

Recitation 04

I/O & Dynamic Allocation

George Mappouras

9/29/2017

I/O files & User interactions

- What if we want to create a calculator?

User must provides inputs

Program and user interaction

- “argc” the number of inputs (arguments)
- “argv[]” an array of pointers to the arguments from 1 to argc-1

Example – Calculator with 3 inputs

- User provides + or – and two numbers

```
int main(int argc, char **argv){
    if ( argc != 4 ) {
        printf("Usage: ./prog <oper> <num1> <num2> \n");
        return EXIT_FAILURE;
    }
    else
        ...
}
```

Example – Calculator with 3 inputs

- User provides + or – and two numbers

```
int main(int argc, char **argv){  
    if ( argc != 4 ) {  
        printf("Usage: ./prog <oper> <num1> <num2> \n");  
        return EXIT_FAILURE;  
    }  
    else  
        ...  
}
```

Argv and Argc

Argc: A box storing the number of arguments in argv (i.e. 4)

Argv: A pointer that points to an array (A) of pointers.

The pointers of array A point to our input arguments

Open and close files

```
FILE *input;  
input = fopen("./file.txt, "w+");  
if (!input){  
    printf("Error opening file!\n");  
}  
int check = fclose(input);  
if (!check){  
    printf("Error closing file!\n");  
}
```

Open and close files

```
FILE *f;
```

```
f = fopen("./file.txt", "w+");
```

→ Different options for different requirements

```
if (!input){
```

```
    printf("Error opening file!\n");
```

```
}
```

```
if (fclose(input)){
```

```
    printf("Error closing file!\n");
```

```
}
```

Open and close files

```
FILE *f;
```

```
f = fopen("./file.txt", "w+");
```

→ Different options for different requirements

```
if (!input){
```

```
    printf("Error opening file!\n");
```

```
}
```

```
if (fclose(input)){
```

```
    printf("Error closing file!\n");
```

```
}
```

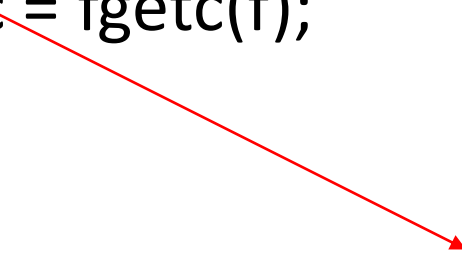
Page 219 on AOP

Reading files

```
int c = fgetc(f);
```

Reading files

```
int c = fgetc(f);
```



Why is this an int and not a char. What is important about it?

EOF

Special “mark” that at the end of a file (EOF)

How do I detect EOF

One way is `fgetc(input)`. Returns EOF at the end of file.

```
int x = fgetc(input)
```

```
if(x != EOF){
```

```
...
```

```
}
```

```
int x;
```

```
while (x = fgetc(input) != EOF){
```

```
...
```

```
}
```

Reading files (continued...)

```
int c = fgetc(f);
```

```
fgets(char * in, int size, FILE * f);
```

example:

```
char input[line_size];
```

```
fgets(input, line_size, f);
```

Reading files

```
int c = fgetc(f);
```

```
fgets(char * in, int size, FILE * f);
```

example:

```
char input[line_size];
```

```
fgets(input, line_size, f);
```

```
fread(void * p, size_t size_of_element, size_t num_elements, FILE * binfile);
```

example: `fread(input, sizeof(char), sizeof(line_size), f);`

Exercise

Assume we want to read and print the file text.txt:

