91.102 : Exercise 4

Read

- Section 1.4
- Section 1.5
- Section 1.6
- Section 1.7
- Section 1.8

in Esakov and Weiss and do the following tasks.

1 Preparation

1. Make a new directory, called ex4, for this exercise. Always make a new directory for every exercise as you may need previous work later in the course.
2. Make sure you are using the most current version of the Makefile and globals.h.

2 Header Files

There are some things that you must make a habit of doing for every header file. Therefore, these things are not included in the instructions for each header file but are simply mentioned here.

1. Include the necessary pre-processor directive to prevent multiple inclusions.

3 All Files

There are some things that you must make a habit of doing for all files. Those things are not included in the instructions for each file, but are simple mentioned here.

1. Include the necessary header files so that your file can compile on its own. You must type "make headers" at the prompt to check for this property with header files.

2. Always use angle brackets when including system headers. Always use double quotes when including your personal header files.

3. Always include system headers before personal header files. Always alphabetize your list of system includes and your list of personal includes.

4 Primitive and Application

Note that array.h and array.c constitute the interface and implementation of the primitive while main.c is the application.

1. Put the following procedure declarations in a file called array.h.

```
status init_array(generic_ptr * * a,
                  size_t const size);

status set(generic_ptr a[],
           size_t const size,
           size_t const index,
           generic_ptr const val);


status get(generic_ptr a[],
           size_t const size,
           size_t const index,
           generic_ptr * const val);

void destroy_array(generic_ptr * * a,
                   size_t const size,
                   void (*p_func_f)());
```

2. Put the following procedure implementation in a file called array.c.

```
status init_array(generic_ptr * * a,
                  size_t const size)
{
  generic_ptr * temp =
    (generic_ptr*) calloc(size, sizeof(generic_ptr));
  if (temp == NULL) {
    return ERROR;
  }
  *a = temp;
  for (size_t i = 0; i != size; ++i) {
    (*a)[i] = NULL;
  }
  return OK;
}
```

   Make sure this compiles before you move on.

3. Start a file called main.c to exercise the code that you have placed in array.c. Place the following main procedure in main.c and make sure that you can compile and execute main. Make sure that main returns 0, indicating success.

```
int main(int argc, char * * argv)
{
  size_t const a_size = 3;
  generic_ptr * a;
  if (init_array(&a, a_size) == ERROR) {
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

4. Add the implementation of the set procedure to array.c like this.

```
status set(generic_ptr a[],
           size_t const size,
           size_t const index,
           generic_ptr const val)
{
  if (index >= size) {
    return ERROR;
  }
  a[index] = val;
  return OK;
}
```

5. Exercise the set procedure by adding the following lines of code to the main procedure. Make sure the code compiles, that you can run the executable, main, and that main returns 0.

```
if (set(a, a_size, 3, NULL) != ERROR) {
  return EXIT_FAILURE;
}
if (set(a, a_size, 2, NULL) == ERROR) {
  return EXIT_FAILURE;
}
```

6. Add the implementation of the get procedure to array.c like this.

```
status get(generic_ptr a[],
           size_t const size,
           size_t const index,
           generic_ptr * const val)
{
  if (index >= size) {
    return ERROR;
  }
  *val = a[index];
  return OK;
}
```

7. Exercise the get procedure by adding the following lines of code to the main procedure. Make sure the code compiles, that you can run the executable, main, and that main returns 0.

```
{
  generic_ptr p;
  if (get(a, a_size, 3, &p) != ERROR) {
    return EXIT_FAILURE;
  }
  if (get(a, a_size, 2, &p) == ERROR) {
    return EXIT_FAILURE;
  }
}
```

8. Add the implementation of the destroy_array procedure to array.c like this.

```
void destroy_array(generic_ptr * * a,
                   size_t const size,
                   void (*p_func_f)())
{
  for (size_t i = 0; i != size; ++i) {
    (*p_func_f)(&(*a)[i]);
    (*a)[i] = NULL;
  }
  free(*a);
  *a = NULL;
}
```

9. Exercise the destroy_array procedure by adding the following lines of code to main.c.

```
void destroy_generic_ptr(generic_ptr * p)
{
  free(*p);
  *p = NULL;
}
```

   Make sure the code compiles, that you can run the executable, main, and that main returns 0.

10. Then add the following lines of code to the main procedure.

```
destroy_array(&a, a_size, destroy_generic_ptr);

if (a != NULL) {
  return EXIT_FAILURE;
}
```

    Make sure the code compiles, that you can run the executable, main, and that main returns 0.

5 Final Steps

1. Type "make cppcheck" and fix any errors. Note that cppcheck may not be installed on your personal computer. If it is not, you will have to do the final step on the CS computers.

2. You are now done. Hand in using Bottlenose and fix any errors that Bottlenose exhibits.