Started on	Monday, October 24, 2022, 11:46 AM
State	Finished
Completed on	Monday, October 24, 2022, 11:47 AM
Time taken	1 min 36 secs

Question **1**

Correct

Points out of 1.00

Array definition: An array definition consists of a declaration specifier, an identifier, a dimension, followed by an optional comma-separated list of values enclosed in braces:

```
type-specifier name [ dimension ] = { initialization-list }opt;
```

type-specifier can be any of the simple data types we've studied so far including **signed** and **unsigned** versions of **char**, **int**, **long**, and floating-point types such as **float**, **double**, and **long double** except **void**. Later in the course we'll study about storage specifiers and type qualifiers and expand on the declaration specifier.

Square brackets surround ***dimension*** which specifies the number of elements in the array. ***dimension*** must be a constant integral expression, that is, it must be possible to evaluate the value of ***dimension*** at compile-time and the type of the expression must be integral. Note that the constant integral expression ***dimension*** must evaluate to a value greater than or equal to **1**. Note that C99 introduced variable-length arrays which allows the dimension to be specified at run-time. The committee changed its mind and C11 makes variable-length arrays optional. But the big thing is that variable-length arrays have always been illegal. Therefore, assume throughout this course and all assessments related to this course that ***variable-length arrays are illegal*** and this is enforced during compilation by using the **-Werror=vla** option of gcc.

initialization-list sets array elements to specific values and is ***optional***. The number of values in the initialization list may be less than ***dimension***, but not more. ***dimension*** is optional if you include ***initialization-list*** and explicitly initialize all values.

Array operations: Arrays are not first-class citizens: an array cannot be initialized with another array, nor can one array be assigned to another. To copy one array into another, each element must be copied in turn.

The only operations that can be performed directly on an array name are the application of the **sizeof** and address-of (**&**) operators. For the **sizeof** operator, the result is the number of bytes occupied by the array. That is, if each element of the array is of type **T** and the array dimension is **n**, the result of the **sizeof** operator is equal to **n** times **sizeof(T)**.

Does the following array definition compile?

```
int x[100];
```

Select one:

- True ✔
- False

Question **2**

Correct

Points out of 1.00

When we define an array, as in

```
int x[3];
```

the name **x** refers to a set of 3 unnamed **int** elements that are contiguously stored in memory. We make references to each of these three unnamed **int** elements using the ***subscript operator*** (also known as array or index operator) which has the format

```
x [ integral-expression ]
```

C/C++ arrays are always "0-origin". That is, the definition **int x[3]** defines the elements **x[0]**, **x[1]**, and **x[2]**.

Write the exact output printed to standard output by the following code fragment:

```
float x[] = {1.1f, 2.2f, 3.3f};
printf("%lu", sizeof(x[1]));
```

Answer: 4 ✔

Question **3**

Correct

Points out of 1.00

A ***string literal*** or ***string constant*** is a sequence of zero or more characters enclosed in double quotes, as in

```
"I am a string"
```

or

```
"" /* the empty string */
```

The quotes are not part of the string, but serve only to delimit it. Examples of string literals are

```
"hello world"
"total expenditures: "
"C comments begin with '/*'.\n"
```

String literals can be concatenated at compile time:

```
"hello, " "world"
```

is equivalent to

```
"hello, world"
```

Technically, a string literal is an array of characters with the internal representation having a null character **'\0'** at the end. This means that the physical storage required is one more than the number of characters written before the quotes. This also means that a string literal containing a single character is not the same as a character constant. The string literal **"a"** is an array of 2 **char** elements (the first **char** element having the value **'a'** and the second **char** value having the null character **'\0'**). On the other hand, character constant **'a'** has an integral value 97 (in the ASCII character set) and is of type **int**.

Since a string literal is an array of characters, we can use the subscript operator to access individual characters in the array, as in

```
"representation"[2]
```

which evaluates to the char value equivalent to **'p'**.

What is the value printed to standard output by the following code fragment

```
printf("%lu", sizeof("representation"));
```

Answer: 15 ✔

Question **4**

Correct

Points out of 1.00

Does the following array definition compile?

```
int x[5] = {1, 2, 3, 4, 5};
```

Select one:

- True ✔
- False

Question **5**

Correct

Points out of 1.00

An array subscript is an _____

Select one:

- element in an array
- alternate name for an array
- floating-point number that indicates the position of an array element
- integral number that represents the highest value stored within an array
- integral expression that indicates the position of an array element ✔

Question **6**

Correct

Points out of 1.00

Does the following array definition compile?

```
int x[] = {1, 2, 3, 4, 5};
```

Select one:

- True ✔
- False

Question **7**

Correct

Points out of 1.00

Any expression that evaluates to an integral value may be used as an operand to the subscript operator.