ECE551

GEORGE

TA

```
int readFunc(){
    FILE * f = fopen("./file.txt" ,"r");
    if (!f){
        printf("Error opening file!\n");
        return EXIT_FAILURE;
    }
    int c;
    while (1){
        c = fgetc(f);
        if (c == EOF) break;
        else printf("%c",c);

    }
    fclose(f);
    return 0;
}
```

```
int readFunc(){
    FILE * f = fopen("./file.txt" ,"r");
    if (!f){
        printf("Error opening file!\n");
        return EXIT_FAILURE;
    }
    int c;
    while (1){
        c = fgetc(f);
        if (c == EOF) break;
        else printf("%c",c);

    }
    fclose(f);
    return 0;
}
```

```
int readFunc(){
    FILE * f = fopen("./file.txt" ,"r");
    if (!f){
        printf("Error opening file!\n");
        return EXIT_FAILURE;
    }
    char c[8];
    int i;
    for (i=0;i<3;i++){
        fgets(c,sizeof(c),f);
        printf("%s",c);
    }
    fclose(f);
    return 0;
}
```



```
int readFunc(){
    FILE * f = fopen("./file.txt" ,"r");
    if (!f){
        printf("Error opening file!\n");
        return EXIT_FAILURE;
    }
    int c;
    while (1){
        c = fgetc(f);
        if (c == EOF) break;
        else printf("%c",c);

    }
    fclose(f);
    return 0;
}
```

```
int readFunc(){
    FILE * f = fopen("./file.txt" ,"r");
    if (!f){
        printf("Error opening file!\n");
        return EXIT_FAILURE;
    }
    char c[8];
    int i;
    for (i=0;i<3;i++){
        fgets(c,sizeof(c),f);
        printf("%s",c);
    }
    fclose(f);
    return 0;
}
```

```
int readFunc(){
    FILE * f = fopen("./file.txt" ,"r");
    if (!f){
        printf("Error opening file!\n");
        return EXIT_FAILURE;
    }
    char c[18];
    fread(c,sizeof(char),16,f);
    c[16] = '\n';
    c[17] = '\0';
    printf("%s",c);
    fclose(f);
    return 0;
}
}
```

Writing files

`fwrite(const void * p, size_t size, size_t elements, FILE * binfile);`

example: `fwrite(out, 1, sizeof(out)-1, f);`

`fprintf(FILE * f, const char * string, ...);`

example: `fprintf(f, "Write that to the file\n");`

Writing files - Example

```
FILE * input = fopen("./file.txt", "w+");
if (!input){
    printf("Error opening file!\n");
    return EXIT_FAILURE;
}
char a = "GEORGE\n";
int i;
for(i=0; i<7; i++){
    fwrite(&a[i], sizeof(char),1,input);
}
fclose(input);
```

