][2][2] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

(a) 3 (b) 5 (c) 7

(d) 9 (e) Garbage value.
< Answer >

28.

What number would be shown on the screen after the following statements of C are executed?

char ch; int I; ch=’G’; I=ch-‘A’;

pirntf(“%d”, I);

(a) 6 (b) 7 (c) 8 (d) 5 (e) 9.
< Answer >

29.
Which of following is not a valid name for a C variable?

(a) Hairaj (b) Hello_raj (c) Hello raj

(d) Both (a) and (b) above (e) None of the above.
< Answer >

30.
What value would be stored in an integer variable “i” as a result of the following expression?

int i, j = 3; i = 4 + 2 * j/(j-1);

(a) 1 (b) 7 (c) 9 (d) 8 (e) 2.
< Answer >

END OF SECTION A

Section B : Problems (50 Marks)

· This section consists of questions with serial number 1 – 5.

· Answer all questions.

· Marks are indicated against each question.

· Detailed workings should form part of your answer.

· Do not spend more than 110 - 120 minutes on Section B.

1. a. Write a program which would print the alphabet set a to z and A to Z in decimal and character form.

b. Compare and contrast C functions and macros.

(5 + 5 = 10 marks) < Answer >

2. Write a program using for loop to print the following output.

I

IC

ICF

ICFA

ICFAI

(10 marks) < Answer >

3. The names of the employees of an organization are stored in three arrays, namely first_name, second_name, and last_name. Write a program to concatenate the three parts into one array to be called name

(10 marks) < Answer >

4. Taking a two-dimensional array (N x N) of integers as input to your program.. The program should display the matrix elements in the sorted order

Example:

Input:

Output:

(10 marks) < Answer >

5. A file named DATA contains a series of integer numbers. Code a program to read these numbers and then write all odd numbers to a file to be called ODD and all even numbers to a file to be called EVEN.

(10 marks) < Answer >

END OF SECTION B

Section C : Applied Theory (20 Marks)

· This section consists of question with serial number 6.

· Do not spend more than 25 -30 minutes on section C.

6. a. Write a short note on Union. What are the differences between Structure and Union?

b. What is recursion? Explain recursion technique with an example.

c. Explain about the storage classes in C language with suitable examples.

(5 + 5 + 10 = 20 marks) < Answer >

END OF SECTION C

END OF QUESTION PAPER

Suggested Answers

Programming and Problem solving in C (MC151) : April 2006

Section A : Basic Concepts

1.
Answer : (a)

Reason: According to precedence of operators.
< TOP >

2.
Answer : (d)

Reason:
< TOP >

3.
Answer : (c)

Reason: After successful completion of a program 0 is returned to the operating system.
< TOP >

4.
Answer : (c)

Reason: Since main() is the starting of a program execution.
< TOP >

5.
Answer : (b)

Reason: real is not a variable type in C language.
< TOP >

6.
Answer : (a)

Reason: Since array index starts with 0 in c language.
< TOP >

7.
Answer : (b)

Reason: Since return type for the function is int and in the function definition return statement is used by returning an integer value of x.
< TOP >

8.
Answer : (d)

Reason: Getchar() is not a keyword in C language.
< TOP >

9.
Answer : (e)

Reason: If the same statement is used in a C program compilation error will occur.
< TOP >

10.
Answer : (c)

Reason: Function returns a generic value.
< TOP >

11.
Answer : (c)

Reason: Continue statement is used to skip the rest of the statements in a loop and carry from the top of the loop again.
< TOP >

12.
Answer : (c)

Reason: #define statement is used.
< TOP >

13.
Answer : (b)

Reason: Struct is not a storage class in C language.
< TOP >

14.
Answer : (c)

Reason: In the two dimensional array representation there will be 3 rows and 4 columns. The elements are stored in row wise first. Hence the fifth element is 5.
< TOP >

15.

Answer : (c)

Reason: The remainder is 1 for only 1, 3, 5, 7, 9 numbers and the sum of these four numbers is 25.
< TOP >

16.
Answer : (c)

Reason: In the fourth location of an array contains garbage value since we are storing only three elements in the first three locations.
< TOP >

17.
Answer : (c)

Reason: We can call them as function arguments or parameters.
< TOP >

18.
Answer : (b)

Reason: Scanf() function returns multiple values.
< TOP >

19.
Answer : (b)

Reason: Right to left is the operator associativity for the conditional operator.
< TOP >

