Question 1

Which of the following represents the most correct use of malloc?

- `int x = malloc(sizeof(int));`
- `int *x = (void *)malloc(sizeof(char));`
- `int *x = (int *)malloc(sizeof(int));`
- `int *x = (int *)malloc(4);`

https://www.cs.virginia.edu/~jh2jf/courses/cs2130/spring2023/exams/s2022e2key.pdf

Question 2

In the following code snippet, where are a, b, and c stored in memory?

```
int foo() {
  int a = 9;
  int b[3] = {2, 7, 8};
  char *c = (char *) malloc(100);
}
```

- b and c are stored on the stack, a is stored on the heap
- b is stored on the stack, a and c are stored on the heap
- a is stored on the stack, b and c are stored on the heap
- a and b are stored on the stack, c is stored on the heap

Question 3

6. [2 points] Assuming the following line of code is inside the main( ) function, in which part of memory is the pointer variable `parray` allocated and in which part of memory is the 10 element integer array allocated?

```
int *parray = malloc (sizeof (int) * 10);
```

https://pages.cs.wisc.edu/~remzi/Classes/354/Spring2017/OldExams/midterm-spring-16.pdf

Question 4

```c
int main() {
    char *p;
    p = (char *) malloc(sizeof(char));
    *p = 'a';

    p = (char *) malloc(sizeof(char));
    *p = 'c';

    return 0;
}
```

A. Syntax error in `*p = 'a';`
B. Memory leak
C. Dangling pointer
D. No issue with code

Question 5

```
int main() {
    int *x;

    do {

        x = (int *)malloc(sizeof(int));
        printf("Enter an integer (0 to stop): ");
        scanf("%d", x);
        printf("You entered %d\n", *x);

    } while (*x != 0);

    free(x);

    return 0;
}
```

https://people.cs.pitt.edu/~aus/cs449/ts-midterm1-sample-solution.pdf

Question 6

```
int *p, *q, *r;

p = malloc(8);

...
q = p;

...

free(p);
r = malloc(8);

...
*q=5;
```

Question 7

```
int *q;


void foo() {
  int a;
  q = &a;
}


int main() {
  foo();
  /* ... */
  *q = 5;
}
```

Question 8

Consider the C program fragment which uses a function foo.

```
main(){
    int *p = foo();
    p = NULL;
    //some other stuff
}
```

foo can be implemented in various following ways, all pieces of code are identical except for their use of free(). Each of them may be correct or they may have a memory leak, dangling pointer or both.

You should provide an answer for the complete program, which includes combining the main function with a specific implementation of the function foo.

https://ubccsss.org/files/213-2015-mt-soln.pdf

C Programming

P1:

```
int *copy(int *src) {
    int *dst = malloc(sizeof(int));
    *dst = *src;
    return dst;
}
int foo() {
    int a = 3;
    int *b = copy(&a);
    return * b;
}
```

P2:

```
int *copy(int *src) {
    int *dst = malloc(sizeof(int));
    *dst = *src;
    free (dst);
    return dst;
}
int foo() {
    int a = 3;
    int *b = copy( &a);
    return * b;
}
```

P3:

```
int *copy(int *src) {
    int *dst = malloc(sizeof(int));
    *dst = *src;
    return dst;
}
int foo() {
    int a = 3;
    int *b = copy( &a);
    free (b);
    return * b;
}
```

P4:

```
int *copy(int *src) {
    int *dst = malloc(sizeof(int));
    *dst = *src;
    free (dst);
    return dst;
}
int foo() {
    int a = 3;
    int *b = copy( &a);
    free (b);
    return * b;
}
```

6 (6 marks) **Dynamic Allocation.** The following four pieces of code are identical except for the their use of `free()`. Each of them may be correct or they may have a memory leak, dangling pointer or both. In each case, determine whether these bugs exists and if so, briefly describe the bug(s); do not describe how to fix the bug.

6a

```
int* copy (int* src) {
  int* dst = malloc (sizeof (int));
  *dst = *src;
  return dst;
}
```

```
int foo() {
  int  a = 3;
  int* b = copy (&a);
  return *b;
}
```

Memory leak, because object allocated in `copy` is not freed in the shown code and when `foo` returns it is unreachable.

6b

```
int* copy (int* src) {
  int* dst = malloc (sizeof (int));
  *dst = *src;
  free (dst);
  return dst;
}
```

```
int foo() {
  int  a = 3;
  int* b = copy (&a);
  return *b;
}
```

Dangling pointer. After `free` in `copy`, `dst` is a dangling pointer. This value is returned by copy and so `b` in `foo` is also a dangling pointer. The last statement of `foo`, `return *b` dereferences this dangling pointer.