Malloc

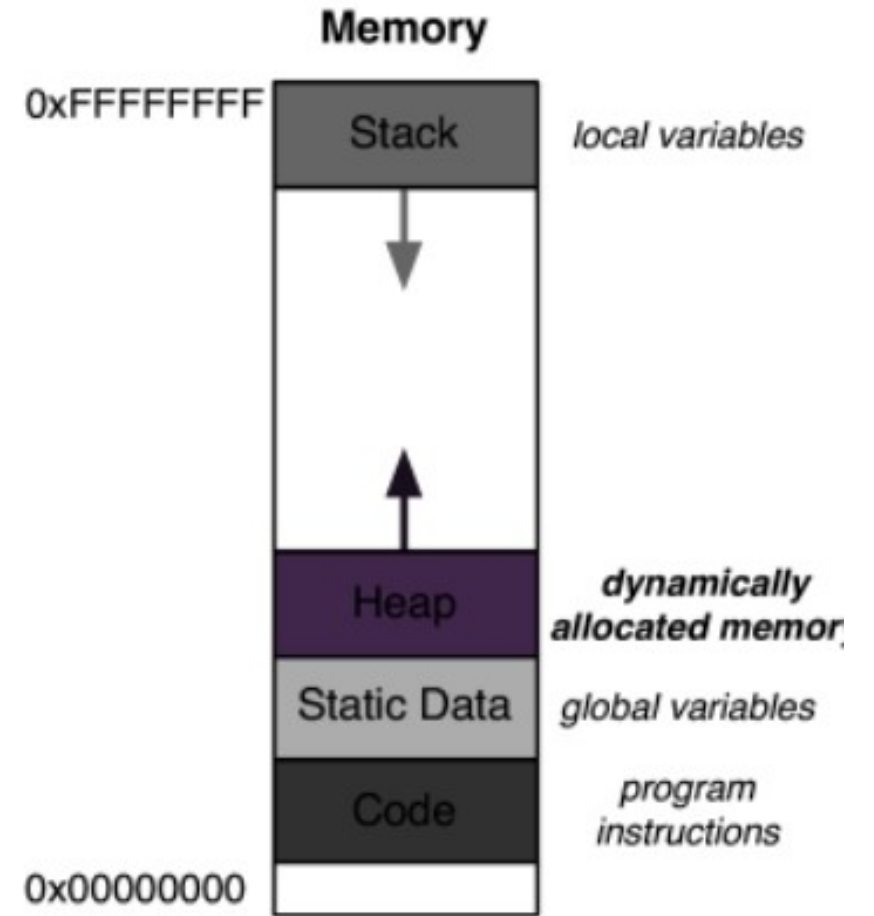
```
int * p;
```

p → ?

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

p →

--	--	--	--	--



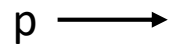
Free

Now we have:



Assume we do not need the array anymore and we want to free that memory

`free(p);`



Note that the pointer `p` is not deleted!

What not to Free!

Assume we have:

```
int * p1, p3;
```

```
int * p2 = NULL;
```

```
int array[5] = {0};
```

```
p1 = array;
```

```
p3 = malloc(5*sizeof(int));
```

Is it ok to do the follow:

1) free(p1);

2) free(p3[2]);

3) for(i=0;i<2;i++) free(array);

4) free(p2)

5) free (p1); free(p1)

What not to Free!

Assume we have:

```
int * p1, p3;
```

```
int * p2 = NULL;
```

```
int array[5] = {0};
```

```
p1 = array;
```

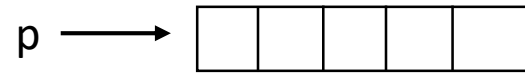
```
p3 = malloc(5*sizeof(int));
```

Is it ok to do the follow:

- | | |
|----------------------------------|-----|
| 1) free(p1); | NO |
| 2) free(p3[2]); | NO |
| 3) for(i=0;i<2;i++) free(array); | NO |
| 4) free(p2) | YES |
| 5) free (p1); free(p1) | NO |

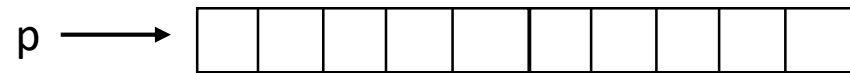
Realloc

From before we had:



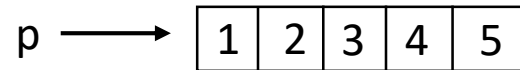
What if we decided we need more space?

```
p = realloc(p, 10*sizeof(*p));
```



Realloc

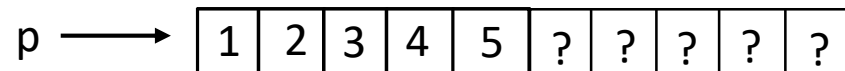
From before we had:



&p[0] == a

What if we decided we need more space?

```
p = realloc(p, 10*sizeof(*p));
```



&p[0] == b

address "a" and address "b" are different

What are the steps realloc has to follow?

Writing our own Realloc

Let's assume we have a `getsize()` function that returns the size of the allocated space of a pointer

```
size_t getsize(void * p);
```

Writing our own Realloc

```
void * myRealloc(void * ptr, size_t new_size){
    void * new_ptr = malloc(new_size);
    if (ptr == NULL) return new_ptr;
    size_t size = getsize(ptr);
    if (size >= new_size){
        for(int i=0;i<new_size;i++) new_ptr[i] = ptr[i];
    }
    else{
        for(int i=0; i<size;i++) new_ptr[i] = ptr[i];
    }
    free(ptr);
    return new_ptr;
}
```

Simple Exercise 1

Assume that an int is 4 bytes, a char 1 byte and a pointer 8 bytes.