20.
Answer : (c)

Reason: The control string %v is printed.
< TOP >

21.
Answer : (b)

Reason: Since the evaluation is from right to left.
< TOP >

22.
Answer : (d)

Reason: Since the condition is always true it will go to infinite loop.
< TOP >

23.
Answer : (b)

Reason: The size of int is grater than or equal to short int.
< TOP >

24.
Answer : (e)

Reason:
< TOP >

25.
Answer : (c)

Reason:
< TOP >

26.
Answer : (a)

Reason: After 10 iterations for loop terminated and the result is 10.
< TOP >

27.
Answer : (e)

Reason: Since that many number of rows and columns are not actually represented the garbage value is printed.
< TOP >

28.
Answer : (a)

Reason: Since the ASCII value of G is 71 and the garbage value if A is 65 and hence the difference is 6.
< TOP >

29.
Answer : (c)

Reason: No spaces are allowed in the variable names.
< TOP >

30.
Answer : (b)

Reason: According to operator precedence rule the output is 7.
< TOP >

Section B : Problems

1. a.
```
#include

#include

void main()

{

char c;

printf(“\n\n”);

for( c=65;c<=122;c++)

{

if ((c> 90)&& (c<97)

continue;

printf(“%5d - %c”, c);

}

printf(“\n”);

}
```

b. Functions:

i. executed by interpreters and are compiled by the compilers

ii. Function calling mechanism works.

iii. Run time overhead if frequent calls are there for a function.

iv. Reduces the size of the source code.

v. Good if a function is having less parameter and is called for more number of times.

Macros:

i. Are processed by pre- compiler

ii. Are expanded inline

iii. Whenever are referenced, are replaced at the print of call

iv. Size of the object code is increased

v. Good if the function is called for less number of times or if the function is having large no of parameters.

< TOP >

```
2. #include

#include

#include

void main()

{

char a[20], *p;

int i, j;

*p=a;

clrscr();

printf(“ Enter any string”);

scanf(“%s”, a);
for(i=0;a[i]!=’\0’;i++)

{

printf(“\n”);

for(j=0;j<=i;j++)

printf(“%c”, *(p+j));

}

getch();

}
```

< TOP >

```
3. #include

main()

int i, j, k;

static char first_name[10]={“VENKATA”};

static char second _name[10]={“SAI”};

static char first_name[10]={“PRASANTH”};

char name[30];

for(i=0;first_name[i]!=’\0’;i++)

{

name[i]=first_name[i];

}

name[i]=’ ‘;

for(j=0;second_name[j]!=’\0’;j++)

{
```

```
name[i+j+1]=second_name[i];

}

name[i+j+1]=’ ‘;

for(k=0;last_name[k]!=’\0’;k++)

{

name[i+j+k+2]=last_name[k];

}

name[i]=’ ‘;

name[i+j+k+2]=’\0’;

printf(“\n\n”);

printf(“%s\n”, name);

}
```

< TOP >

```
4. #include

#include

void main()

{

int a[10][10], b[30];

int t,i,j,p,q,s;

clrscr();

printf(“ enter the order of the matrix\n”);

scnaf(“%d%d”, &p,&q);

t=0;
```

```
for(i=o;i

{

for(j=0;j

{

scanf("%d",&a[i][j]);

b[t]=a[i][j];

t++;

}

}

for(i=0;i

{

for(j=i+1;j

{

if(b[i]>b[j])

{

s=b[i];

b[i]=b[j];

b[j]=s;

}

}

}

printf("elements of matrix a :");

for(i=0;i
{
printf("\n");
for(j=0;j
printf("%d\t",a[i][j]);
}
/* assigning sorted elements*/
t=0;
for(i=0;i
{
for(j=0;j
{
a[i][j]=b[t];
t++;
}
}
printf("\n after Sorting elements of matrix are");
for(i=0;i
{
printf("\n");
for(j=0;j
printf("%d\t",a[i][j]);
}
getch();
```

}

< TOP >

5. #include

```
main()

{

file *f1, *f2, *f3;

int number, i ;

printf(“ Contents of Data file\n\n”);

f1=fopen(“DATA”, “w”);

for (i=1; i<=30; i++)

{

scanf(“%d”, &number);

if (number== -1)

break;

putw(number, f1);

}

fclose(f1);

f1=fopen(“DATA”, “r”);

f2=fopen(“ODD”, “w”);

f3= fopen(“EVEN”, “w”);

while(number=getw(f1)!=EOF)

{

if number(%2==0)
putw(number,f3);

else

putw(number,f2);

}

fclose(f1);

fclose(f2);

fclose(f3);

f2=fopen("ODD", "r");

f3=fopen("EVEN", "r");

printf("\n\ncontents ofODD file \n\n");

while(number=getw(f2))!=EOF)

printf("%4d", number);

printf("\n\nContents of EVEN file \n\n");

while ((number=getw(f3))!=EOF)

printf("%4d",number);

fclose(f2);

fclose(f3);

}
```

< TOP >

Section C: Applied Theory

6. a. Union is a derived datatype having the syntax and representation similar to structure but is served and used differently in ‘C’

Syntax : union tagname

{

Member DT Member 1;

Member DT Member2;

MemberDatatype Member n;

} unionvariable1, …….. unionvariable n;

The major difference between these two is with respect to the organization of members of each one. In structure a separate memory is (space) allocated to each and every individual member. And hence the size of any structure variable is the summation of the size of all the individual members present.

