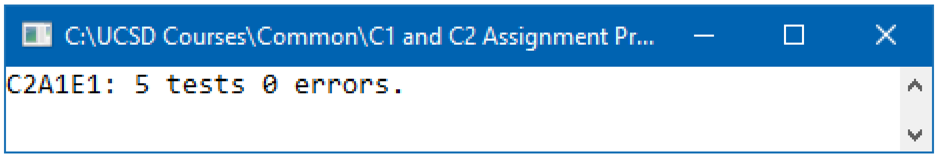
Exercise 3 *(4 points total (0.2 per answer) – No program required)*

```
...the usual title block Student/Course/Assignment information goes here...

 1. D
 2. A
 3. C
 4. B
 5. E
 6. C
 7. D
 8. E
 9. A
10. C
11. A
12. C
13. D
14. B
15. A
16. B
17. D
18. E
19. B
20. E
```

Exercise 1 *(2 points – C Program)*

```
/*
 *  ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains macros:
 *    Product: Produces the product of its two parameters.
 *    Negate: Produces the negation of its parameter.
 *    Elements: Produces a count of the number of elements in its array
 *              type parameter.
 */

#ifndef C2A1E1_MACROS_H
#define C2A1E1_MACROS_H

#define Product(a,b) ((a)*(b))
#define Negate(a) (-(a))
#define Elements(arrayDesig) (sizeof(arrayDesig)/sizeof(*(arrayDesig)))

#endif
```

C2A1E1 Screen Shot

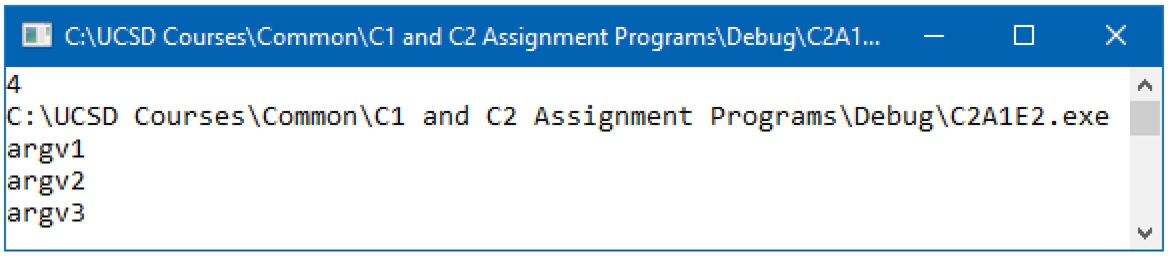


```
C2A1E1: 5 tests 0 errors.
```

Exercise 2 *(2 points – C Program)*

```c
/*
 *  ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function:
 *    main: Displays the value of argc and all command line argument strings.
 */

#include <stdio.h>

/*
 * Display the value of argc and all command line
 * argument strings on separate lines.
 */
int main(int argc, char *argv[])
{
   int argIx;

   printf("%d\n", argc);
   /* Loop to display all arguments. */
   for (argIx = 0; argIx < argc; ++argIx)
      printf("%s\n", argv[argIx]);

   return 0;
}
```

C2A1E2 Screen Shot
(argv[1], argv[2], and argv[3] values = argv1 argv2 argv3)

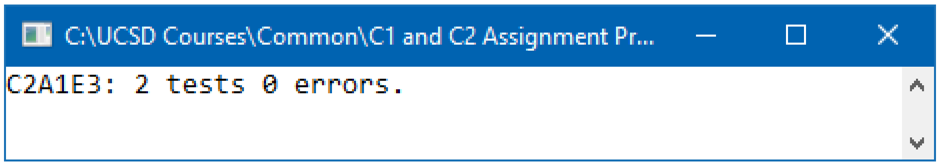

C:\UCSD Courses\Common\C1 and C2 Assignment Programs\Debug\C2A1...

```
4
C:\UCSD Courses\Common\C1 and C2 Assignment Programs\Debug\C2A1E2.exe
argv1
argv2
argv3
```

Exercise 3 *(2 points – C Program)*

```c
/*
 *  ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function:
 *    FindFirstInt: Finds the first occurrence of a value in an array.
 */

#include <stddef.h>

/*
 * FindFirstInt finds the first occurrence of <value> in the array
 * that has <count> elements represented by <ptr>.  If the value is
 * found a pointer to that element is returned.  Otherwise, a null
 * pointer is returned.
 */
int *FindFirstInt(const int *ptr, size_t count, int value)
{
   /* Pointer to end of array. */
   const int *end = ptr + count;
   /* Loop to find first occurrence in array. */
   for (; ptr < end; ++ptr)
      if (*ptr == value)
         return (int *)ptr;
   return 0;
}
```

