This page is blank.

1 GDB (10 points)

Consider the following program:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void swap1(int a, int b)
4  {
5    int c = a;
6    a = b;
7    b = c;
8  }
9
10 void swap2(int * a, int * b)
11 {
12   int c = * a;
13   * a = * b;
14   * b = c;
15 }
16
17 int main(int argc, char * * argv)
18 {
19   int s = 264;
20   int t = 7012;
21   swap1(s, t);
22   printf("s = %d, t = %d\n", s, t);
23   // restore the values
24   s = 264;
25   t = 7012;
26   swap2(& s, & t);
27   printf("s = %d, t = %d\n", s, t);
28   return EXIT_SUCCESS;
29 }
```

Here are some frequently used commands:

b	break	r	run	c	continue
n	next	s	step	f	frame

Suppose you run the following GDB commands ("(gdb)" is the prompt).

```
(gdb) b swap1
(gdb) r
(gdb) b 7
(gdb) c
(gdb) print a
```

1.A: What is the ouput?

Continue with the following GDB commands

```
(gdb) f 1
22   swap1(s, t);
(gdb) print s
```

1.B: What is the ouput?

```
(gdb) b 14
(gdb) c
(gdb) print c
```

1.C: What is the ouput?

```
(gdb) n
(gdb) f 1
(gdb) print t
```

1.D: What is the ouput?

```
(gdb) c
```

The output is

```
s = ???
```

The answer is 1.E. Do not worry about the value of t.
Answer:

```
A: 7012
B: 264
C: 264
D: 264
E: 7012
```

2 Structure (40 points)

Consider the following program:

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #pragma pack(1) // tell compiler not to pad any space
 4 // assume sizeof(char) = 1, sizeof(int) = 4,
 5 // sizeof(double) = 8, sizeof(a pointer) = 8
 6 #define NUMVECTOR 10
 7 typedef struct
 8 {
 9   double x;
10   double y;
11   double z;
12   double t;
13 } Vector;
14
15 void allocateVector(Vector * * ptraddr, int number)
16 {
17   Vector * ptr = malloc(sizeof(Vector) * number);
18   * ptraddr = ptr;
19 }
20
21 int main(int argc, char ** argv)
22 {
23   Vector * vptr;
24   printf("sizeof(vptr) = %ld\n", sizeof(vptr));
25   // 2.A, what is the output?
26
27   printf("sizeof(* vptr) = %ld\n",  sizeof(* vptr));
28   // 2.B, what is the output?
29
30   // ---> FIX ME <---
31   allocateVector(/* 2.C */, NUMVECTOR);
32
33   // assume the address of &vptr[0] is 1000 (decimal value)
34
35   printf("& vptr[1] = %p\n", (void *) & vptr[1]);
36   // 2.D, what is the output?
37   // Your anwer can be either decimal or hexadecimal.
38   // If you use hexadecimal, please add 0X prefix.
39   // Please be careful that 1000 decimal is not 0X1000
```

```
40
41    printf("& vptr[2].z = %p\n", (void *) & vptr[2].z);
42    // 2.E, what is the output?
43    // Your anwer can be either decimal or hexadecimal.
44    // If you use hexadecimal, please add 0X prefix
45
46    double * iptr;
47    iptr = & vptr[0].x;
48
49    iptr[10] = 264;
50
51    // 2.F, Which attribute of vptr has been changed?
52    // Write your answer in the form of vptr[A].B
53    // A is an index (between 0 and NUMVECTOR - 1)
54    // B is an attribute (x, y, z, or t)
55
56    // ---> FIX ME <---
57    // 2.G release memory
58
59    // 2.H If the memory is not released, how much
60    // memory is leaked?
61
62    return EXIT_SUCCESS;
63 }
```

Answer:

2.A 8
2.B 32
2.C & vptr
2.D 1032
2.E 1080
2.F vptr[2].z
2.G free (vptr);
2.H 320

3 File (40 points)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char * * argv)
4  {
5    // read from a file
6    // argv[1]: name of the file
7    // 3.A check whether argv[1] exists
8    if (/* ---> FIX ME <--- */)
9      {
10       return EXIT_FAILURE;
11     }
12   // 3.B open the file whose name is argv[1] for reading
13   // the result is stored in a pointer called fptr
14   // ---> FIX ME <---
15
16   // 3.C check whether fopen fails
17   if (/* ---> FIX ME <--- */)
18     {
19       return EXIT_FAILURE;
20     }
21
22   // Assume that if fopen succeeds,
23   // reading the data would be successful
24
25   // 3.D read one byte from the file using fgetc
26   // and store it in a variable called onebyte
27   // Hint: be careful about the type of onebyte
28   // ---> FIX ME <---
29   int oneint;
30
31   // 3.E read one integer from the file using fscanf
32   // ---> FIX ME <---
33
34   char wordone[80];
35   char wordtwo[80];
36
37   // 3.F read two words from the file using fscanf
38   // assume the two words are separated by space
39   // assume each word is no longer than 79 characters
40   // Your answer should read these two words using only
```

```
41    // one function call
42    // ---> FIX ME <---
43
44
45    // 3.G read the entire line (including space) using fgets
46    // assume the line is no longer than 160 characters
47    char oneline[160];
48    // ---> FIX ME <---
49
50    // 3.H close the file
51    // ---> FIX ME <---
52 }
```

```
int fgetc(FILE *stream);

fgetc() reads the next character from stream and returns it as an
unsigned char cast to an int, or EOF on end of file or error.

------------------------------
char *fgets(char *s, int size, FILE *stream);

fgets() reads in at most one less than size characters from stream and stores them
into the buffer pointed to by s. Reading stops after an EOF or a newline. If a
newline is read, it is stored into the buffer. A terminating null byte ('\0')
is stored after the last character in the buffer.

------------------------------
int fscanf ( FILE * stream, const char * format, ... );
The following conversion specifiers are available:
%

d

Matches an optionally signed decimal integer; the next pointer must be
a pointer to int.

s

Matches a sequence of non-white-space characters; the next pointer must be a
pointer to character array that is long enough to hold the input sequence and
the terminating null byte ('\0'), which is added automatically. The input string
stops at white space or at the maximum field width, whichever occurs first.
```

Answer:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char * * argv)
4  {
5    // read from a file
6    // argv[1]: name of the file
7    // 3.A check whether argv[1] exists
8    if (argc < 2)
9      {
10       return EXIT_FAILURE;
11     }
12
13   // 3.B open the file whose name is argv[1] for reading
14   FILE * fptr = fopen(argv[1], "r");
15
16   // 3.C check whether fopen fails
17   if (fptr == NULL)
18     {
19       return EXIT_FAILURE;
20     }
21
22   // 3.D read one byte from the file using fgetc
23   // and store it in a variable called onebyte
24   // Hint: be careful about the type of onebyte
25   int onebyte = fgetc(fptr);
26
27   int oneint;
28
29   // 3.E read one integer from the file using fscanf
30   fscanf(fptr, "%d", & oneint);
31
32   char wordone[80];
33   char wordtwo[80];
34
35   // 3.F read two words from the file using fscanf
36   // assume the two words are separated by space
37   // assume each word is no longer than 79 characters
38   fscanf(fptr, "%s %s", wordone, wordtwo);
39
40
41   // 3.G read the entire line (including space) using fgets
```

```
42    // assume the line is no longer than 160 characters
43    char oneline[160];
44    fgets(oneline, 160, fptr);
45
46    // 3.H close the file
47    fclose(fptr);
48 }
```

4 Memory (10 points)

Consider this program:

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int main(int argc, char * * argv)
 5 {
 6   // a two-dimensional array of integers
 7   // 5 rows and 3 columns
 8   int * * array2d;
 9   array2d = malloc(sizeof(int) * 5);
10   int row;
11   int col;
12   for (row = 0; row < 5; row ++)
13     {
14       array2d[row] = malloc(sizeof(int) * 3);
15     }
16   // assign zero to all elements
17   for (row = 0; row < 5; row ++)
18     {
19       for (col = 0; col < 3; col ++)
20         {
21           array2d[row][col] = 0;
22         }
23     }
24
25   // release memory
26   for (row = 0; row < 5; row ++)
27     {
28       free(array2d[row]);
29     }
30   free (array2d);
31   return EXIT_SUCCESS;
32 }
```

Running the program encounters `Segmentation fault (core dumped)`.
Using GDB and the result is