In union no any separate memory (space) is allocated to each of the member, but a total space equivalent to the size of the largest member of union variable is allocated and hence the size of the union variable is the size of the largest member of itself. As separate memory is allocated to structure members, all the members access their own memory individually and hence all the members can be accessed and used at a time without affecting the other members.

In union as all the members share the common memory location and hence all the members can't be accessed at a time, and only one member can be accessed at any moment of time.

Ex: struct student Memory Allocated to a

{

int id, marks [5] id (2 Bytes)

char name [20] name (20 bytes)

float Avg; Avg (4 bytes)

Marks (10 Bytes)

} a; and tota of 36 bytes

union student

```
{

int id, marks [5]

char name [20]

float avg;

} a;
```

In union one a new value is assigned to any member then it effects all the other members as all the other members simple loose their previous values and get assigned with the new value set. Nut there is no such effect in case of structures.

Generally union is used in working with system programming, or at a place where one would like to minimize the memory utilization and to make all the other members share some part of the common sharable memory. Hence in union the memory usage is very low.

b. Recursion:

Recursion is a process in which a function calls itself repeatedly until some specified condition has been satisfied. C implements recursion using recursive functions. A function is called recursive, if a statement within the body of the function calls the same function. Sometimes called circular definition, recursion is thus a process of defining some thing in terms of itself.

A standard example of recursion is the factorial function. This is a mathematical function which is important in statistics. The factorial function is defined to be the product (multiplication) of all integers from 1 to the parameter of the function. (The factorial of 0 is 1.)

Here are some examples of the factorial function. These are not executable C code examples, but pseudocode:

```
factorial(3) == 1 * 2 * 3 == 6

factorial(4) == 1 * 2 * 3 * 4 == 24

factorial(3) == 1 * 2 * 3 * 4 * 5 == 120
```

Formally, the factorial function is defined by two equations. (Again, these are in pseudocode).

factorial(n) = n * factorial(n-1)

factorial(0) = 1

The first of these statements is recursive, because it defines the value of factorial(n) in terms of factorial(n-1). The second statement allows the function to "bottom out".

Here is a short code example that incorporates a factorial function.

```
#include



int factorial (int n)

{

if (n == 0)

return 1;

else

return (n * factorial (n-1));

}

int main ()

{

printf ("%d\n", factorial(3));

return 0;

}
```

Let's follow the control flow in this program to see how controlled recursion can work. The main function prints the value of factorial(3). First, the factorial function is called with the parameter 3. The function tests whether its parameter n is zero. It is not, so it takes the else branch if the if statement, which instructs it to return the value of factorial(3-1). It therefore calls itself recursively with a parameter of 2.

The new call checks whether its parameter is zero. It isn't (it's 2), so it takes the else branch again, and tries to calculate 2 * factorial (1). In order to do so, it calls itself recursively with a value of 2-1, or 1. The new call checks whether its parameter is zero. It is actually 1, so it takes the else branch again and attempts to calculate 1 * factorial (0). In order to do so, it calls itself again with the parameter 0.

Again, the function checks whether its parameter is zero. This time it is, so the function bottoms out. It takes the first branch of the if statement and returns a value of 1. Now the previous function call can also return a value, and so on, until the very first call to factorial terminates, and the function returns a value of 6.

To sum up, the expression factorial(3) goes through the following steps before finally being evaluated:

factorial (3) == 3 * factorial(2)

== 3 * (2 * factorial(1))

== 3 * (2 * (1 * factorial(0)))

== 3 * (2 * (1 * 1)))

== 6

Remember a recursive function must satisfy the following two conditions:

First there must be some conditional statement in the function definition, on satisfying which the function must stop invoking itself. In the above example

if (n == 0)

return 1

is the stopping condition that needs to be satisfied for the function to stop invoking itself recursively.

Every time the function gets invoked recursively it must reach nearer to the stopping condition, on satisfying which the function stops invoking itself recursively.

c. Storage Classes

Storage classes are used to determine the scope and life time of a variable in C.

