**This page is blank.**

**Please read the entire question before you write down any answer.**

# 1 GDB

Consider the following program:

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 int f1(int a)
 4 {
 5   if (a <= 1)
 6     {
 7       return 1;
 8     }
 9   int b = f1 (a - 1) + a;
10   return b;
11 }
12
13 int f2(int a)
14 {
15   int b = f1(a);
16   int c = f1(a - 1);
17   int d = b + c;
18   return d;
19 }
20
21 int main(int argc, char * * argv)
22 {
23   int s = f2(5);
24   printf("s = %d\n", s);
25   return EXIT_SUCCESS;
26 }
```

Here are some frequently used commands:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| b | break | r | run | c | continue | n | next |
| s | step | f | frame | p | print | bt | backtrace |

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

```
(gdb) b f1
(gdb) r
(gdb) bt
```

What is the ouput? Answer A.

```
#0  f1 (a= ANSWER A) at q1.c:5
#1  0x000000000040056f in f2 (a=DO NOT WORRY) at q1.c:15
#2  0x00000000004005ab in main (argc=1, argv=0x7fffffffe2c8) at q1.c:23
```

---

Continue with the following GDB commands

```
(gdb) c
(gdb) bt
```

What is the ouput? Answer B.

```
#0  f1 (a=DO NOT WORRY) at q1.c:5
#1  0x000000000040054b in f1 (a=DO NOT WORRY) at q1.c:ANSWER B
#2  0x000000000040056f in f2 (a=DO NOT WORRY) at q1.c:15
#3  0x00000000004005ab in main (argc=1, argv=0x7fffffffe2c8) at q1.c:23
```

---

Continue with the following GDB commands

```
(gdb) b 16
(gdb) c
(gdb) bt
```

How many frames are show? Answer C.

---

Continue with the following GDB commands

```
(gdb) c
(gdb) c
(gdb) f 3
(gdb) p a
```

What is the ouput? Answer D.
**Answer:**

```
A: 5
B: 9
C: 5
D: 4
```

## 2 Structure

Consider the following program. Write down the answers.

```
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    int x;
10   int y;
11   int z;
12   int t;
13 } Vector;
14
15 int main(int argc, char ** argv)
16 {
17   Vector varr[NUMVECTOR];
18   printf("sizeof(varr) = %ld\n", sizeof(varr));
19   // <--- Answer A: what is the output?
20
21   // initialize the attributes
22   int ind;
23   for (ind = 0; ind < NUMVECTOR; ind ++)
24     {
25       varr[ind].x = ind;
26       varr[ind].y = ind * 2;
27       varr[ind].z = ind * 3;
28       varr[ind].t = ind * 4;
29     }
30
31   int * ptr = & varr[2].y;
32   int n = ptr[4];
33   printf("n = %d\n", n); // <--- Answer B: value of n?
34
35   for (ind = 0; ind < NUMVECTOR; ind ++)
36     {
37       ptr[ind] = ind;
38     }
39   int m = varr[3].x;
```