Select one:

- True ✔
- False

Question **8**

Correct

Points out of 1.00

In every array, a subscript is out of bounds when it is _____

Select one:

- 999
- negative ✔
- 0
- 1

Question **9**

Correct

Points out of 1.00

Does the following array definition compile?

```
int N = 5;
int x[N] = {1, 2, 3, 4, 5};
```

Select one:

- True
- False ✔

Question **10**

Correct

Points out of 1.00

When you use a subscript value that is negative or higher than the number of elements in an array, _____

Select one:

- a value in a memory location that is outside the area occupied by the array will be accessed, but only if the value is the correct data type
- the compiler will flag this as an error
- execution of the program stops and an error message is issued
- a value in a memory location that is outside the area occupied by the array will be accessed ✔

Question **11**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment when compiled using 64-bit gcc:

```
/*
The <limits.h> header provides macros that define the range of each
integer type (including the character types).
*/
#include <limits.h>

unsigned long int x = ULONG_MAX;
printf("%lu", sizeof(x));
```

Answer: 8 ✔

Question **12**

Correct

Points out of 1.00

Does the following array definition compile?

```
double x[5.5] = {1.1, 2.2, 3.3, 4.4, 5.5};
```

Select one:

- True
- False ✔

Question **13**

Correct

Points out of 1.00

The gcc option **-Werror=vla** ensures that array size must be determined at compile time.

Select one:

- True ✔
- False

Question **14**

Correct

Points out of 1.00

Suppose that you've defined an array of **int**s named **grades** that has 100 elements. Which of the following must be true?

Select one:

- **grades[99]** is the largest value in the array
- **grades[100]** is the last element of the array
- **grades[2]** is stored adjacent to **grades[4]**
- **grades[100]** is out of bounds ✔
- **grades[0]** is smaller than **grades[1]**

Question **15**

Correct

Points out of 1.00

Does the following array definition compile?

```
#define N (5)
int x[N] = {1, 2, 3, 4, 5};
```

Select one:

- True ✔
- False

Question **16**

Correct

Points out of 1.00

When an array is defined, C automatically sets the value of its elements to zero.

Select one:

- True
- False ✔

Question **17**

Correct

Points out of 1.00

The subscripts of an array must always evaluate to _____ type.

Select one:

- fractional
- pointer
- integral ✔
- string
- floating-point

Question **18**

Correct

Points out of 1.00

Is the following array definition legal?

```
int x[4] = {1, 2, 3, 4, 5};
```

Select one:

- True
- False ✔

Question **19**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
/*
The <limits.h> header provides macros that define the range of each integer type
(including the character types).
*/
#include <limits.h>

unsigned char x = UCHAR_MAX;
printf("%lu", sizeof(x));
```

Answer: 1 ✔

Question **20**

Correct

Points out of 1.00

Consider the following code fragment involving arrays. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int grades[5] = {10,20,30,40,50}, new_grades[5];
int i, sum;

new_grades = grades;
for (i = sum = 0; i < 5; ++i) {
  sum += new_grades[i];
}
printf("%d", sum);
```

Answer: CTE ✔

Question **21**

Correct

Points out of 1.00

Does the following array definition compile?

```
float x[5] = {1.1f, 2.2f, 3.3f, 4.4f, 5.5f};
```

Select one:

- (●) True ✔
- ( ) False

Question **22**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
/*
The <limits.h> header provides macros that define the range of each integer type
(including the character types).
*/
#include <limits.h>

unsigned int x = UINT_MAX;
printf("%lu", sizeof(x));
```

Answer: 4 ✔

Question **23**

Correct

Points out of 1.00

A function can return a value of type array.

Select one:

- ( ) True
- (●) False ✔

Question **24**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
/*
The <limits.h> header provides macros that define the range of
each integer type (including the character types).
*/
#include <limits.h>

short int x = SHRT_MAX;
printf("%lu", sizeof(x));
```

Answer: 2 ✔

Question **25**

Correct

Points out of 1.00

Each element in an array must have the same _____ as the others.

