**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);

}

1. **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;

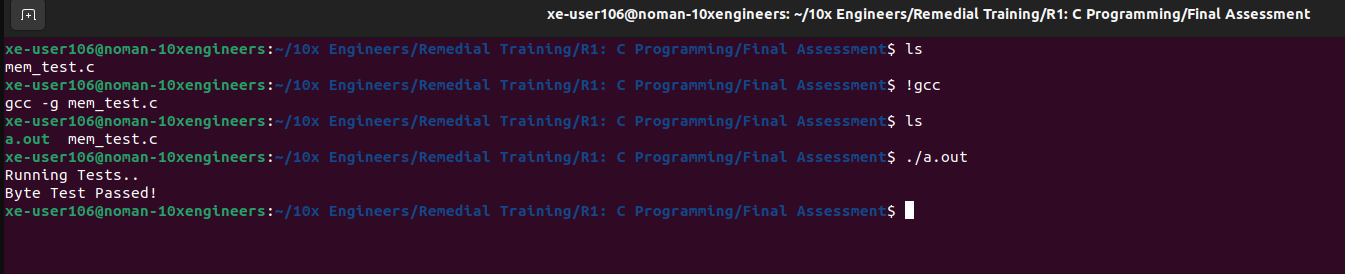
}

}

printf("Byte Test Passed!\n");

} // store\_byte\_test

* + **Output:**

****

1. **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;

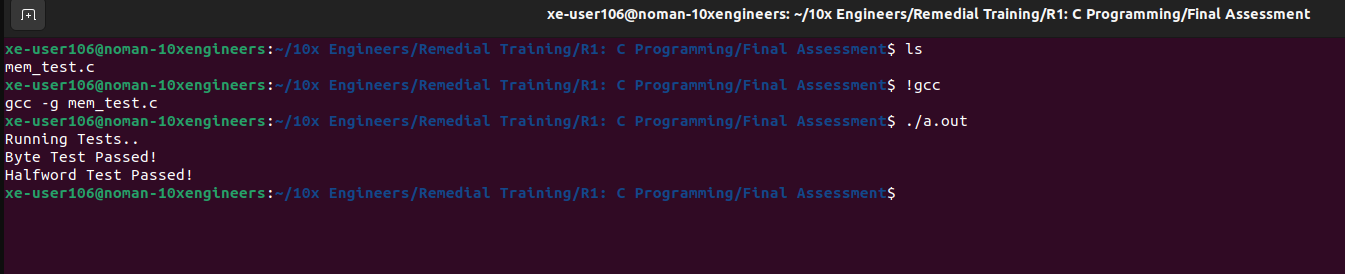
}

}

printf("Halfword Test Passed!\n");

} // store\_half\_word\_test

* + **Output:**



1. **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:**



1. **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;

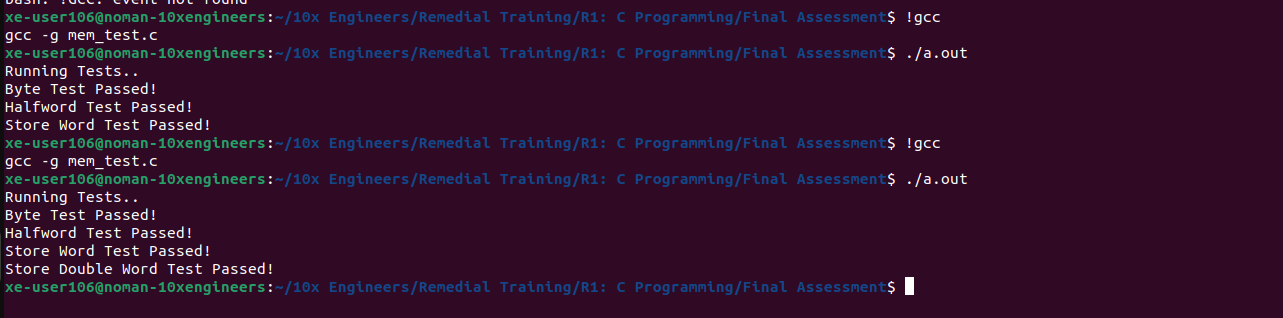
}

}

printf("Store Double Word Test Passed!\n");

} // store\_double\_word\_test

* + **Output:**



1. **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:**

