Test Plan

Running intrusions:

Typing (make) or (make test) into the terminal will create test file which will test testcase.c and mymalloc.c. (./test) can be used to test. In order to test memgrind and mymalloc you must do (make memgrind). Then use (./memgrind).

My program is deemed correct if the user is able to request space, use that space and be able to free that space accordingly. When the user calls malloc multiple times for data, the data is received, and pervious data is never to be overwritten. The same can be said for free. When free is called, the user data should never be overwritten/lost. And the data that was meant to be freed should be reset and be able to be used again in future malloc calls.

First, I would check for any overwritten data by the malloc function. Due to the fact that malloc is easier to test and needs to be fully functional before doing free, it makes sense to test this first. I would do a continuous amount of malloc calls varying in size and types to see if any of the data was overwritten. Of course, I would be adding things to these malloc called variables to make sure that works as well. I would print the memory array as these calls are happening and manually count the bytes of the array to make sure all data is there that is meant to be there, and no data is lost that shouldn't be. Our malloc creates a Node at the end of the memory array that. Will keep track of the remaining bytes of memory we have. This last Node will continue to be remade after every malloc call, the last Node will continuously be updated to allow for no data to be lost in the future. In Malloc when we go through the memory and find a free node that fits the size of what's being asked, we first see if we can make an entirely new node within that space before just giving away the space. This will help fix many errors, for example is I can for 120 bytes, I free it and then ask for 1 byte. I would not be giving the entire 120 bytes but instead a new node will be created within that free space to help with space management.

Now I will test the free function. Calling free should remove all data that the user wanted to free. And free should also set the node to being free and able to be used again future malloc calls. After a few successful frees, I would attempt to both free and malloc to make sure that no data is being overwritten/lost. Once these tests are finished I would start doing edge cases. For example, requesting more memory than available, free an invalid pointer, free something not created by malloc, free a pointer that was already freed, etc. After all of these tests my code would be fully tested and proven to work as the project was intended. Also the free function does not allow for any pointers that are Null. Negative number not in scope, not tested against.

Description of Test Cases:

```
Int x;
free(&x);
```

This test cases will cover the portion of the code that will reject any address that is not

valid/created by malloc.

```
Int *p = malloc(sizeof(int)*100);
free(p+1);
```

The following test will lead to a not valid address. The address is not the start of the data meaning free will not be able to properly access the MetaData.

```
Int *p = malloc(sizeof(int)*100);
Int *q = p;
free(p);
free(q);
```

The address is already freed. Meaning that the MetaData and user data for that address has already been removed from the memory and shifted in order to get more space. This should not be allowed.

```
Char *p = malloc(10);
Char *q = malloc(5);
free(q);
Char *bestTA = malloc(10);
```

After the two malloc calls and the free calls the third malloc call should not overwrite any data. This test freeing from the middle of the malloc array and requesting more memory without any errors.

```
Char *p = malloc(4097);
```

This should not be allowed due to the MEMSIZE being equal to 4096. Overall any call to malloc over the remaining memory will result in a failure.

```
  Char *p = malloc(120);
free(p);
  Char *p = malloc(1);
```

The code should not return the entire 120 bytes to the user but instead make a new Node in order to do better space management.

Design Notes

The MetaData is worth 9 bytes that store the size of the user data and whether the node is free or not. There's a freeNode that is created at the end I order to help keep malloc easier to maintain.

The free function is far simpler than malloc, with all it doing is freeing the user data and setting the Node too free. When running testCase.c it would make it more clear if the other four test cases are commented out. Makes it easier to see. “Exit failure” was not added because it stopped the program from running correctly. Memgrind is were we do part 2 and do the basic testing. Testcase.c is where the stress cases are. Removed address sanitizer because when testing for an invalid pointer it would point out the error instead of our code giving the error message.

Thank you for reading happy grading!