```
Program received signal SIGSEGV, Segmentation fault.
0x00000000004005ef in main (argc=1, argv=0x7fffffffe368) at q4.c:21
21   array2d[row][col] = 0;
```

This is the output of valgrind (the lines are truncated).

```
==7025== Invalid write of size 8
==7025==    at 0x4005AC: main (q4.c:14)
==7025==  Address 0x5204050 is 16 bytes inside a block of size 20 alloc'd
==7025==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
==7025==    by 0x40057F: main (q4.c:9)
==7025==
==7025== Use of uninitialised value of size 8
==7025==    at 0x4005EF: main (q4.c:21)
==7025==
==7025== Invalid read of size 8
==7025==    at 0x4005DF: main (q4.c:21)
==7025==  Address 0x5204058 is 4 bytes after a block of size 20 alloc'd
==7025==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
==7025==    by 0x40057F: main (q4.c:9)
==7025==
--7025-- REDIR: 0x4ebe4f0 (libc.so.6:free) redirected to 0x4c2ed80 (free)
==7025== Conditional jump or move depends on uninitialised value(s)
==7025==    at 0x4C2EDA1: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==7025==    by 0x400630: main (q4.c:28)
==7025==
==7025== Invalid read of size 8
==7025==    at 0x400626: main (q4.c:28)
==7025==  Address 0x5204058 is 4 bytes after a block of size 20 alloc'd
==7025==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
==7025==    by 0x40057F: main (q4.c:9)
==7025==
==7025==
==7025== HEAP SUMMARY:
==7025==     in use at exit: 0 bytes in 0 blocks
==7025==   total heap usage: 6 allocs, 6 frees, 80 bytes allocated
==7025==
==7025== All heap blocks were freed -- no leaks are possible
==7025==
==7025== Use --track-origins=yes to see where uninitialised values come from
==7025== ERROR SUMMARY: 15 errors from 5 contexts (suppressed: 0 from 0)
==7025==
==7025== 1 errors in context 1 of 5:
==7025== Conditional jump or move depends on uninitialised value(s)
==7025==    at 0x4C2EDA1: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==7025==    by 0x400630: main (q4.c:28)
==7025==
==7025==
```

```
==7025== 2 errors in context 2 of 5:
==7025== Invalid read of size 8
==7025==    at 0x400626: main (q4.c:28)
==7025==  Address 0x5204058 is 4 bytes after a block of size 20 alloc'd
==7025==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
==7025==    by 0x40057F: main (q4.c:9)
==7025==
==7025==
==7025== 3 errors in context 3 of 5:
==7025== Use of uninitialised value of size 8
==7025==    at 0x4005EF: main (q4.c:21)
==7025==
==7025==
==7025== 3 errors in context 4 of 5:
==7025== Invalid write of size 8
==7025==    at 0x4005AC: main (q4.c:14)
==7025==  Address 0x5204050 is 16 bytes inside a block of size 20 alloc'd
==7025==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
==7025==    by 0x40057F: main (q4.c:9)
==7025==
==7025==
==7025== 6 errors in context 5 of 5:
==7025== Invalid read of size 8
==7025==    at 0x4005DF: main (q4.c:21)
==7025==  Address 0x5204058 is 4 bytes after a block of size 20 alloc'd
==7025==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
==7025==    by 0x40057F: main (q4.c:9)
==7025==
==7025== ERROR SUMMARY: 15 errors from 5 contexts (suppressed: 0 from 0)
```

The error is caused by **one single line** in the program. Please identify the line number (4.A) and write down the correct code (4.B). Please write the **entire line** for the answer.
Answer:
Line 9

```
array2d = malloc(sizeof(int* ) * 5);
```