Which of the follow will allocate memory space for 4 integers?

```
int * p;
```

1. `p = malloc(4*sizeof(p));`
2. `p = malloc(4*sizeof(int));`
3. `p = malloc(4*sizeof(char));`
4. `p = malloc(16*sizeof(char));`

Simple Exercise 1

Assume that an int is 4 bytes, a char 1 byte and a pointer 8 bytes.

Which of the follow will allocate memory space for 4 integers?

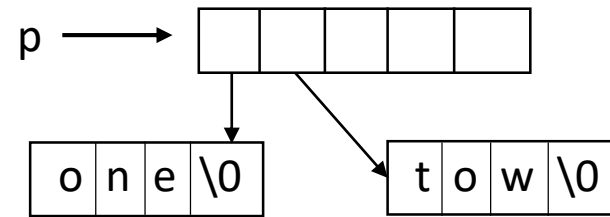
```
int * p;
```

- | | |
|--|-------|
| 1. <code>p = malloc(4*sizeof(p));</code> | FALSE |
| 2. <code>p = malloc(4*sizeof(int));</code> | TRUE |
| 3. <code>p = malloc(4*sizeof(char));</code> | FALSE |
| 4. <code>p = malloc(16*sizeof(char));</code> | TRUE |

Simple Exercise 2 – Memory leak

Assume that we have allocated the memory:

How should we free it?



1. `free(p);`

2. `free(p);`
`free(p[0]);`
`free(p[1]);`

3. `free(p[1]);`
`free(p[0]);`
`free(p);`

Malloc and read at the same time

```
ssize_t getline(char **result, size_t * size, FILE * file);
```

example:

```
char * newline=NULL; size_t size = 0;
```

```
FILE * f = fopen("./myfile.txt","r");
```

```
getline(&newline, &size, f);
```

getline does the following 4 tasks:

1. Allocate memory
2. update size
3. copy the line to the allocated memory
4. return length of line

Question 4 – midterm 2 years ago

write your own

```
char * myStrchr(const char *s, int c);
```

Takes a string and returns a pointer to the first location with the character c. If c is not found it returns NULL;

Question 4 in last years midterm

```
char * myStrchr(const char *s, int c){  
    int i = 0;  
    char * ptr = NULL;  
    while (s[i] != (char) c){  
        if (s[i] == '\0') return ptr;  
        i++;  
    }  
    ptr = (char *) (&s[i]);  
    return ptr;  
}
```

Question 4 in last years midterm

```
char * myStrchr(const char *s, int c){
    int i = 0;
    char * ptr = NULL;
    while (s[i] != (char) c){
        if (s[i] == '\0') return ptr;
        i++;
    }
    ptr = (char *) (&s[i]);
    return ptr;
}
```

More elegant solution:

```
char *strchr(const char *s, int c)
{
    const char ch = c;

    for ( ; *s != ch; s++)
        if (*s == '\0')
            return 0;
    return (char *)s;
}
```

Question 5

```
struct _vect_t {  
    double * values;  
    size_t n;  
};  
typedef struct _vect_t vect_t;  
  
double dotProduct(vect_t a, vect_t b) {
```

Question 5

```
double dotProduct(vect_t a, vect_t b) {  
    assert(a.n == b.n);  
    double result = 0;  
    for (int i=0; i<n; i++){  
        result = result + a.values[i]*b.values[i];  
    }  
    return result;  
}
```

Example

$\{a_1, a_2, a_3\} \cdot \{b_1, b_2, b_3\} =$
 $a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3$

