# Some Practice Problems for Week 2

Try to answer the following questions **only after studying** the lecture-contents. You may search the internet while solving these practice problems.

Q1. This question is about memory layout of a typical C program. What are the segments of a C program?

Q2. Describe the differences between local, heap, global and static variables.

Q3. With an example, describe how stack frames are created and destroyed during function calls.

Q4. Describe the consequences of returning a pointer to a local variable from a function call. Give an example.

Q5. Describe the differences between pass-by-value and pass-by-pointer.

Q6. What is wrong with this program?

```
int *max(int*a, int*b){
      int temp;
      if(*a >*b)
            temp=*a;
      else
            temp=*b;
      return &temp
}
int main(){
      int a=4,b=5;
      int*c;
      c=max(&a,&b);
      printf("Max value=%d",*c);
      return0;
}
```

Q7. This question is about dynamic memory allocation. Write C language syntax for dynamically allocating an integer array of length LENGTH using malloc(). What happens when malloc() fails to allocate memory?

Q8. The following unfinished C program declares an array of four pointers in line 5. Using nested for loops with counters i and j, complete the unfinished program to construct the two-dimensional matrix

```
00 01 02 03
04 05 06 07
08 09 10 11
12 13 14 15
```

such that p[i] points to the i-th row of the matrix. Do not forget to free the allocated memory in the end.

```c
#include<stdio.h>
#include<stdlib.h>
int main(){
    int i, j;
    int *p[4];
    /*

    your code

    */

    return 0;
}
```

Q9. What is wrong with these function calls?

```c
int *foo1(){
    int x = 20;
    return &x;
}
int *foo2(){
    int *p;
    *p = 20;
    return p;
}
```

Q10. In the following code snippet, a structure object is passed to a function foo() by value.

```
struct VectorPair{
 int a[512];
 int b[512];
};

VectorPair foo(VectorPair VP1){
     VectorPair VP2;
     int i;
     for(i=0; i<512; i++){
          VP2.a[i] = VP1.a[i]+VP1.b[i];
          VP2.b[i] = VP1.a[i]-VP1.b[i];
     }
     return VP2;
}
```

Rewrite the function foo so that objects are passed using C style pointers.

Q11. Consider the following three C functions:

```
int * g1 (void)
{
  int x= 10;
  return (&x);
}
int * g2 (void)
{
  int * px;
  *px= 10;
  return px;
}
int *g3 (void)
{
  int *px;
  px = (int *) malloc (sizeof(int));
  *px= 10;
  return px;
}
```

Which of the above three functions are likely to cause problems with pointers?

Q12. Does this program leak memory?

```
int main(){
     int *p;
     p = (int*)malloc(sizeof(int));
     *p = 6;
     printf("%d",*p);
     return(0);
```

```
}
```

Q13. Does this program function correctly? Why or why not?

```
void foo(int *a){
      a = (int*)malloc(sizeof(int));
}

int main(){
      int *p;
      fun(p);
      *p = 6;
      printf("%d",*p);
      return(0);
}
```

Q14. In C, a pointer to a pointer variable is declared using **. How many bytes are allocated, deallocated and leaked in this program?

```
int main()
{
      int *p1, **p2;
      p1 = malloc(sizeof(int));
      *p1 = 5;
      p2 = malloc(sizeof(int*));
      *p2 = p1;
      free(p1)
      return 0;
}
```

Q15. What is the ‘double free’ problem?

Q16. Type casting in C refers to changing a variable of one data type into another. With examples, describe advantages and pitfalls of typecasting in C.

Q17. Describe the difference between ‘little’ and ‘big’ endian representations.

Q18. How many bytes are there in `unsigned int a = 0x12345678;`
Print the bytes one-by-one.

Q19. You have two 32-bit integer variables.

```
unsigned int a = 0x12345678;
unsigned int b;
```

Copy ‘a’ into ‘b’ *byte-by-byte* such that in the end of the copy operation, we have a and b equal.

Q20. What is the memory hierarchy of a computer? What are the advantages of using a ‘memory hierarchy’?

Q21. What is the locality of reference?

Q22. With a diagram, show the memory layout of a 2D matrix.

Q23. The following program negates a 2D array.

```
int sum_array(int a[N][M]){
      int i,j;
      for(i=0;i<M;i++)
            for(j=0;j<N;j++)
                  a[j][i] = -a[j][i];

}
```

Why does this program cannot exploit memory hierarchy?

Rewrite the program such that it can exploit the memory hierarchy and thus become faster.