C2A1E3 Screen Shot

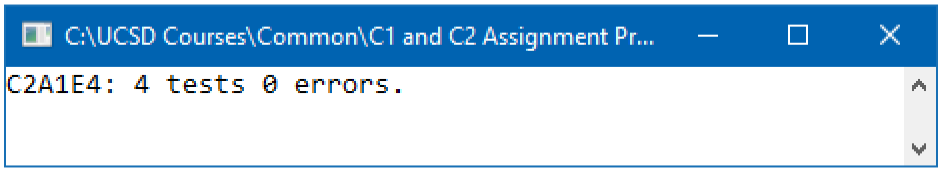


```
C2A1E3: 2 tests 0 errors.
```

Exercise 4 *(2 points – C Program)*

```c
/*
 *  ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function:
 *    StrToUpper: Copies a string, converting to uppercase in the copy.
 */

#include <string.h>
#include <ctype.h>
/*
 * StrToUpper copies the string in <source> into the memory in
 * <destination>, converting any lowercase letters to uppercase in the
 * copy.  The length of the string, not including the null terminator
 * character, is returned.
 */
size_t StrToUpper(char destination[], const char source[])
{
   const char *originalDestination = destination;
   /* Copy character-at-a-time until null character is copied. */
   while (*destination++ = (char)toupper(*source++))
      ;
   return (size_t)(destination - originalDestination - 1);
}
```

C2A1E4 Screen Shot

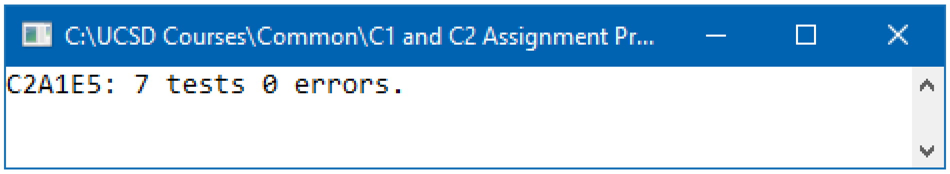


```
C2A1E4: 4 tests 0 errors.
```

Exercise 5 *(2 points – C Program)*

```c
/*
 *  ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function:
 *    ResizeAlloc: Dynamically resizes or creates a dynamic allocation.
 */

#include <stdlib.h>
#include <string.h>
/*
 * ResizeAlloc mimics the standard C library, realloc function, except
 * that ResizeAlloc has a 3rd parameter named <oldSize> that specifies
 * the number of bytes in the old allocation.
 */
void *ResizeAlloc(void *pOld, size_t newSize, size_t oldSize)
{
   void *pNew = NULL;
   /* If newSize != 0 and allocation succeeds. */
   if (newSize != 0 && (pNew = malloc(newSize)) != NULL)
   {
      /* If an allocation already exists. */
      if (pOld != NULL)
      {
         /* Prevent copying from overrunning the new block. */
         if (oldSize > newSize)
            oldSize = newSize;
         /* Copy from old block into new, then free old. */
         memcpy(pNew, pOld, oldSize);
         free(pOld);
      }
   }
   return pNew;
}
```

C2A1E5 Screen Shot



```
C2A1E5: 7 tests 0 errors.
```

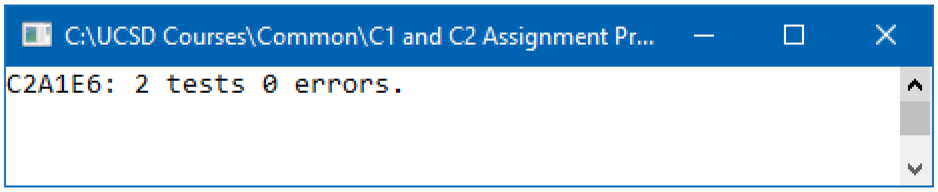
Exercise 6 *(2 points – C Program)*

```c
/*
 *  ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains function:
 *    AppendFile: Appends the contents of one file onto another.
 */

#include <stdio.h>

/*
 * AppendFile appends the contents of the file named in <inFile> onto
 * the end of the file named in <outFile>.  If either file fails to
 * open a non-0 value is returned.  Otherwise, 0 is returned.
 */
int AppendFile(const char *inFile, const char *outFile)
{
   FILE *inFp, *outFp;

   /* Open input file & check for failure. */
   if ((inFp = fopen(inFile, "rb")) == NULL)
   {
      perror(inFile);
      return 1;
   }

   /* Open output file & check for failure. */
   if ((outFp = fopen(outFile, "ab")) == NULL)
   {
      fclose(inFp);
      perror(outFile);
      return 1;
   }

#if 0                           /* Version 1: Append one character at a time. */
   int inChar;
   while ((inChar = getc(inFp)) != EOF)
      putc(inChar, outFp);
#else                           /* Version 2: Append block at a time. */
   #define BLOCK_SIZE 1000u
   size_t bytesRead;
   /* Loop until all bytes have been read and appended. */
   do
   {
      char buf[BLOCK_SIZE];
      /* Read BLOCK_SIZE bytes maximum. */
      bytesRead = fread(buf, 1, BLOCK_SIZE, inFp);
      /* Write all bytes just read. */
      if (bytesRead != 0)
         fwrite(buf, 1, bytesRead, outFp);
   } while (bytesRead == BLOCK_SIZE);
#endif

   fclose(inFp);
   fclose(outFp);

```

```
    return 0;
}
```

C2A1E6 Screen Shot is on the next page...