Question 6 is realloc

We already did it -> slide 18

Question 7

Take two inputs file_in and file_out. Read the lines of file_in and write them in reverse order in file_out

file_in

My name

is

George

file_out

George

is

My name

Question 7

```
int main(int argc, char **argv){
    FILE * fin = fopen(argv[1],"r+");
    FILE * fout = fopen(argv[2],"w+");
    char * in = NULL;
    char ** p = NULL;
    size_t s = 0;
    int count = 0, i, j;
    while (getline(&in,&s,fin) >= 0){
        p = realloc(p,(count+1)*sizeof(*p));
        p[count] = in;
        in = NULL;
        count++;
    }
```

```
for (i=count-1;i>=0;i-- ){
    j=0;
    while(p[i][j]!='\0'){
        fwrite(&(p[i][j]),sizeof(char),1,fout);
        j++;
    }
    fclose(fin);
    fclose(fout);
    return 0;
}
```


Question 7

What I am missing here?

```
int main(int argc, char **argv){
    FILE * fin = fopen(argv[1],"r+");
    FILE * fout = fopen(argv[2],"w+");
    char * in = NULL;
    char ** p = NULL;
    size_t s = 0;
    int count = 0, i, j;
    while (getline(&in,&s,fin) >= 0){
        p = realloc(p,(count+1)*sizeof(*p));
        p[count] = in;
        in = NULL;
        count++;
    }
```

```
for (i=count-1;i>=0;i-- ){
    j=0;
    while(p[i][j]!='\0'){
        fwrite(&(p[i][j]),sizeof(char),1,fout);
        j++;
    }
    fclose(fin);
    fclose(fout);
    return 0;
}
```

Question 6 Coding 3: Dynamic Allocation [13 pts]

Write the function `char * int2Str(unsigned int x)` which takes an `unsigned int`, allocates memory for a string, and fills it in with the digits of the decimal representation of `x`. For example, if your function is passed "123", then it should return 123. Note that `x` is unsigned, so you do not have to deal with negative numbers. You should not make assumptions about the maximum size of a number that is representable in an `unsigned int` (*e.g.*, if I run your code on a weird platform where `unsigned int` is 1024 bits, it should still work right).

You may use any of the following library functions (but only the following library functions): `malloc`, `realloc`, and `free`..

```
char * int2Str(unsigned int x) {
```

```
char * int2Str(unsigned int x){  
    //unsigned int rem = x;  
    char * p = NULL;  
    char tmp;  
    int counter = 0;  
    while ((x/10) != 0){  
        p = realloc(p,(counter+1)*sizeof(char));  
        p[counter] = (x % 10)+'0';  
        x = x/10;  
        counter++;  
    }  
}
```

```
    p = realloc(p,(counter+2)*sizeof(char));  
    p[counter] = (x % 10)+'0';  
    for(int i=0;i<counter/2;i++){  
        tmp = p[i];  
        p[i] = p[counter-i];  
        p[counter-i] = tmp;  
    }  
    p[counter+1] = '\0';  
    return p;  
}
```

Find the memory leaks

- Question 12.8 : Consider the following code (which has memory leaks):

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int f(int n) {
5     int * p = malloc(2 * sizeof(*p));
6     p[0] = n;
7     p[1] = n+2;
8     int ans = p[0] * p[1];
9     return ans;
10 }
11
12 int main(void) {
13     int * p = malloc(4 * sizeof(*p));
14     int * q = p;
15     int ** r = &q;
16     p[0] = f(1);
17     *r = NULL;
18     q = malloc(2 * sizeof(*q));
19     p = q;
20     q = NULL;
21     return EXIT_SUCCESS;
22 }
```

Find the memory leaks

- Question 12.8 : Consider the following code (which has memory leaks):

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int f(int n) {
5     int * p = malloc(2 * sizeof(*p));
6     p[0] = n;
7     p[1] = n+2;
8     int ans = p[0] * p[1];
9     return ans;
10 }
11
12 int main(void) {
13     int * p = malloc(4 * sizeof(*p));
14     int * q = p;
15     int ** r = &q;
16     p[0] = f(1);
17     *r = NULL;
18     q = malloc(2 * sizeof(*q));
19     p = q;
20     q = NULL;
21     return EXIT_SUCCESS;
22 }
```