```
40    printf("m = %d\n", m); // <--- Answer C: value of m?
41
42    Vector * vptr = varr;
43    ptr = & (vptr -> x);
44    int r = ptr[2];
45    printf("r = %d\n", r); // <--- Answer D: value of r?
46
47    return EXIT_SUCCESS;
48 }
```

**Answer:**

A 160
B 6
C 3
D 0

# 3 File

Consider the following program. Assume the program returns `EXIT_SUCCESS`. What are the outputs?

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 // assume sizeof(char) = 1, sizeof(int) = 4,
 4 // sizeof(double) = 8, sizeof(a pointer) = 8
 5 #define NUMELEM 100
 6 int main(int argc, char * * argv)
 7 {
 8   // read from a file
 9   // argv[1]: name of the file
10   if (argc < 2)
11     {
12       return EXIT_FAILURE;
13     }
14
15   FILE * fptr = fopen(argv[1], "w"); // open for writing
16
17   if (fptr == NULL)
18     {
19       return EXIT_FAILURE;
20     }
21
22   // create an array of 100 integers
23   int arr[NUMELEM];
24   // initialize the array
25   int ind;
26   for (ind = 0; ind < NUMELEM; ind ++)
27     {
28       arr[ind] = ind;
29     }
30
31   // use fwrite to write the entire array  to file
32   if (fwrite(arr, sizeof(int), NUMELEM, fptr) != NUMELEM)
33     {
34       printf("fwrite fail\n");
35       fclose (fptr);
36       return EXIT_FAILURE;
37     }
38
```

```
39   fclose (fptr);
40   fptr = fopen(argv[1], "r"); // open for reading
41
42   // read some data
43   if (fread(arr, sizeof(int), NUMELEM / 2, fptr) != (NUMELEM / 2))
44     {
45       printf("fread fail\n");
46       fclose (fptr);
47       return EXIT_FAILURE;
48     }
49   int rtv = fseek(fptr, sizeof(int), SEEK_CUR); // skip one int
50   if (rtv == -1)
51     {
52       printf("fseek fail\n");
53       fclose (fptr);
54       return EXIT_FAILURE;
55     }
56
57   printf("rtv = %d\n", rtv); // <---  answer A
58
59   int m;
60   if (fread(& m, sizeof(int), 1, fptr) != 1)
61     {
62       printf("fread fail\n");
63       fclose (fptr);
64       return EXIT_FAILURE;
65     }
66   printf("m = %d\n", m); // <---  answer B
67   int count = 0;
68   int ch;
69
70   // count how many byets are left in the file
71   while (! feof(fptr))
72     {
73       ch = fgetc(fptr);
74       if (ch != EOF)
75         {
76           count ++;
77         }
78     }
79
80   printf("ch = %d\n", ch);  // <---  answer C
```

```
81
82    printf("count = %d\n", count); // <---  answer D
83    fclose(fptr);
84    return EXIT_SUCCESS;
85 }
```

The following information may be useful.

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE * stream );

size_t fwrite(const void *ptr, size_t size, size_t nmemb,
FILE *stream);

Description

The function fread() reads nmemb elements of data, each size bytes
long, from the stream pointed to by stream, storing them at the
location given by ptr.

The function fwrite() writes nmemb elements of data, each size bytes
long, to the stream pointed to by stream, obtaining them from the
location given by ptr.

Return Value

On success, fread() and fwrite() return the number of items read or
written. This number equals the number of bytes transferred only when
size is 1. If an error occurs, or the end of the file is reached, the
return value is a short item count (or zero).

fread() does not distinguish between end-of-file and error, and
callers must use feof(3) and ferror(3) to determine which occurred.

------------------------------

int 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.

------------------------------

int fseek(FILE *stream, long offset, int whence);

The fseek() function sets the file position indicator for the stream
pointed to by stream.  The new position, measured in bytes, is
```

```
obtained by adding offset bytes to the position specified by whence.
If whence is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is
relative to the start of the file, the current position indicator, or
end-of-file, respectively.

Upon successful completion, fseek() return 0.  Otherwise, -1 is
returned.

------------------------------

#define EXIT_FAILURE 1 /* Failing exit status.  */
#define EXIT_SUCCESS 0 /* Successful exit status.  */
#define EOF (-1)
```

*Answer:*
rtv = 0 m = 51 ch = -1 count = 192

# 4 Comparison Function for `qsort`

Consider this program. The comparison function is wrong. Answer the questions.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #define NUMELEM 16
 4 void printArray(int * arr, int size)
 5 {
 6   int ind;
 7   for (ind = 0; ind < size; ind ++)
 8     {
 9       printf("%d ", arr[ind]);
10     }
11   printf("\n\n");
12 }
13
14 /* This is a correct comparison function
15    for your reference. It is not used
16 int comparefunc(const void * arg1, const void * arg2)
17 {
18   const int * ptr1 = (const int *) arg1;
19   const int * ptr2 = (const int *) arg2;
20   int val1 = * ptr1;
21   int val2 = * ptr2;
22   if (val1 < val2)  { return -1; }
23   if (val1 == val2) { return 0; }
24   return 1;
25 } */
26
27 // The program uses a wrong comparison function
28 int comparefunc(const void * arg1, const void * arg2)
29 {
30   const int * ptr1 = (const int *) arg1;
31   const int * ptr2 = (const int *) arg2;
32   // <--- add code ---> Question D
33   if (ptr1 < ptr2)  { return -1; }
34   if (ptr1 == ptr2) { return 0; }
35   return 1;
36 }
37
38 int main(int argc, char * * argv)
39 {
```

```
40    srand(0);
41    int ind;
42    int arr[NUMELEM];
43    for (ind = 0; ind < NUMELEM; ind ++)
44      {
45        arr[ind] = rand() % 100;
46      }
47    printArray(arr, NUMELEM);
48    /* output is
49       83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 */
50    // use qsort to sort the array arr
51    // <--- FIX ME ---> next line
52    qsort(ANSWER A, NUMELEM, sizeof(int), comparefunc);
53    printArray(& arr[2], 2); // <--- Outputs as Answer B, C --->
54    return EXIT_SUCCESS;
55  }
```

