Module: R1: C Programming
Section: Assessments **Task:** Memory Test

Task 6.1
Final Assessment

---

1. **Main Function:**
   - **Code Snippet:**

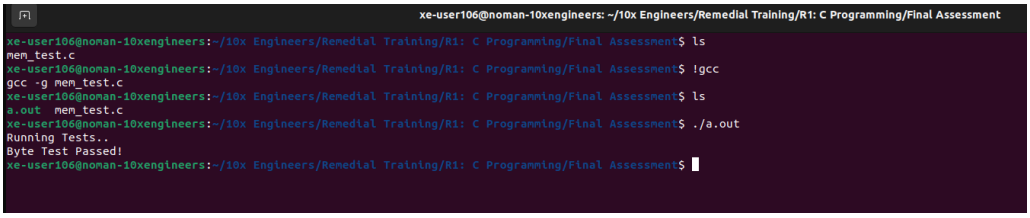
```
int main() {

      printf("Running Tests..\n");
      store_byte_test(store_byte_data);
      store_half_word_test(store_half_word_data);
      store_word_test(store_word_data);
      store_double_word_test(store_double_word_data);
}
```

2. **Store Byte Test:**
   - **Code Snippet:**

```
int store_byte_test (arr_t *p) {

// EF, BE, AD, DE, EF, BE, AD, DE
      unsigned long long store_byte[] = { 0xEF, 0xBE, 0xAD, 0xDE, 0xEF,
0xBE, 0xAD, 0xDE };
      int b = 0;

      //Setting bits
      for (int i = 0; i < 8; i++){
      b = b + 2;
            store_byte [i+1] = store_byte [i+1] << (b*4);
      }

      //storing data
      for (int i = 0; i < 8; i++){
            p[i].double_word[0] = store_byte[i];
      }

      //Comparing Bytes
      for (int i = 0; i < 8; i++){

            if (p[i].double_word[0] !=
store_byte_expected_data[i].double_word[0]){

```

```
            printf("Mismatch at index %d:\n", i);
            printf("Expected: {0x%016llx, 0x%016llx}\n",
store_byte_expected_data[i].double_word[0],
store_byte_expected_data[i].double_word[1]);
            printf("Actual:   {0x%016llx, 0x%016llx}\n",
p[i].double_word[0], p[i].double_word[1]);

            printf("Byte Test Failed!\n");
            return 0;
            }
      }
      printf("Byte Test Passed!\n");
} // store_byte_test
```