Select one:

- value
- data type ✔
- subscript
- memory address

Question **26**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
printf("%lu", sizeof('a'));
```

Answer: 4 ✔

Question **27**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
/*
The <limits.h> header provides macros that define the range of
each integer type (including the character types).
*/
#include <limits.h>

signed int x = INT_MAX;
printf("%lu", sizeof(x));
```

Answer: 4 ✔

Question **28**

Correct

Points out of 1.00

In the definition of the string variable **str**,

```
char str[] = "Sunny";
```

the initializer **"Sunny"** is a string literal.

Select one:

- True
- False ✔

Question **29**

Correct

Points out of 1.00

Does the following array definition compile?

```
int x[5] = {1};
```

Select one:

- True ✔
- False

Question **30**

Correct

Points out of 1.00

Suppose that you've defined an array of **int**s named **grades** that has 100 elements, and two of its elements are **grades[1]** and **grades[4]**. You know that _____.

Select one or more:

- [x] there are exactly two elements between those two elements ✔
- [ ] the two elements are at the same memory location
- [ ] the two elements hold the same value
- [ ] **grades[1]** is stored at higher memory address than **grades[4]**
- [x] **grades[1]** is stored at lower memory address than **grades[4]** ✔

Question **31**

Correct

Points out of 1.00

Is the following string variable definition legal?

```
char str[] = "Redmond" " " "WA";
```

Select one:

- (•) True ✔
- ( ) False

Question **32**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
/*
The <float.h> header provides macros that define the range and accuracy of the float, double, and
long double types.
Macros FLT_MAX, DBL_MAX, and LDBL_MAX provide the largest finite value that can be represented
by floats, doubles, and long doubles, respectively.
*/
#include <float.h>

double x = DBL_MAX;
printf("%lu", sizeof(x));
```

Answer: 8 ✔

Question **33**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
/*
The <limits.h> header provides macros that define the range of each integer type (including the
character types).
*/
#include <limits.h>

unsigned short int x = USHRT_MAX;
printf("%lu", sizeof(x));
```

Answer: 2 ✔

Question **34**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
double z[4];

/* some other code here */

z[0] = 5.5;
z[1] = -5.5;

/* standard library function fabs is described on page 757 of the text */
z[2] = z[3] = fabs(z[1]);
z[4] = z[0]+z[1]+z[2];
printf("%f", z[4]);
```

Answer: UDB ✔

Question **35**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
/*
The <float.h> header provides macros that define the range and accuracy of the float, double,
and long double types.
Macros FLT_MAX, DBL_MAX, LDBL_MAX specify the largest possible values that can be represented by
floats, doubles, and long doubles, respectively.
*/
#include <float.h>

float x = FLT_MAX;
printf("%lu", sizeof(x));
```

Answer: 4 ✔

Question **36**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment when compiling using 64-bit **gcc**:

```
/*
The <float.h> header provides macros that define the range and accuracy of the float, double,
and long double types.
Macros FLT_MAX, DBL_MAX, and LDBL_MAX specify the largest value that can be represented by
variables of type float, double, and long double, respectively.
*/
#include <float.h>

long double x = LDBL_MAX;
printf("%lu", sizeof(x));
```

Answer: 16 ✔

Question **37**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
/*
The <limits.h> header provides macros that define the range of each integer type (including the
character types).
*/
#include <limits.h>

signed char x = SCHAR_MAX;
printf("%lu", sizeof(x));
```

Answer: 1 ✔

Question **38**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment compiled using 64-bit **gcc**:

```
/*
The <limits.h> header provides macros that define the range of each integer type (including the
character types).
*/
#include <limits.h>

signed long int x = LONG_MAX;
printf("%lu", sizeof(x));
```

Answer: 8 ✔

Question **39**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
short x[] = {1, 2, 3};
printf("%lu", sizeof(x)/sizeof(x[1]));
```

Answer: 3 ✔

Question **40**

Correct

Points out of 2.00

Write the exact output printed to standard output by the code fragment. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int k;
double time[9];

/* some other code here */

for (k = 0; k <= 8; ++k)
  time[k] = (k-4)*0.1;
  printf("%d", k);
```

Answer: 9 ✔

Question **41**

Correct

Points out of 1.00

Write the exact output generated by the following code fragment. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int s[] = {3, 8, 15, 21, 30, 41};

for (int k = 0; k <= 5; k += 2)
  printf("%d|%d|", s[k], s[k+1]);
```