6c

```
int* copy (int* src) {
  int* dst = malloc (sizeof (int));
  *dst = *src;
  return dst;
}
```

```
int foo() {
  int  a = 3;
  int* b = copy (&a);
  free (b);
  return *b;
}
```

Dangling pointer. After `free` in `foo`, `b` becomes and dangling pointer and it is then dereferenced in the last statement.

6d

```
int* copy (int* src) {
  int* dst = malloc (sizeof (int));
  *dst = *src;
  free (dst);
  return dst;
}
```

```
int foo() {
  int  a = 3;
  int* b = copy (&a);
  free (b);
  return *b;
}
```

Dangling pointer. After `free` in `copy`, `dst` becomes a dangling pointer. This value is returned by `copy` and so `b` in `foo` is also a dangling pointer. The third statement of `foo` then calls `free` again on this value, attempting to free an object that has already been freed, which results in an error. If the program where to proceed it would then dereference the dandling pointer in the `return` statement.

Solution

https://ubccsss.org/files/213-2015-mt-soln.pdf

Question 9

2. Given the declarations

```
int* ptrA;
int* ptrB;
```

Which of the following does NOT have memory leak ?

Note: Each code segment denoted by (a), (b), (c) and (d) is independent.

A

```
ptrA = malloc(4);
ptrB = malloc(4);

*ptrA = 345;
ptrB = ptrA;
free(ptrA);
free(ptrB);
```

B

```
ptrA = malloc(4);
ptrA = 345;

ptrB = ptrA;
free(ptrA);
```

C

```
ptrA = malloc(4);
ptrB = malloc(4);


*ptrA = 345;
*ptrB = *ptrA;
free(ptrA);
free(ptrB);
```

D

```
ptrA = malloc(4);
ptrB = malloc(4);

*ptrA = 345;
ptrB = malloc(4);
*ptrB = *ptrA;
free(ptrA);
free(ptrB);
```

Question 10

```c
#include <stdio.h>

void we(void) {
    int *ptr;
    {
        int x;
        ptr = &x;
    }
    *ptr = 3;
}

void main() {
    we();
}
```

Question 11

Which code fragment produces a dangling pointer?

a

```
int* f(void)
{    int d;
     return &d;
}
int main()
{    int* p=f();
     return 0;
}
```

b

```
int* f(void){return  malloc(4);  }
int main()
{    int* p=f();
     int  q;
     p=&q;
     return 0;
}
```

c

```
int main()
{    int* p= malloc(4);
     int* q=p;
     free(q);
     return 0;
}
```

d

```
int* f(void){return  malloc(4);  }
int main()
{    int* p=f();
     p= malloc(4);
     return 0;
}
```

Question 12

```
int *p = (int*)malloc(sizeof(int));
int *q = (int*)malloc(sizeof(int));
int *r;


*p = 17;
r = q;
*q = 42;
p = q;
free(r);
```

Question 13

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *p = malloc(sizeof(int));
    *p = 42;
    p = malloc(sizeof(int));
    free(p);
}
```

https://pages.cs.wisc.edu/~remzi/Classes/354/Spring2017/OldExams/midterm-spring-16.pdf

Solution

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *p = malloc(sizeof(int));
    *p = 42;
    p = malloc(sizeof(int));
    free(p);
}
```

ANSWER:

There is a **memory leak** in this program since we overwrite the pointer p before freeing the memory location it points to.

Solution: add `free(p)` before the second malloc.

Question 14

```c
#include <stdio.h>


int main() {
    int *x;
    for (int i = 0; i <= 50; i++) {
        x = (int*)malloc(sizeof(int));
        *x = i;
        printf("%d\n", *x);
    }


       free(x);          // Free the memory allocated for x

    return 0;
}
```

https://www.ecb.torontomu.ca/~courses/coe808/Midterm.pdf

Question 15

Are there any dynamic memory management errors in the following code?

```
int *p = malloc(4);
int *q = malloc(4);
int *r;
*p = 17;
r = q;
*q = 42;
p = q;
free(r);
```

A. No, there are no errors
B. Yes, a memory leak
C. Yes, misuse of a dangling pointer
D. Yes, both a memory leak and misuse of a dangling pointer

https://courses.cs.washington.edu/courses/cse143/00au/exam_quiz/finalsol.pdf

Question 16

Consider the code:

```
struct node * ptr = (struct node *) malloc(sizeof (struct node));
printf("%p", ptr); // LINE 2;    ASSUME THIS PRINTS 0x522f1c0
free(ptr)
printf("%p", ptr); // LINE 4
```

What happens when line 4 executes?