When function f returns, no pointer is pointing to the memory that was allocated in line 5

No pointer is pointing at the initial memory space allocated in line 13

Of course these memory leaks in this naïve program do not matter that much because all the memory allocated by our program will be freed after our main returns and the program terminates

Question 3 Debugging [8 pts]

A C programmer wrote the following buggy code to read in many lines of numbers, sort them, and print them out:

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: int cmplong(const void * vp1, const void * vp2) {
4:     const long * p1 = vp1;
5:     const long * p2 = vp2;
6:     return *p1 - *p2;
7: }
8: int main(void) {
9:     char * line= NULL;
10:    size_t sz =0;
11:    long * array = NULL;
12:    size_t n = 0;
13:    while(getline(&line, &sz, stdin) > 0){
14:        n++;
15:        array=realloc(array, n * sizeof(*array));
16:        array[n] = strtol(line, NULL, 0);
17:    }
18:    free(line);
19:    qsort(&array, n, sizeof(*array), cmplong);
20:    for (size_t i = 0; i < n; i++) {
21:        printf("%ld\n", array[i]);
22:        free(&array[i]);
23:    }
24:    return EXIT_SUCCESS;
25: }
```

void qsort(void *base, size_t nitems, size_t size,
int (*compar)(const void *, const void*))

Error 1

```
Invalid write of size 8
  at 0x400788: main (broken.c:16)
Address 0x51fc108 is 0 bytes after a block of size 8 alloc'd
  at 0x4C2AB80: malloc (in ...)
  by 0x4C2CF1F: realloc (in ...)
  by 0x400759: main (broken.c:15)
```

This problem can be correctly fixed by:

- A. Using `malloc` instead of `realloc` on line 15
- B. Changing line 16 to have `array[n-1]` instead of `array[n]`
- C. Changing line 13 to read `size_t n = 1;`
- D. Moving line 14 after line 15.
- E. None of the above—specify what to do instead:

Question 3 Debugging [8 pts]

A C programmer wrote the following buggy code to read in many lines of numbers, sort them, and print them out:

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: int cmplong(const void * vp1, const void * vp2) {
4:     const long * p1 = vp1;
5:     const long * p2 = vp2;
6:     return *p1 - *p2;
7: }
8: int main(void) {
9:     char * line= NULL;
10:    size_t sz =0;
11:    long * array = NULL;
12:    size_t n = 0;
13:    while(getline(&line, &sz, stdin) > 0){
14:        n++;
15:        array=realloc(array, n * sizeof(*array));
16:        array[n] = strtol(line, NULL, 0);
17:    }
18:    free(line);
19:    qsort(&array, n, sizeof(*array), cmplong);
20:    for (size_t i = 0; i < n; i++) {
21:        printf("%ld\n", array[i]);
22:        free(&array[i]);
23:    }
24:    return EXIT_SUCCESS;
25: }
```

void qsort(void *base, size_t nitems, size_t size,
int (*compar)(const void *, const void*))

Error 1

```
Invalid write of size 8
  at 0x400788: main (broken.c:16)
Address 0x51fc108 is 0 bytes after a block of size 8 alloc'd
  at 0x4C2AB80: malloc (in ...)
  by 0x4C2CF1F: realloc (in ...)
  by 0x400759: main (broken.c:15)
```

This problem can be correctly fixed by:

- A. Using `malloc` instead of `realloc` on line 15
- ☒ B. Changing line 16 to have `array[n-1]` instead of `array[n]`
- C. Changing line 13 to read `size_t n = 1;`
- D. Moving line 14 after line 15.
- E. None of the above—specify what to do instead:

