(1)
What is meaning of following declaration?
```
int(*ptr[5])();
```

(a)ptr is pointer to function.
(b)ptr is array of pointer to function
(c)ptr is pointer to such function which return type is array.
(d)ptr is pointer to array of function.
(e)None of these
Ans: (b)
Explanation:
Here ptr is array not pointer.

(2)
What is meaning of following pointer declaration?
```
int(*(*ptr1)())[2];
```

(a)ptr is pointer to function.
(b)ptr is array of pointer to function
(c)ptr is pointer to such function which return type is pointer to an array.
(d)ptr is pointer array of function.
(e)None of these
Answer: (c)
Exp:

(3)What is size of generic pointer in c?
(a)0
(b)1
(c)2
(d)Null
(e)Undefined
Ans: (c)

Size of any type of pointer is 2 byte (In case of near pointer)
Note. By default all pointers are near pointer if default memory model is small.
(4)

What will be output of following c code?

```
#include<stdio.h>
int main(){
    int *p1,**p2;
    double *q1,**q2;
    clrscr();
    printf("%d %d ",sizeof(p1),sizeof(p2));
    printf("%d %d",sizeof(q1),sizeof(q2));
    getch();
    return 0;
}
```

(a)1 2 4 8
(b)2 4 4 8
(c)2 4 2 4
(d)2 2 2 2
(e)2 2 4 4
Answer: (d)
Exp:
Size of any type of pointer is 2 byte (In case of near pointer)

(5)
What will be output if you will compile and execute the following c code?

```
#include<stdio.h>
int main(){
    char huge *p=(char *)0XC0563331;
    char huge *q=(char *)0XC2551341;
    if(p==q)
        printf("Equal");
    else if(p>q)
        printf("Greater than");
    else
        printf("Less than");
    return 0;
}
```

(a)Equal
(b)Greater than
(c)Less than
(d)Compiler error
(e)None of above
Output: (a)

Explanation:
As we know huge pointers compare its physical address.
Physical address of huge pointer p
Huge address: 0XC0563331
Offset address: 0x3331
Segment address: 0XC056
Physical address= Segment address * 0X10 + Offset address
=0XC056 * 0X10 +0X3331
=0XC0560 + 0X3331
=0XC3891
Physical address of huge pointer q
Huge address: 0XC2551341
Offset address: 0x1341
Segment address: 0XC255
Physical address= Segment address * 0X10 + Offset address
=0XC255 * 0X10 +0X1341
=0XC2550 + 0X1341
=0XC3891
Since both huge pointers p and q are pointing same physical address so if condition will true.

(6)
What will be output if you will compile and execute the following c code?

```c
#include<stdio.h>
int main(){
    int a=5,b=10,c=15;
    int *arr[]={&a,&b,&c};
    printf("%d",*arr[1]);
    return 0;
```

```
}
```

(a)5
(b)10
(c)15
(d)Compiler error
(e)None of above
Output: (d)
Explanation:
Array element cannot be address of auto variable. It can be address of static or extern variables.

(7)
What will be output if you will compile and execute the following c code?

```
#include<stdio.h>
int main(){
    int a[2][4]={3,6,9,12,15,18,21,24};
    printf("%d %d %d",*(a[1]+2),*(*(a+1)+2),2[1[a]]);
    return 0;
}
```

(a)15 18 21
(b)21 21 21
(c)24 24 24
(d)Compiler error
(e)None of above
Output: (b)
Explanation:
In c,
a [1][2]
=*(a [1] +2)
=*(*(a+1) +2)
=2[a [1]]
=2[1[a]]
Now, a [1] [2] means 1*(4) +2=6th element of an array staring from zero i.e. 21.

(8)
What will be output if you will compile and execute the following c code?

```
#include<stdio.h>
int main(){
    const int x=25;
    int * const p=&x;
    *p=2*x;
    printf("%d",x);
    return 0;
}
```

(a)25
(b)50
(c)0
(d)Compiler error
(e)None of above
Output: (b)
Explanation:

const keyword in c doesn't make any variable as constant but it only makes the variable as read only. With the help of pointer we can modify the const variable. In this example pointer p is pointing to address of variable x. In the following line:
`int * const p=&x;`
p is constant pointer while content of p i.e. *p is not constant.
*p=2*x put the value 50 at the memory location of variable x.
(9)