- **Output:**

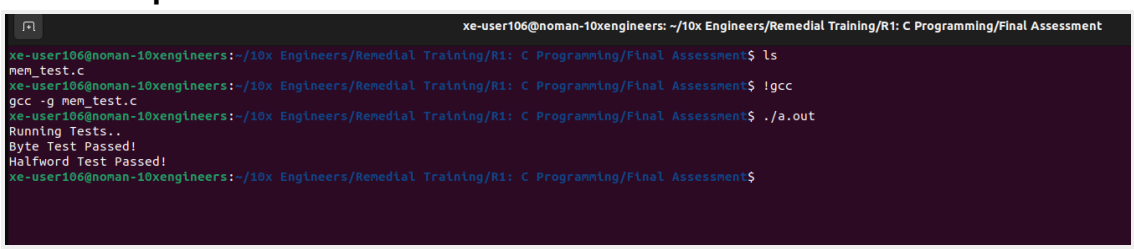


3. **Store Half-Word Test:**
   - **Code Snippet:**

```
int store_half_word_test (arr_t *p) {

// BEEF, ADBE, DEAD, EFDE, BEEF, ADBE, DEAD, EFDE
      unsigned long long store_half_word[] = { 0xBEEF, 0xADBE, 0xDEAD,
0xEFDE, 0xBEEF, 0xADBE, 0xDEAD, 0xDE, 0xEF };
      int b = 0;

      //Setting Bits
      for (int i = 0; i < 8; i++){
            b += 2;
            store_half_word[i+1] =  store_half_word[i+1] << (b*4);
      }

      //Store_Halfwords
      for (int i = 0; i < 8; i++){
            p[i].double_word[0] =   store_half_word[i];
            p[7].double_word[1] =   store_half_word[8];

      }
```

```
      //Comparing Halfwords
      for (int i = 0; i < 8; i++){
            if ( (p[i].double_word[0] !=
store_half_word_expected_data[i].double_word[0]) || (p[7].double_word[1]
!=  store_half_word_expected_data[7].double_word[1]) ){

            printf("Mismatch at index %d:\n", i);
            printf("Expected: {0x%016llx, 0x%016llx}\n",
store_half_word_expected_data[i].double_word[0],
store_half_word_expected_data[i].double_word[1]);
            printf("Actual:   {0x%016llx, 0x%016llx}\n",
p[i].double_word[0], p[i].double_word[1]);

            printf("Halfword Test Failed!\n");
            return 0;
            }
      }
      printf("Halfword Test Passed!\n");

} // store_half_word_test
```

- **Output:**



4. **Store Word Test:**
   - **Code Snippet:**

```
int store_word_test (arr_t *p) {

// DEADBEEF, F00DC0DE
      unsigned long long store_word[] = { 0xDEADBEEF, 0xDEADBEEF,
0xDEADBEEF, 0xDEADBEEF, 0xF00DC0DE, 0xF00DC0DE, 0xF00DC0DE, 0xF00DC0DE,
0xF0, 0xF00D, 0xF00DC0 };

      int b = 0;

      //Setting Bits
```

```
      for (int i = 0; i < 8; i++){
            b += 2;
            //printf("i = %d - - - - %llx\n", i, store_word[i+1]);
            store_word[i+1] = store_word[i+1] << (b * 4);
            //printf("i = %d - - - - %llx\n", i, store_word[i+1]);
      }

      //Store Words
      for (int i = 0; i < 8; i++){
            p[i].double_word[0] = store_word[i];
            p[5].double_word[1] = store_word[8];
            p[6].double_word[1] = store_word[9];
            p[7].double_word[1] = store_word[10];
      }

      //Comparing Words
      for (int i = 0; i < 8; i++){

            if ( ( p[i].double_word[0] !=
store_word_expected_data[i].double_word[0] ) ||
            ( i>=5 && p[i].double_word[1] !=
store_word_expected_data[i].double_word[1] ))
            {

            printf("Mismatch at index %d:\n", i);
            printf("Expected: {0x%016llx, 0x%016llx}\n",
store_word_expected_data[i].double_word[0],
store_word_expected_data[i].double_word[1]);
            printf("Actual:   {0x%016llx, 0x%016llx}\n",
p[i].double_word[0], p[i].double_word[1]);
            printf("Store Word Test Failed!\n");
            return 0;
            }
      }
      printf("Store Word Test Passed!\n");

} // store_word_test
```

- **Output:**

```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Final Assessment$ ls
mem_test.c
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Final Assessment$ !gcc
gcc -g mem_test.c
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Final Assessment$ ./a.out
Running Tests..
Byte Test Passed!
Halfword Test Passed!
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Final Assessment$ !gcc
gcc -g mem_test.c
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Final Assessment$ ./a.out
Running Tests..
Byte Test Passed!
Halfword Test Passed!
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Final Assessment$ !gcc
gcc -g mem_test.c
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Final Assessment$ ./a.out
Running Tests..
Byte Test Passed!
Halfword Test Passed!
Store Word Test Passed!
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programming/Final Assessment$
```