To fully define a variable one needs to mention not only it’s type but also its storage class. If we do not specify the storage class of a variable in its declaration, the compiler will assume a storage class depending upon the context in which the variable has been declared and used. Thus, variables have certain default storage class specifiers. The storage class of a variable gives the following information;

- Where the variable would be stored (memory or registers)

- What will be initial value of the variable if the variables are not initialized.

- Scope of the variable (In what and all functions the variable is accessible.

- Lifetime of the variable (how long variable exists in the memory)

Depending upon the above factors a variable can have one of the following storage classes associated with it.

Storage class	Keyword representing storage class
Automatic	auto
Static	static
External	extern
Register	register

Automatic storage class: The default storage class for variables declared inside a function or a block in C is automatic (auto).

The automatic variables are created and initialized when the function is invoked or block that declares the variable gets executed and gets destroyed once the function or the block finishes their execution.

Variables with automatic storage class specifier are not accessible outside the function or block in which they are declared. This may assure that we may declare and use the same variable name in different functions name in different functions in the same program without causing any confusion to the compiler.

Automatic variables cannot retain their values between the function calls.

An example program using automatic variables.

```
/* example program for static class specifier*/

#include
```

```
void autofun();

main()

{

int i;



clrscr();

for(i=1;i<=10;i++)

autofun();

}



void autofun()

{

int x=0; /* variable with automatic storage specifier */



printf("%d\t", x);

x++;

}
```

Output of the above function:

```
0 0 0 0 0 0 0 0 0 0
```

In the above program x value is always reinitialized to 0 whenever autofun() is called because automatic variables are not able to retain the values between the function calls. If you try to access the variable x beyond the function autofun(), it would generate an error because x is local to autofun() and is not accessible outside the function in which it has been declared.

Static storage class: As the name suggests variables with static storage specifier can retain their values even between the function calls. Static variables gets initialized only once, i.e.; at the time of declaration itself.

Like auto variables, static variables are also local to the block/function in which they have been declared with the only difference that unlike auto variables which gets destroyed when the function completes its execution, static variables can persists their values, even if the function finishes its execution, and if the control comes back to the same function again the static variables will have same value they had last time around. Static variables gets destroyed only when the program finishes its execution.

Example program for static variables

```
/* example program for static class specifier*/

#include

void staticfun();

main()

{

int i;



clrscr();

for(i=1;i<=10;i++)

staticfun();

}



void staticfun()

{

static int x; /* static variable */
printf("%d\t", x);

x++;

}
```

Output of the above program

```
0 1 2 3 4 5 6 7 8 9 10
```

If you observe the above program static variable x has been initialized to 0 by default and is been able to retain it's value even between the function calls. First time when staticfun() gets invoked the value of x is 0, after printing value of x, x value has been incremented by 1. Next time when the function invoked once again, the x value is 1, it prints 1 , and gets incremented by one more (ie. x becomes 2). This clearly shows that x is able to retain it's value even between function calls as its been declared as static.

External storage class: The external variables are accessible to all the functions within the program or even to the external programs. External variables are declared outside all the functions and are static by default (i.e., they get initialized to 0 by default and can retain their values even between function calls).

An Example program using external storage class

```
/* example program for external storage class specifier*/

#include

void increment();

void decrement();

int x; /* external variable or global variable */

main()

{

clrscr();

printf("Original value of x at declaration \t d\n", );

increment();
increment();

decrement();

decrement();

}

void increment()

{

x++;

printf("x value after incrementing %d \n",x);

}


void decrement()

{

x--;

printf("x value after decrementing %d \n",x);

}
```

Output of the above program

```
Original value of x at declaration 0

x value after incrementing 1

x value after incrementing 2

x value after decrementing 1

x value after decrementing 0
```

If you observe the output of the above program, the variable x, which has been declared globally outside all the functions, is accessible to all the functions within the program (main(), increment() and decrement()). External variables are static by default. That’s why x value has been automatically initialized to 0 and able to retain its value even between function calls.

Register storage specifier: Variables with register storage specifier are similar to automatic variables, with the only difference that they get stored in CPU registers. A value stored in the CPU register can be accessed faster than the one that is stored in memory. Therefore, if a variable is used in many places in a program then it is better to declare its storage class as register. Register storage specifier causes the compiler to request the CPU to the store the associated variable in CPU’s register. However it is optional to CPU whether to allocate the register to the variable or not, because CPU has limited no .of storages.

Declaring a variable to be register: register int x;

< TOP >

< TOP OF THE DOCUMENT >