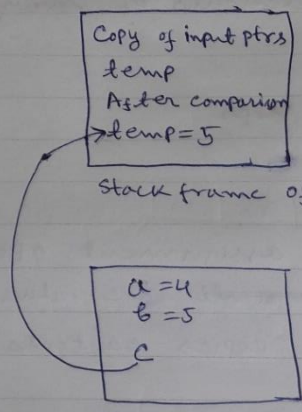
# Solutions of Some of the Practice Problems of Week 2

# Week 2

(96)




This problem was covered during online session.

c points to 'temp' which resides in the stack frame of max(). ↘ becomes invalid after max() completes.

Conclusion: c points to an invalid object.

(97) Answer is in the lecture.

(98)

```
for (i=0; i<4; i++){
    if ( (P[i] = (int*) malloc(4*sizeof(int))) == NULL){
        printf ("Allocation error");
        exit (-1);
    }
}
for (i=0; i<4; i++)
    for (j=0; j<4; j++)
        P[i][j] = i*4 + j;
for (i=0; i<4; i++){
    for (j=0; j<4; j++)
        Printf ("%d \t", P[i][j]);
}
for (i=0; i<4; i++)
    free (P[i]);
return 0;
```

(Note: you can use pointer with [])

(Q9) foo1() is returning a pointer to a local object x. x becomes invalid after foo1() completes.

(Q10) Answer in ~~the~~ lecture materials.
use → with pointers.

(Q11) ~~g3()~~ g1() returns pointer to local object.
g(2) has a bigger problem.
No memory is allocated for px. Yet some assignment *px = 10 is made. [Program will cause Segfault].

g3() looks fine. It allocates heap memory and then returns a pointer to that memory.

```
main(){
    int *pm;
    pm = g3();
    ...
    free(pm);
}
```

The caller function has the pointer. It can free the memory.

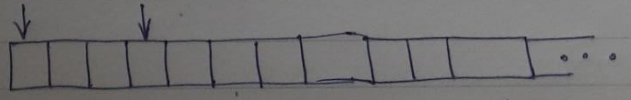
(Q12) Obvious. Leaks memory. No free() is called.

(Q13) This is a tricky question. It was explained in detail during online session.

(Q14) Lecture material.

(Q23) A matrix such as int a[2][3] is stored in row-major order ~~in~~ in the memory.
If you access the elements along a row, then the program benefits from spatial locality.

In the above example code, the elements are accessed "column major" order. So, doesn't benefit from spatial locality.

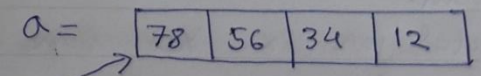


Column-major access can't exploit spatial locality.

Radboud Universiteit

(Q19) ~~This~~ A similar example was covered during the Online session of Week 1.

Remember the two-party communication over the internet.

a = 


(Assuming Little endian)

Pa

~~int *Pa = &a;~~ char *Pa = &a;

~~int *Pb = &b;~~ char *Pb = &b;

```
for (i=0; i<sizeof(int); i++)
    Pb[i] = Pa[i];
printf ("b = %x", b);
```

You can also use

char *Pa = (char*) &a;

type casting to avoid warning.