Answer: 3|8|15|21|30|41| ✔

Question **42**

Correct

Points out of 1.00

Write the exact output printed to standard output by the code fragment. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int s[] = {3, 8, 15, 21, 30, 41};

for (int k = 0; k <= 5; ++k)
  if (!(s[k]%2)) printf("%i|", s[k]);
```

Answer: 8|30| ✔

Question **43**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
int grades[5] = {10, 20, 30};
printf("%d,", grades[0]);
printf("%d,", grades[1]);
printf("%d,", grades[2]);
printf("%d,", grades[3]);
printf("%d", grades[4]);
```

Answer: 10,20,30,0,0 ✔

Question **44**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
char name[4] = {'E','l','m','o'};
fputs(name, stdout);
```

Answer: UDB ✔

Question **45**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
#define MAX_STR_LEN (256)
char str[MAX_STR_LEN];
str = "Sunny"; /* copy a string into a character array using the = operator */
```

Answer: CTE ✔

Question **46**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
#define MAX_STR_LEN (256)
char str1[MAX_STR_LEN] = "Sunny", str2[MAX_STR_LEN];
/* copy a character array to another character array using = operator */
str2 = str1;
```

Answer: CTE ✔

Question **47**

Correct

Points out of 1.00

Determine whether the following code fragment will compile or not. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
#define MAX_STR_LEN (256)

char str1[MAX_STR_LEN] = "Sunny", str2[MAX_STR_LEN] = "days";

if (str2 == str1) { // compare a character array to another character array
  fputs("alike", stdout);
} else {
  fputs("not alike", stdout);
}
```

Answer: not alike ✔

Question **48**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment?

```
char name[] = {'D','i','g','i','\0','p','e','n'};
fputs(name, stdout);
```

Answer: Digi ✔

Question **49**

Correct

Points out of 1.00

Consider the following code fragment involving an array. Write the additional initializer that must be added to ensure that the output printed to standard output is **Elmo**.

```
char name[] = {'E','l','m','o'};
fputs(name, stdout);
```

Answer: '\0' ✔

Question **50**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment is illegal or cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int grades[5] = {10,20,30,40,50,60};
int i, sum;
for (i = sum = 0; i < 5; ++i) {
  sum += grades[i];
}
printf("%d", sum);
```

Answer: CTE ✔

Question **51**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int grades[5] = {10,20,30,40,50};
int i, sum;
for (i = sum = 0; i <= 5; ++i) {
  sum += grades[i];
}
printf("%d", sum);
```

Answer: UDB ✔

Question **52**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
#define N (5)

int grades[N] = {10,20,30,40,50};
int i = 1, sum = grades[0];

while (i < N) {
  sum += grades[i++];
}
printf("%d", sum);
```

Answer: 150 ✔

Question **53**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int grades[5] = {10,20,30,40,50};
int i, sum;

for (i = sum = 0; i < 6; ++i) {
  sum += grades[i];
}
printf("%d", sum);
```

Answer: UDB ✔

Question **54**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
#define N (5)
int grades[N] = {-10, 20, -30, 40, -50};
int i = -1, sum = 0;

while (++i < N) {
  sum += grades[i];
}
printf("%d", sum);
```

Answer: -30 ✔

Question **55**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
#define N (5)

int grades[N] = {10,-20,30,-40,50};
int i = 0, sum = 0;

while (++i < N) {
  sum += grades[i];
}
printf("%d", sum);
```

Answer: 20 ✔

Question **56**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
#define N (5)

int grades[N] = {10,20,30,40,50};
int i = 0, sum = 0;

while (i++ < N) {
  sum += grades[i];
}
printf("%d", sum);
```

Answer: UDB ✔

Question **57**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the exact output printed to standard output.

```
int grades[5];
int i, N;

for (N = sizeof(grades)/sizeof(grades[0]), i = 0; i < N; ++i) {
  grades[i] = 2*i-3;
}

grades[0] = grades[4],
grades[4] = grades[1],
grades[2] = grades[3]+grades[0];

for (i = 0; i < N; ++i) {
  printf("%d%s", grades[i],(i==N-1?"":","));
}
```

Answer: 5,-1,8,3,-1 ✔

Question **58**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the exact output printed to standard output.

```
int grades[5];
int i, N;

for (N = sizeof(grades)/sizeof(grades[0]), i = 0; i < N; ++i) {
  grades[i] = 2*i+5;
  grades[i] -= (i%2 == 0) ? 3 : 0;
}

for (i = 0; i < N; ++i) {
  printf("%d%s", grades[i],(i==N-1?"":","));
}
```

Answer: 2,7,6,11,10 ✔

Question **59**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the exact output printed to standard output.