Answer D: In Line 32, which of the following has compilation (i.e., syntax) error? **Ignore** the warning of *unused variable*. Choose **all** correct answers.

A. ptr1 = ptr2;

B. int n = * ptr1;

C. const int n = * ptr1;

D. * ptr1 = * ptr2;

E. None of the above (no syntax error)

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));

The qsort() function sorts an array with nmemb elements of size
size. The base argument points to the start of the array.

The contents of the array are sorted in ascending order according to a
comparison function pointed to by compar, which is called with two
arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or
greater than zero if the first argument is considered to be
respectively less than, equal to, or greater than the second. If two
members compare as equal, their order in the sorted array is
undefined.
```

**Answer:**

```
arr
77 15
D
```

# 5 Makefile

Consider `Makefile` from HW01.

```
WARNING = -Wall -Wshadow --pedantic
ERROR = -Wvla -Werror
COVERAGE = -fprofile-arcs -ftest-coverage
GCC = gcc -std=c99 -g $(WARNING) $(ERROR) $(COVERAGE)
TESTFALGS = -DTEST_ADD -DTEST_SUB -DTEST_MUL -DTEST_DIV -DTEST_MAIN
# list all .c files
SRCS = add.c div.c mul.c  main.c solution.c sub.c
# object files are created by converting .c to .o
# OBJS should be add.o div.o mul.o  main.o solution.o sub.o
OBJS = Answer A # <--- FIX ME --->

main: $(OBJS)
        $(GCC) $(TESTFALGS) $(OBJS) -o main

# convert .c to .o
Answer B  # <--- FIX ME --->
        $(GCC) $(TESTFALGS) Answer C  $*.c # <--- FIX ME --->

testall: testadd testsub testmul testdiv

testadd: main
        ./main 4 5 A > add1.out
        diff add1.out add1.correct
        ./main -2 17 A > add2.out
        diff add2.out add2.correct

testsub: main
        ./main 4 5 S > sub1.out
        diff sub1.out sub1.correct
        ./main 7 26 S > sub2.out
        diff sub2.out sub2.correct

testmul: main
        ./main 4 5 M > mul1.out
        diff mul1.out mul1.correct
        ./main 7 26 M > mul2.out
        diff mul2.out mul2.correct

testdiv: main
```

```
40          ./main 240 5 D > div1.out
41          diff div1.out div1.correct
42          ./main 70 26 D > div2.out
43          diff div2.out div2.correct
44
45  clean: # remove all machine generated files
46          rm -f main *.o *.out *gcda *gcno *gcov
47
48  # DO NOT DELETE THIS LINE -- make depend depends on it.
49
50  # determine the dependence of headers
51
52  depend:
53          makedepend $(SRCS)
```

### 5.1 Convert `.c` to `.o`

How can line 10 (Answer A) convert the list of `.c` files to the list of `.o` files? Your answer **must** include the symbol `SRC` and you must **not** list all `.o` files. Specifically, you will receive no point if your answer is
`OBJS = add.o div.o mul.o main.o solution.o sub.o`

### 5.2 Compile

What is the answer for B? (line 16)
What is the answer for C? (line 17) This line uses `gcc` to compile a `.c` file and generate a `.o` file.

### 5.3 Test

In `testall`, if `add1.out` and `add1.correct` are different, what does `make` do?

A. It stops `testadd` and starts `testsub`.

B. It stops completely.

C. It continues to execute `./main -2 17 A > add2.out`.

D. It rebuilds `main`.

E. None of the above.

**Answer:**

```
$(SRCS:%.c=%.o)
.c.o:
-c
B
```