Error 2

```
Conditional jump or move depends on uninitialised value(s)  
by 0x4E726CB: qsort_r (...)  
by 0x4007CF: main (broken.c:19)
```

This problem can be correctly fixed by:

- A. Changing `qsort(&array,` to `qsort(array,` on line 19.
- B. Changing `qsort(&array,` to `qsort(*array,` on line 19.
- C. Changing `sizeof(*array)` to `sizeof(array)` on line 19.
- D. Changing `sizeof(*array)` to `sizeof(&array)` on line 19.
- E. None of the above—specify what to do instead:

Question 3 Debugging [8 pts]

A C programmer wrote the following buggy code to read in many lines of numbers, sort them, and print them out:

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: int cmplong(const void * vp1, const void * vp2) {
4:     const long * p1 = vp1;
5:     const long * p2 = vp2;
6:     return *p1 - *p2;
7: }
8: int main(void) {
9:     char * line= NULL;
10:    size_t sz =0;
11:    long * array = NULL;
12:    size_t n = 0;
13:    while(getline(&line, &sz, stdin) > 0){
14:        n++;
15:        array=realloc(array, n * sizeof(*array));
16:        array[n] = strtol(line, NULL, 0);
17:    }
18:    free(line);
19:    qsort(&array, n, sizeof(*array), cmplong);
20:    for (size_t i = 0; i < n; i++) {
21:        printf("%ld\n", array[i]);
22:        free(&array[i]);
23:    }
24:    return EXIT_SUCCESS;
25: }
```

void qsort(void *base, size_t nitems, size_t size,
int (*compar)(const void *, const void*))

Error 2

```
Conditional jump or move depends on uninitialised value(s)  
by 0x4E726CB: qsort_r (...)  
by 0x4007CF: main (broken.c:19)
```

This problem can be correctly fixed by:

- ☒ A. Changing `qsort(&array,` to `qsort(array,` on line 19.
- B. Changing `qsort(&array,` to `qsort(*array,` on line 19.
- C. Changing `sizeof(*array)` to `sizeof(array)` on line 19.
- D. Changing `sizeof(*array)` to `sizeof(&array)` on line 19.
- E. None of the above—specify what to do instead:

Error 3

```
Invalid read of size 8
  at 0x4007ED: main (broken.c:21)
Address 0x51fd408 is 8 bytes inside a block of size 224 free'd
  at 0x4C2BDEC: free (in ...)
  by 0x40081C: main (broken.c:22)
```

```
Invalid free() / delete / delete[] / realloc()
  at 0x4C2BDEC: free (in ...)
  by 0x40081C: main (broken.c:22)
Address 0x51fd408 is 8 bytes inside a block of size 224 free'd
  at 0x4C2BDEC: free (in ...)
  by 0x40081C: main (broken.c:22)
```

This problem can be correctly fixed by:

- A. Changing line 22 from `free(&array[i]);` to `free(array[i]);`
- B. Changing line 22 from `free(&array[i]);` to `free(array);`
- C. Changing line 22 from `free(&array[i]);` to `free(*array[i]);`
- D. Changing line 22 from `free(&array[i]);` to `free(array+i);`
- E. None of the above—specify what to do instead:

Question 3 Debugging [8 pts]

A C programmer wrote the following buggy code to read in many lines of numbers, sort them, and print them out:

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: int cmplong(const void * vp1, const void * vp2) {
4:     const long * p1 = vp1;
5:     const long * p2 = vp2;
6:     return *p1 - *p2;
7: }
8: int main(void) {
9:     char * line= NULL;
10:    size_t sz =0;
11:    long * array = NULL;
12:    size_t n = 0;
13:    while(getline(&line, &sz, stdin) > 0){
14:        n++;
15:        array=realloc(array, n * sizeof(*array));
16:        array[n] = strtol(line, NULL, 0);
17:    }
18:    free(line);
19:    qsort(&array, n, sizeof(*array), cmplong);
20:    for (size_t i = 0; i < n; i++) {
21:        printf("%ld\n", array[i]);
22:        free(&array[i]);
23:    }
24:    return EXIT_SUCCESS;
25: }
```

void qsort(void *base, size_t nitems, size_t size,
int (*compar)(const void *, const void*))

