CSE/IT Batch: Hinglish

Programming in C

Arrays and Pointers

DPP-04

[MCQ]

1. Consider the following codes:

P:
```
void *p;
p=malloc(1);
*p=65;
printf("%c",*(char*)p);
```

Q:
```
void *p;
char a='A';
p=malloc(1);
p=&a;
printf("%c",*(char*)p);
```

Which of the following is CORRECT?

(a) Both P and Q are valid.
(b) Only P is valid.
(c) Only Q is valid.
(d) Neither P nor Q is valid.

[MSQ]

2.
```
#include <stdio.h>
#include <stdlib.h>
int * f()
{
 int *p=(int*)malloc(sizeof(int));
 *p=10;
return p;
}
int * g(int a)
{
   return &a;
}
int main()
{
   printf("%p", f());//line 1
   printf("%p", g(15));//line 2
   return 0;
}
```

Which of the following statement(s) is/are INCORRECT?

(a) Line 1 will result into compilation error.
(b) Line 2 will result into compilation error.
(c) The outputs are garbage values.
(d) The hexadecimal addresses of pointer variables p and local variable are displayed.

[MCQ]

3.
```
#include <stdio.h>
int main()
{
   void *p, *q;
   int a=324;
   p=&a;
   printf("%d", *(char*)p);
   return 0;
}
```

The output is-

(a) Garbage value
(b) Compilation error
(c) 68
(d) 324

[NAT]

4.
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int *p=(int*)malloc(sizeof(int));
   int *q=(int*)malloc(sizeof(int));
   *p=376;
   *q=5;
while(*p>*q){
printf("%d\t",*p);
 *p/=*q;
 *q+=1;
}
return 0;
}
```

The sum of the printed values is _____.

[MCQ]

5.
```
#include <stdio.h>
#include <stdlib.h>
int main() {
```

```
int count=0;
char *p=(char *)malloc(sizeof(char));
*p=65;
printf("%c",*p);
p=realloc(p, 4*sizeof(char));
*p=256;
printf("%d",*(int*)p);
return 0;
}
```

The output printed is-

(a) A followed by Garbage values
(b) A0
(c) A512
(d) Compilation error

[MCQ]

6.

```
#include <stdio.h>
#include <stdlib.h>
int * f()
{
   int *p=(int*)malloc(sizeof(int));
   *p=20;
   return p;
}
int * g()
{
   static int a=10;
   int *q;
   q=&a;
   return q;
}
int main()
{
   printf("%d\t", *g());//line 1
   printf("%d", *f());//line 2
   return 0;
}
```

The output is-

(a) Garbage value
(b) Compilation error
(c) 10 20
(d) 20 10

[MCQ]

7. When the memory is full, malloc returns-

(a) Void pointer
(b) Wild pointer
(c) Dangling pointer
(d) NULL pointer

[MCQ]

8.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int *p=(int *)calloc(2, sizeof(int));
   int *q;
   q=p+1;
   printf("%d\t", *p);
   printf("%d\t", *q);
   *p=10;
   *q=15;
   printf("%d\t", *p);
   printf("%d\t", *q);
   free(p);
   return 0;
}
```

The output is:

(a) 10 15 Garbage 15
(b) Garbage Garbage 10 15
(c) 0 0 10 15
(d) 10 15 0 0

Answer Key

1.	**(c)**	**5.**	**(b)**
2.	**(a, c, d)**	**6.**	**(c)**
3.	**(c)**	**7.**	**(d)**
4.	**(463)**	**8.**	**(c)**

Hints and solutions

1. (c)

P: void *p;
p=malloc(1);
*p=65; //Invalid use of void expression. Void pointer stores the address of any variable and needs proper typecasting.
printf("%c",*(char*)p);
Hence, P is **invalid**.

Q: void *p;
char a='A';
p=malloc(1);
p=&a; //Void pointer p is storing the address of char variable 'a'.
printf("%c",*(char*)p);//Proper type casting is done hence, the code Q is **valid**..

2. (a, c, d)

%p is the format specifier for hexadecimal memory address.
The function g() is returning the address of local variable 'a' which will go out of scope as soon as g() finishes execution.
So, Line 2 in the main() given as-
printf("%p", g(15));
will give ERROR.

3. (c)

The binary of 324 is $(101000100)_2$.
The address of the variable a containing 324 is stored in void pointer p.
It is type-casted to char* i.e. it dereferences only 8 bits from right i.e $(1000100)_2$. Its decimal equivalent is 68.
Output: 68

4. (463)

*p=376
*q=5
while(376>5)→True
*p=376→printf() executed.
*p=376/5=75
*q=5+1=6

while(75>6)→True
*p=75→printf() executed.
*p=75/6=12
*q=6+1=7

while(12>7)→True
*p=12→printf() executed.
*p=12/7=1
*q=7+1=8

while(1>8)→False
STOP.

Output: 376 75 12

Sum: 463

5. (b)

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  char *p=(char *)malloc(sizeof(char));
  *p=65;
  printf("%c",*p); //A
  p=realloc(p, 4*sizeof(char)); //realloc reallocates the
   memory with size equivalent to 4 bytes and stores
   the address in pointer variable 'p'.
  *p=256;
  printf("%d",*(int*)p); // This will result into
   overflow.
  return 0;
}
```

Output: A0

6. (c)

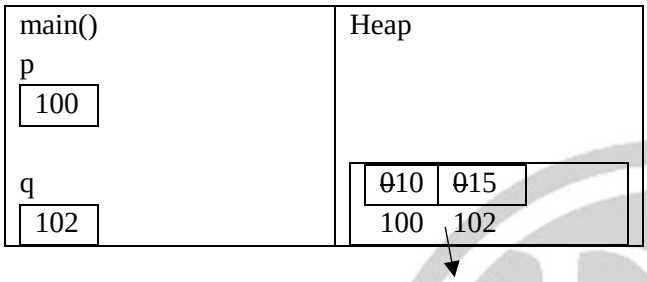
A static variable has scope throughout the program.
The function g() is returning the address of static variable 'a'.
printf("%d\t", *g());//10
printf("%d", *f());//20
Output: 10 20

7. (d)

When the memory is full, malloc() returns NULL pointer.

8. (c)

calloc() allocates two continuous memory block of the sizes equivalent to store an integer. It returns the address to integer pointer p.



printf("%d\t", *p);//0
printf("%d\t", *q);//10
*p=10;
*q=15;
printf("%d\t", *p);
//10
printf("%d\t", *q);
//15
free(p); //free(100)
//Entire block allocated by calloc() is freed.
	Free(100) frees the entire space.

Output: 0 0 10 15