- ○ It may crash (with segfault or other error meesage)
- ○ It will print NULL. Nothing crashes.
- ○ It will print the same as line 2 (0x522f1c0). Nothing crashes.
- ○ It will print a random memory address. Nothing crashes.

Solution

Consider the code:

```
struct node * ptr = (struct node *) malloc(sizeof (struct node));
printf("%p", ptr); // LINE 2;    ASSUME THIS PRINTS 0x522f1c0
free(ptr)
printf("%p", ptr); // LINE 4
```

What happens when line 4 executes?

- ○ It may crash (with segfault or other error meesage)
- ○ It will print NULL. Nothing crashes.
- ◉ It will print the same as line 2 (0x522f1c0). Nothing crashes.
- ○ It will print a random memory address. Nothing crashes.

```c
#include<stdio.h>

int *assignval (int *x, int val) {
    *x = val;
    return x;
}

void main () {
    int *x = malloc(sizeof(int));
    if (NULL == x) return;
    x = assignval (x,0);
    if (x) {
        x = (int *)malloc(sizeof(int));
        if (NULL == x) return;
        x = assignval (x,10);
    }
    printf("%d\n", *x);
    free(x);
}
```

Question 17

GATE 2017

Options for previous question:

The code suffers from which one of the following problems:

A. compiler error as the return of $malloc$ is not typecast appropriately.
B. compiler error because the comparison should be made as $x == NULL$ and not as shown.
C. compiles successfully but execution may result in dangling pointer.
D. compiles successfully but execution may result in memory leak.

[P1]

```
int *g(void)
{
    int x = 10;
    return (&x);
}
```

[P2]

```
int *g(void)
{
    int *px;
    *px = 10;
    return px;
}
```

[P3]

```
int *g(void)
{
    int *px;
    px = (int*) malloc (sizeof(int));
    *px = 10;
    return px;
}
```

Question 18

GATE 2001

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

A. Only $P3$

B. Only $P1$ and $P3$

C. Only $P1$ and $P2$

D. $P1, P2$ and $P3$

Question 19

4. What is the output of the following program?

```
#include <stdio.h>

main(void) {
  static char *array[3][3]=
    {{":|(",  "Hope", "you"},
     {"have", "a",    "good"},
     {"fall", "break",":|)"}};

  char *(*p)[3]=array;
  char **q=*(array+1);

  printf("1 : %s\n",array[0][2]);
  printf("2 : %s\n",*(array[1]));
  printf("3 : %s\n",*(*(array+1)+1));
  printf("4 : %s\n",*(q+2));
  printf("5 : %s\n",*(*(p+2)+1));
}
```

https://www.ndsl.kaist.edu/~kyoungsoo/ee209_2011/oldmidterm/fall00exam1.pdf

Question 4

```
1 : you
2 : have
3 : a
4 : good
5 : break
```

Explanation:

```
array[0][2]
= "you"

*(array[1])
= *(array[1] + 0)
= array[1][0]
= "have"

*(*(array + 1) + 1)
= *(array[1] + 1)
= array[1][1]
= "a"

*(q + 2)
= *(*(array + 1) + 2)
= *(array[1] + 2)
= array[1][2]
= "good"

*(*(p + 2) + 1)
= *(*(array + 2) + 1)
= *(array[2] + 1)
= array[2][1]
= "break"
```

Question 20

```c
#include <stdio.h>
int main (void)
{
    int a[3] = {0, 1, 2};
    int *p[3] = {a, a+2, a+1};
    printf("%d  %d  %d\n", *(p[0]), *(p[1]), *(p[2]));


    *(p[1]) = 3;


    printf("%d  %d  %d\n", a[0], a[1], a[2]);
    return 0;
}
```