```
#define N (6)

int grades[N];
grades[0] = 5;
for (int i = 1; i < N; ++i) {
  grades[i] = i*i+5;
  grades[i] = (i > 2) ? 2*(grades[i] - grades[i-1]) : grades[i-1];
}

for (int i = 0; i < N; ++i) {
  printf("%d%s", grades[i],(i==N-1?"":","));
}
```

Answer: 5,5,5,18,6,48 ✔

Question **60**

Correct

Points out of 1.00

If the user enters the line through the keyboard (followed by the Enter/Return key)

```
today is a good day
```

what is the output of the following code fragment?

```
#define MAX_LEN (256)

char str[MAX_LEN+1];
fputs("Enter a sentence: ", stdout);
scanf("%s", str);
fputs(str, stdout);
```

Answer: today ✔

Question **61**

Correct

Points out of 1.00

A short note on functions **fgets**

fgets reads lines from an input stream. **fgets** takes 3 parameters: the base address of the buffer to store the read-in line, the maximum number of characters to read from the file minus one, and a pointer to an input file stream. So the call

```
fgets(buf, 81, infile)
```

says to read up to 80 characters from **infile** and store it into **buf**. **fgets** will read less than 80 characters if it reaches the end of the file or if its reads a newline character first. In any case, **fgets** stores a null character at the end of the array. Remember that the count given to **fgets** is one greater than the maximum number of characters it will read. Typically, this will be the actual size of your array. **fgets** will store the newline character into the array if it reads it. Since **fgets** takes an upper bound on the number of characters to read, it's better to use **fgets** especially if you're not sure how long the lines you're reading are. In such a case, simply give **stdin** as the third argument to **fgets**:

```
fgets(buf, 81, stdin)
```

This call will read up to 80 characters from standard input. **fgets** will return **NULL** when it reaches the end of the file without reading any characters.

A short note on functions **fputs**

Function **fputs** writes a line to a specified stream, so

```
fputs(buf, outfile);
```

writes the contents of **buf** to **outfile**. **fputs** does not append a newline character to the stream. Therefore, the call

```
fputs(buf, stdout)
```

is equivalent to

```
printf("%s", buf)
```

Now supposing a user enters the line through the keyboard (followed by Enter/Return)

```
today is a good day
```

what is the output of the following code fragment?

```
#define MAX_LEN (257)
char str[MAX_LEN];
fputs("Enter a sentence: ", stdout);
fgets(str, MAX_LEN, stdin);
fputs(str, stdout);
```

Answer: today is a good day ✔

Question **62**

Correct

Points out of 1.00

Consider the following code fragment involving an array. Assume the user enters the following line through the keyboard (followed by Enter/Return):

```
today is a good day
```

If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
#define MAX_LEN (13)
char str[MAX_LEN];
fputs("Enter a sentence: ", stdout);
fgets(str, MAX_LEN, stdin);
fputs(str, stdout);
```

Answer: today is a g ✔

Question **63**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int grades[] = {10,20,30,40,50};
printf("%lu", sizeof(grades));
```

Answer: 20 ✔

Question **64**

Correct

Points out of 1.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int sum = 0, grades[] = {-10,20,-30,40,-50};
unsigned int i;

for (i = 0; i < sizeof(grades)/sizeof(grades[0]); ++i) {
  sum += grades[i];
}
printf("%d", sum);
```

Answer: -30 ✔

Question **65**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
printf("%lu", sizeof(""));
```

Answer: 1 ✔

Question **66**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
printf("%lu", sizeof("a"));
```

Answer: 2 ✔

Question **67**

Correct

Points out of 1.00

Write the exact output printed to standard output by the following code fragment:

```
printf("%lu", sizeof("representation"[5]));
```

Answer: 1 ✔

Question **68**

Correct

Points out of 3.00

Write the exact output printed to standard output by the following code fragment:

```
int foo(int const array[], unsigned int elem_cnt) {
  int val = array[0];
  for (unsigned int i = 1; i < elem_cnt; ++i) {
    val = (array[i] < val) ? array[i] : val;
  }
  return val;
}

int x[] = {10, -10, 20, -20, 30, -30};
printf("%i", foo(x, sizeof(x)/sizeof(x[0])));
```

Answer: -30 ✔

Question **69**

Correct

Points out of 3.00

Write the exact output printed to standard output by the following code fragment (don't copy-paste-compile - similar questions will show up in the final test where you'll not have access to a compiler):

```
for (int i = 0; "representation"[i]; ++i) {
  fputc("representation"[i] - 'a' + 'A', stdout);
}
```