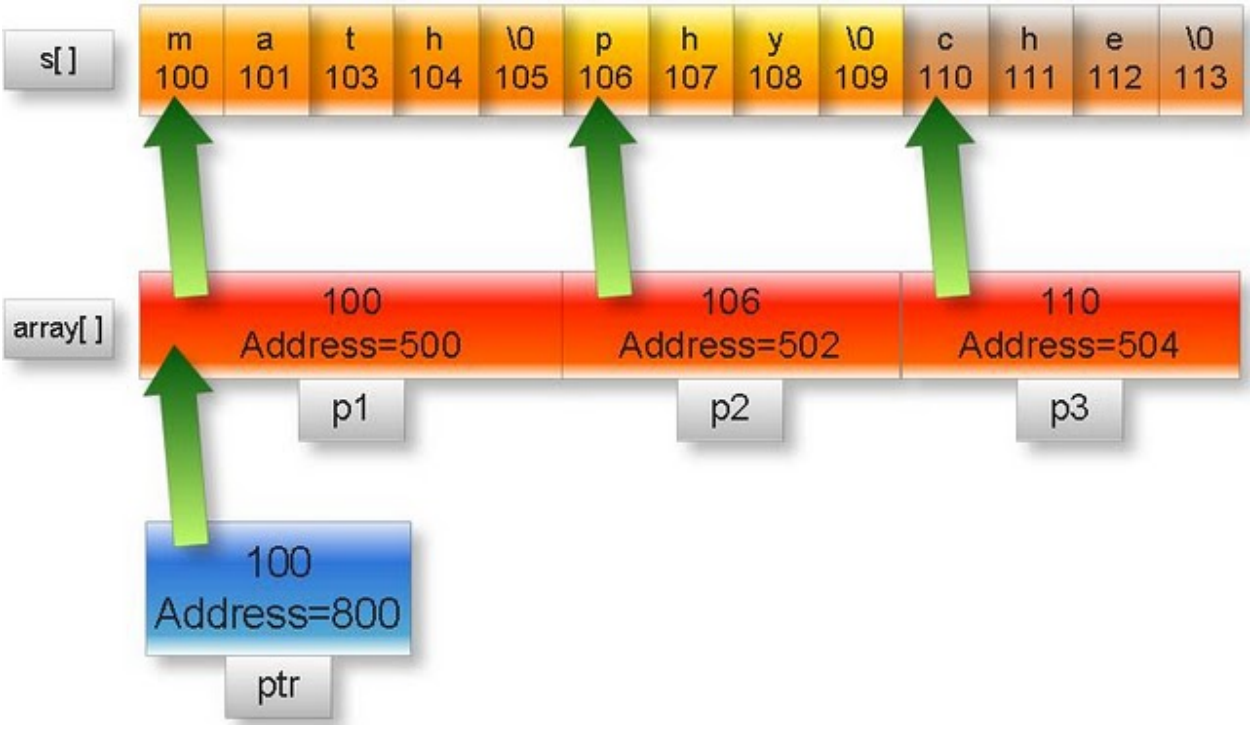
What will be output if you will compile and execute the following c code?

```
#include<stdio.h>
int main(){
    static char *s[3]={"math","phy","che"};
    typedef char *( *ppp)[3];
```

```
    static ppp p1=&s,p2=&s,p3=&s;
    char * (*(*array[3]))[3]={&p1,&p2,&p3};
    char * (*(*(*ptr)[3]))[3]=&array;
    p2+=1;
    p3+=2;
    printf("%S",(***ptr[0])[2]);
    return 0;
}
```

(a) math
(b) phy
(c) che
(d) Compiler error
(e) None of these
Answer: (c)

Explanation:
Here ptr is pointer to array of pointer to string. P1, p2, p3 are pointers to array of string. array[3] is array which contain pointer to array of string.
Pictorial representation:



Note: In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

As we know p[i]=*(p+i)
(***ptr[0])[2]=(*(***ptr+0))[2]=(***ptr)[2]
=(***(&array))[2] //ptr=&array
=(**array)[2] //From rule *&p=p
=(**(&p1))[2] //array=&p1
=(*p1)[2]
=(*&s)[2] //p1=&s
=s[2]=”che”

(10)
What will be output if you will compile and execute the following c code?

```
#include<conio.h>
#include<stdio.h>
int display();
int(*array[3])();
int(*(*ptr)[3])();
int main(){
```

```
    array[0]=display;
    array[1]=getch;
    ptr=&array;
    printf("%d",(**ptr)());
    (*(*ptr+1))();
    return 0;
}

int display(){
    int x=5;
    return x++;
}
```

(a)5
(b)6
(c)0
(d)Compiler error
(e)None of these
Answer: (a)

Explanation:
In this example:
array []: It is array of pointer to such function which parameter is void and return type is int data type.
ptr: It is pointer to array which contents are pointer to such function which parameter is void and return type is int type data.

```
(**ptr)() = (** (&array)) () //ptr=&array
= (*array) () // from rule *&p=p
=array [0] () //from rule *(p+i)=p[i]
=display () //array[0]=display
(*(*ptr+1))() =(*(*&array+1))() //ptr=&array
=*(array+1) () // from rule *&p=p
=array [1] () //from rule *(p+i)=p[i]
```

```
=getch () //array[1]=getch
```