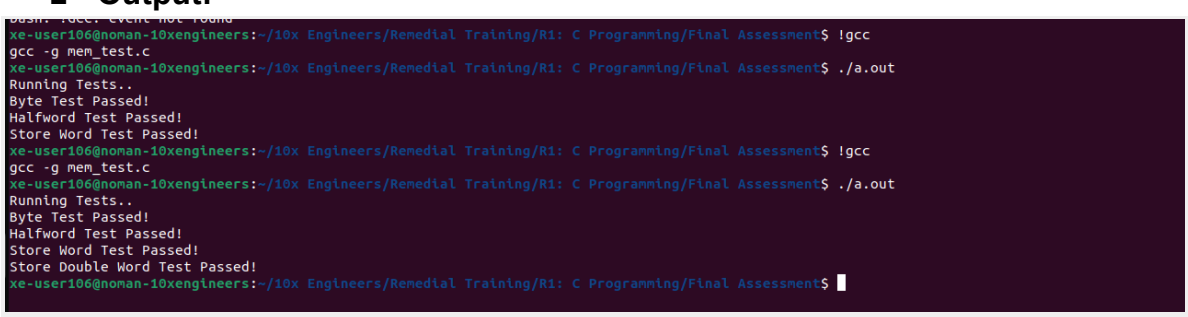
5. Store Double Word Test:

- **Code Snippet:**

```
int store_double_word_test (arr_t *p) {

// DEADBEEFF00DC0DE
      unsigned long long store_double_word[] = { 0xDEADBEEFF00DC0DE,
0xDEADBEEFF00DC0DE, 0xDEADBEEFF00DC0DE, 0xDEADBEEFF00DC0DE,
0xDEADBEEFF00DC0DE, 0xDEADBEEFF00DC0DE, 0xDEADBEEFF00DC0DE,
0xDEADBEEFF00DC0DE, 0xDE, 0xDEAD, 0xDEADBE, 0xDEADBEEF, 0xDEADBEEFF0,
0xDEADBEEFF00D, 0xDEADBEEFF00DC0 };

      int b = 0;

      //Setting Bits
      for (int i = 0; i < 8; i++){
            b += 2;
            store_double_word[i+1] <<= b * 4;
      }


      int x = 8;
      //Storing Double Words
      for (int i = 0; i < 8; i++){
            p[i].double_word[0] = store_double_word[i];

            if ( i>=1 ){
            p[i].double_word[1] = store_double_word[x];
            x++;
            }
      }
```

```
      //Comparing Double Words
      for (int i = 0; i < 8; i++){
            if ( (p[i].double_word[0] !=
store_double_word_expected_data[i].double_word[0]) ||
             ( i>=1 && p[i].double_word[1] !=
store_double_word_expected_data[i].double_word[1])){

            printf("Mismatch at index %d:\n", i);
            printf("Expected: {0x%016llx, 0x%016llx}\n",
store_double_word_expected_data[i].double_word[0],
store_double_word_expected_data[i].double_word[1]);
            printf("Actual:   {0x%016llx, 0x%016llx}\n",
p[i].double_word[0], p[i].double_word[1]);

            printf("Store Double Word Test Failed!\n");
            return 0;

            }
      }

            printf("Store Double Word Test Passed!\n");
} // store_double_word_test
```

- **Output:**



6. **Printing any Mismatches:**
   In order to print mismatches, we need to alter the store strings, let's change the string for **store_word[ ].** Here's a modified version of the code:

```
int store_word_test (arr_t *p) {

// DEADBEEF, F00DC0DE
```

```
      unsigned long long store_word[] = { 0xDEADBEEF, 0xDEADBEEF, 0xDEADBEEF, 0xDEADBEEF, 0xF00DC0DE, 0xF00DC0DE, 0xF00DC0DE, 0xF00DC0DE, 0xF0, 0xF00D, 0xF00DC0 };
```

Before

```
int store_word_test (arr_t *p) {

// DEADBEEF, F00DC0DE
      unsigned long long store_word[] = { 0xDEADBEEF, 0xDEADBEEF, 0xBEEFDEAD, 0xDEADBEEF, 0xF00DC0DE, 0xF00DC0DE, 0xF00DC0DE, 0xF00DC0DE, 0xF0, 0xF00D, 0xF00DC0 };
```

After

- **Output:**