```
Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming II
Section 149123, Ray Mitchell
July 13, 2020
C2A1E0_Quiz.txt
Quiz Answers

 1. B
 2. D
 3. C
 4. D
 5. C
 6. B
 7. D
 8. A
 9. B
10. A
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A1E1_Macros.h
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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)))
// OR  #define Elements(arrayDesig) (*(&(arrayDesig) + 1) - (arrayDesig))
#endif
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A1E2_main.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function:
//    main: Displays the value of argc and all command line argument strings.
//

#include <stdio.h>

//
// Function main loops to display the value of argc and all command line
// argument strings on separate lines.
//
int main(int argc, char *argv[])
{
   printf("%d\n", argc);
   // Loop to display all arguments.
   for (int argIx = 0; argIx < argc; ++argIx)
      printf("%s\n", argv[argIx]);

   return 0;
}
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A1E3_FindFirstInt.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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)
{
   // Loop to find first occurrence in array.
   // Return a pointer to it or NULL if not found.
   for (const int *end = ptr + count; ptr < end; ++ptr)
      if (*ptr == value)
         return (int *)ptr;
   return 0;
}
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A1E4_StrToUpper.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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);
}
```

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A1E5_ResizeAlloc.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A1E6_AppendFile.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// 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)
{
   // Open input file & check for failure.
   ifstream ifStmIn;
   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.
   ofstream ofStmOut;
   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;
   }

   // Version 1: Append one character at a time.
   for (int inChar; (inChar = ifStmIn.get()) != EOF;)
      ofStmOut.put((char)inChar);

#if 0
   // 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;
}
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A1E7_Employee.h
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains:
//    The definition of class type Employee
//    The definitions of all but one of its member functions.
//

#ifndef C2A1E7_EMPLOYEE_H
#define C2A1E7_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
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming II
// Section 149123, Ray Mitchell
// June 25, 2019
// C2A1E7_Employee.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains Employee member function:
//    Employee::Set: Deep copies the employee's name into the Employee object.
//

#include <cstring>
#include "C2A1E7_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);
}
```