Select one:

- The string **"REPRESENTATION"** ✔
- Compile-time error
- Undefined behavior
- Nothing is printed because the loop is never executed
- The string **"representation"**

Question **70**

Correct

Points out of 3.00

Write the exact output printed to standard output by the following code fragment:

```
int foo(int const array[], unsigned int elem_cnt) {
  int val = array[0];
  for (unsigned int i = 1; i < elem_cnt; ++i) {
    val = (array[i] > val) ? array[i] : val;
  }
  return val;
}

int x[] = {10, -10, 20, -20, 30, -30};
printf("%i", foo(x, sizeof(x)/sizeof(x[0])));
```

Answer: 30 ✔

Question **71**

Correct

Points out of 2.00

Write the exact output printed to standard output by the code fragment. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int i, list[10] = {0};

/* some other code here */

for (i = 0; i <= 5; ++i) list[2*i+1] = i+2;
for (i = 0; i < 10; ++i) printf("%d|", list[i]);
```

Answer: UDB ✔

Question **72**

Correct

Points out of 3.00

Consider the following code fragment involving an array. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the exact output printed to standard output.

```
int i, grades[] = {-10,20,-30,40,-50};
for (i = sizeof(grades)/sizeof(grades[0]); i >= 0; --i) {
  printf("%d%s", grades[i],(i==0?"":","));
}
```

Answer: UDB ✔

Question **73**

Correct

Points out of 3.00

Write the exact output printed to standard output by the code fragment. If the code fragment cannot be compiled, write CTE (for compile-time error). If the code fragment generates undefined behavior (see pages 65 and 163 of text), write UDB (for undefined behavior). Otherwise, write the output printed to standard output.

```
int list[] = {2, 1, 2, 1, 1, 2, 3, 2, 1, 2};

printf("%d|", list[2]);
printf("%d|", list[list[2]]);
printf("%d|", list[list[2]+list[3]]);
printf("%d", list[list[list[2]]]);
```

Answer: 2|2|1|2 ✔

Question **74**

Correct

Points out of 3.00

Consider the following code involving string variables. If the user enters the line through the keyboard (followed by Enter/Return)

```
today is a good day
```

write the exact output printed to standard output by the code fragment:

```
int foo(char const array[]) {
  int i, val;

  for (i = val = 0; array[i]; ++i) {
    val = (array[i] == ' ') ? val+1 : val;
  }
  return val;
}

char str[257];
fputs("Enter a sentence: ", stdout);
fgets(str, 256, stdin);
printf("%i", foo(str));
```

Answer: 4 ✔

Question **75**

Correct

Points out of 3.00

Write the exact output printed to standard output by the following code fragment:

```
char foo(int digit) {
  return "0123456789ABCDEF"[digit];
}

char str[16] = {0};
int x = 54321, i = 0;

while (x) {
  str[i++] = foo(x%16);
  x /= 16;
}

str[i] = '\0';
for(--i; i >= 0; --i) {
  fputc(str[i], stdout);
}
```

Answer: D431 ✔

Question **76**

Correct

Points out of 3.00

Write the exact output printed to standard output by the following code fragment:

```
#define MAX_STR_LEN 256

void foo(char array1[], char const array2[]) {
 int i = 0, j = 0;
 while (array1[i++]);
 --i;
 while (array1[i++] = array2[j++]);
}

char str[MAX_STR_LEN] = "FourScore", str2[MAX_STR_LEN]= "AndSeven";
foo(str, str2);
fputs(str, stdout);
```

Answer: FourScoreAndSeven ✔

Question **77**

Correct

Points out of 3.00

Write the exact output printed to standard output by the following code fragment:

```
#define MAX_STR_LEN 256

void foo(char array1[], char const array2[]) {
 int i = 0;
 while (array1[i] = array2[i]) ++i;
}

char str[MAX_STR_LEN] = "FourScore", str2[MAX_STR_LEN];
foo(str2, str);
fputs(str2, stdout);
```

Answer: FourScore ✔

Question **78**

Correct

Points out of 3.00

Write the exact output printed to standard output by the following code fragment:

```
int foo(char const array[]) {
  int i = 0;

  while (array[i++]) ;
  return i-1;
}

char str[] = "FourScoreAndSeven";
printf("%i", foo(str));
```

Answer: 17 ✔

◄ Lecture 15 [10/17/2022]: Introduction to Character Strings

Jump to...

Lab 7: Using Pointers ►