Error 3

```
Invalid read of size 8
  at 0x4007ED: main (broken.c:21)
Address 0x51fd408 is 8 bytes inside a block of size 224 free'd
  at 0x4C2BDEC: free (in ...)
  by 0x40081C: main (broken.c:22)
```

```
Invalid free() / delete / delete[] / realloc()
  at 0x4C2BDEC: free (in ...)
  by 0x40081C: main (broken.c:22)
Address 0x51fd408 is 8 bytes inside a block of size 224 free'd
  at 0x4C2BDEC: free (in ...)
  by 0x40081C: main (broken.c:22)
```

This problem can be correctly fixed by:

- A. Changing line 22 from `free(&array[i]);` to `free(array[i]);`
- B. Changing line 22 from `free(&array[i]);` to `free(array);`
- C. Changing line 22 from `free(&array[i]);` to `free(*array[i]);`
- D. Changing line 22 from `free(&array[i]);` to `free(array+i);`
- ☒ E. None of the above—specify what to do instead: remove line 22

Error 4

```
224 bytes in 1 blocks are definitely lost in loss record 1 of 1
   at 0x4C2CE8E: realloc (...)
   by 0x400759: main (broken.c:15)
```

This error can be corrected by:

- A. Placing `free(array[i]);` between lines 22 and 23 (inside the for loop)
- B. Placing `free(array)` between lines 23 and 24 (after the for loop)
- C. Placing `free(line);` between lines 16 and 17 (inside the while loop)
- D. Changing `sizeof(*array)` to `\verbsizeof(array)+` on line 19
- E. None of the above—specify what to do instead:

Question 3 Debugging [8 pts]

A C programmer wrote the following buggy code to read in many lines of numbers, sort them, and print them out:

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: int cmplong(const void * vp1, const void * vp2) {
4:     const long * p1 = vp1;
5:     const long * p2 = vp2;
6:     return *p1 - *p2;
7: }
8: int main(void) {
9:     char * line= NULL;
10:    size_t sz =0;
11:    long * array = NULL;
12:    size_t n = 0;
13:    while(getline(&line, &sz, stdin) > 0){
14:        n++;
15:        array=realloc(array, n * sizeof(*array));
16:        array[n] = strtol(line, NULL, 0);
17:    }
18:    free(line);
19:    qsort(&array, n, sizeof(*array), cmplong);
20:    for (size_t i = 0; i < n; i++) {
21:        printf("%ld\n", array[i]);
22:        free(&array[i]);
23:    }
24:    return EXIT_SUCCESS;
25: }
```

void qsort(void *base, size_t nitems, size_t size,
int (*compar)(const void *, const void*))

Error 4

```
224 bytes in 1 blocks are definitely lost in loss record 1 of 1
   at 0x4C2CE8E: realloc (...)
   by 0x400759: main (broken.c:15)
```

This error can be corrected by:

- A. Placing `free(array[i]);` between lines 22 and 23 (inside the for loop)
- ☒ B. Placing `free(array)` between lines 23 and 24 (after the for loop)
- C. Placing `free(line);` between lines 16 and 17 (inside the while loop)
- D. Changing `sizeof(*array)` to `\verbsizeof(array)+` on line 19
- E. None of the above—specify what to do instead:

How to do well in your midterm

Prepare for the exam:

Practice on previous midterm exams first without solutions

Look at solutions and verify your results

Gather questions and go to TA hours

Train until you feel confident -> Time your self.

During the exam:

Leave nothing blank!

Give an answer even if it is incomplete or wrong

Leave the things you don't know how to do for the end

Always read your answers once more before leaving. Make sure you did not leave any questions behind.