C2A1E6 Screen Shot

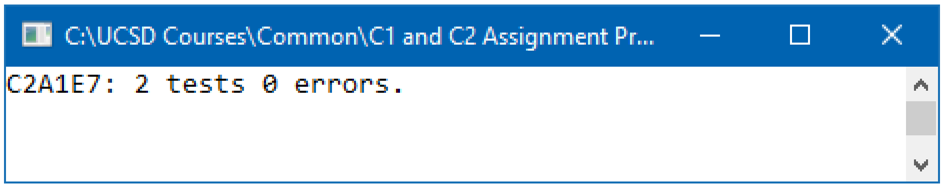


```
C2A1E6: 2 tests 0 errors.
```

Exercise 7 *(2 points – C++ Program)*

```
//
//  ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function:
//    AppendFile: Appends the contents of one file onto another.
//

#include <fstream>
#include <iostream>
using namespace std;

//
// AppendFile appends the contents of the file named in <inFile> onto
// the end of the file named in <inFile>.  If either file fails to
// open a non-0 value is returned.  Otherwise, 0 is returned.
//
int AppendFile(const char *inFile, const char *outFile)
{
   ifstream ifStmIn;
   ofstream ofStmOut;

   // Open input file & check for failure.
   ifStmIn.open(inFile, ios_base::binary);
   if (!ifStmIn.is_open())
   {
      cerr << "Input file open failure: \"" << inFile << "\".\n\n";
      return 1;
   }

   // Open output file & check for failure.
   ofStmOut.open(outFile, ios_base::binary | ios_base::app);
   if (!ofStmOut.is_open())
   {
      ifStmIn.close();
      cerr << "Output file open failure: \"" << outFile << "\".\n\n";
      return 1;
   }

#if 0                         // Version 1: Append one character at a time.
   int inChar;
   while ((inChar = ifStmIn.get()) != EOF)
      ofStmOut.put((char)inChar);
#else                         // Version 2: Append block at a time.
   const unsigned BLOCK_SIZE = 1000u;
   streamsize bytesRead;
   // Loop until all bytes have been read and appended.
   do
   {
      char buf[BLOCK_SIZE];
      // Read BLOCK_SIZE bytes maximum.
      ifStmIn.read(buf, BLOCK_SIZE);
      // Write all bytes just read.
      if ((bytesRead = ifStmIn.gcount()) != 0)
         ofStmOut.write(buf, bytesRead);
   } while (bytesRead == BLOCK_SIZE);
#endif

    ifStmIn.close();
    ofStmOut.close();

    return 0;
}
```

C2A1E7 Screen Shot



```
C2A1E7: 2 tests 0 errors.
```

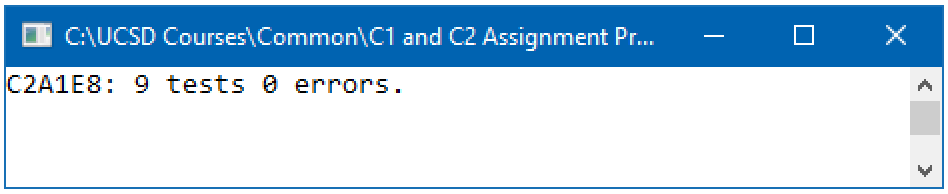
Exercise 8 *(2 points – C++ Program)*

```
   *************************************** FILE C2A1E8_Employee.h ***************************************
//
//  ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains:
//    The definition of class type Employee
//    The definitions of all but one of its member functions.
//

#ifndef C2A1E8_EMPLOYEE_H
#define C2A1E8_EMPLOYEE_H

// Definition of data type "class Employee"
class Employee
{
public:
   void Set(const char *str);
   void Set(int value = 25) {age = value;}
   void Set(const float &value) {raise = value;}
   void Set(const double *pValue) {salary = *pValue;}

   char *Get(char **outVar) const {return *outVar = name;}
   int Get(int &outVar) const {return outVar = age;}
   float &Get(float &outVar) const {return outVar = raise;}
   inline double Get(double *outVar) const;

private:
   char *name;
   int age;
   float raise;
   double salary;
};

// Define inline member function Employee::Get.  It returns the value
// of the salary data member and also places that value in the address
// pointed to by its parameter.
double Employee::Get(double *outVar) const
{
   return *outVar = salary;
}

#endif
```

-- EXERCISE CONTINUES ON NEXT PAGE --

```
   ************************************* FILE C2A1E8_Employee.cpp **************************************
//
//  ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains Employee member function:
//    Employee::Set: Deep copies the employee's name into the Employee object.
//

#include <cstring>
#include "C2A1E8_Employee.h"

// Set the Employee's name to the string in <str>
// by creating a "deep" copy of it.
void Employee::Set(const char *str)
{
   size_t length = strlen(str) + 1;
   name = new char[length];
   memcpy(name, str, length);
}
```

C2A1E8 Screen Shot



```
C2A1E8: 9 tests 0 errors.
```