Question 21

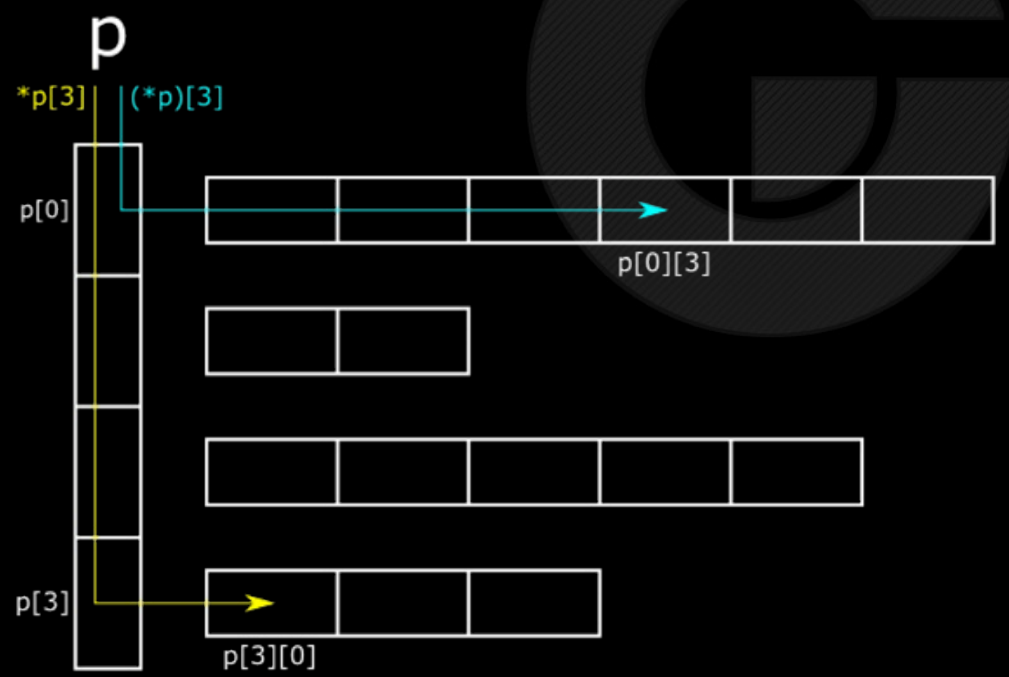
```
#include<stdio.h>
main()
{
char a[] = "GATE";
char b[] = "GO";
char c[] = "Overflow";
char d[] = "Classes";


char *p[] = {a, b, c, d};
printf("%c", *p[3]);
printf("%c", (*p)[3]);


}
```

Hint/Solution

Note: [ ] has higher precedence than *. We will study this in subsequent classes.

The order of evaluation differs. `*p[3]` resolves to `*(p[3])` but not `(*p)[3]`. Also note that `*p` is equivalent to `p[0]`. Therefore, ...

- `*p[3]` ⇔ `*(p[3])` ⇔ `(p[3])[0]` ⇔ `p[3][0]`
- `(*p)[3]` ⇔ `(p[0])[3]` ⇔ `p[0][3]`



Question 22

```
main()
{
    int c[3][4] =
    {{1, 3, 0,-5}, {-1, 5, 9, 8}, {3, 5, 99,7}};

    printf("%u\n", sizeof(c));

    fun (c);

    //what is content of c here ?

    for(int i =0; i<3; i++){
        for(int j =0; j<4; j++)
            printf("%d ", c[i][j]);
        printf("\n");
    }
}
```

```
void  fun (int (*c)[4]){
    printf("%u\n", sizeof(c));

    (*c+1)[2] = 4;
    c++;
    *c[1] = 6;
    (*c)[1] = 8;
}
```

Question 23

```
typedef struct Point { int x; int y; } Point;

void changeX1 (Point pt)  { pt.x = 11; }
void changeX2 (Point *pt) { pt->x = 33; }
void changeX3 (Point *pt) { (*pt).x = 44; }

int main() {
    Point my_pt = {1,2};
    changeX1(my_pt);
    printf("%d\n", my_pt.x);

    my_pt.x = 1,my_pt.y = 2;
    changeX2(&my_pt);
    printf("%d\n", my_pt.x);

    my_pt.x = 1,my_pt.y = 2;
    changeX3(&my_pt);
    printf("%d\n", my_pt.x);

}
```

Question 24

```
typedef struct {
    int x, y;
} point;
```

```
void bar(point* p1, point* p2) {
    point p = { p1->x, p2->y };
    p.x = 5;
    p.y = 6;
    *p2 = p;
    p1 = &p;
}

int main() {
    point p1 = { 0, 0 };
    point p2 = { 1, 2 };
    bar(&p1, &p2);

    printf("p1.x = %d p1.y = %d\n", p1.x, p1.y);

    printf("p2.x = %d p2.y = %d\n", p2.x, p2.y);

    return 0;
}
```

Question 25

```c
#include<stdio.h>

typedef struct {
    int x, y;
} point;


point* create_point(int x, int y) {
    point p = {x, y};
    point* ptr = &p;
    return ptr;
}


int main() {
    point* pl = create_point(1, 2);
    printf("%d, %d", pl->x, pl->y);
    return 0;
}
```