Started on	Sunday, November 27, 2022, 5:54 PM
State	Finished
Completed on	Sunday, November 27, 2022, 6:25 PM
Time taken	30 mins 32 secs
Grade	**100.00** out of 100.00

Information

Operator Precedence Chart:

A sheet containing the precedence and associativity of C operators will be provided to you in the final test venue. For this review, you should print an operator precedence chart [available in the file folder "Final Test Details" on the meta course page].

Don't Forget to Submit Test

After you've completed the test, you must use the "**Submit All and Finish**" button to actually submit the test.

Standard Library Headers

Assume every question in this test has all necessary headers included. Don't choose "Doesn't Compile" as an answer only because code fragments don't include necessary standard library headers.

Programming Questions

Pre-check button allows you to compile, link, and execute your code. If the grader seems to freeze or hang with the **Pre-check** button showing an spinning icon, it is most possibly caused by an infinite loop. The simplest solution that will save you time is to kill the process running the bad submission. You can do this by quitting the question and then reattempting it. Just like with any editor, you can use **Ctrl-Z** to keep a copy of your code in the clipboard.

Click on **Reset** [at top-right of code window] to clear the code window. Use **CTRL-c** if you want to copy your code to the clipboard before clicking the **Reset** button.

Question **1**
Correct
3.00 points out of 3.00

Given the following definitions:

```
int i = 5, j = 10, *p = &i, *q = &j;
```

determine whether each of the following assignment statements is legal [because the statement doesn't cause a compile-time error] or illegal [because the statement causes a compile-time error].

`*p = &i;`	Statement is illegal	✓
`p = &q;`	Statement is illegal	✓
`p = q;`	Statement is legal	✓
`*p = *q;`	Statement is legal	✓
`p = i;`	Statement is illegal	✓
`p = *q;`	Statement is illegal	✓

Your answer is correct.

Question **2**

Correct

4.00 points out of 4.00

Given the following definitions:

```
int a[] = {15,13,2,7,9,1,8,3,6,4}, *p = &a[2], *q = a+7;
```

determine the value resulting from the evaluation of each of the following expressions.

`*(q-4)`	7	✓
`*(p+2)`	9	✓
`p-q`	-5	✓
`*p-*q`	-1	✓

Your answer is correct.

Question **3**

Correct

4.00 points out of 4.00

Given the following definitions:

```
int a[] = {/* some initializers*/}, *p = a+3;
```

for each of the following expressions, choose an equivalent expression from the listed choices.

`*p + 2`	a[3]+2	✓
`*(p+1)`	a[4]	✓
`p[4]`	a[7]	✓
`p-1`	&a[2]	✓

Your answer is correct.

Question **4**

Correct

2.00 points out of 2.00

Fill the boxes with the values printed to standard output by the corresponding **`printf`** statements.

```
int x = 10, y = 20, *p1 = &x, *p2 = &y;
x = *p1 + *p2;
printf("%d", x);   30 ✓
y = x - *p2;
*p1 -= *p2;
printf("%d", x);   20 ✓
```

Question **5**

Correct

3.00 points out of 3.00

Given the definition of function `foo`

```
int foo(int i) {
  static int f = 1;
  return f+=++i;
}
```

fill the boxes with the values printed to standard output by the corresponding `printf` statements.

```
int main(void) {
  printf("%d", foo(0));   2 ✓

  printf("%d", foo(1));   4 ✓

  printf("%d", foo(2));   7 ✓

  return 0;
}
```

Question **6**

Correct

3.00 points out of 3.00

Consider the following code fragment:

```
int vector[5] = {1, 2, 3, 4, 5}, *pv = vector, value = 3;
for (size_t i=0; i < sizeof(vector)/sizeof(int); ++i) {
  *pv++ *= value;
}
```

If the code fragment doesn't compile, write CTE [for compile-time error]. If the code fragment has undefined behavior at runtime, write UDB [for undefined behavior]. Otherwise, write the the sum of the values in array `vector` after the code fragment's execution.

Answer: 45 ✓

Question **7**

Correct

3.00 points out of 3.00

Write CTE if the following code fragment doesn't compile. Write UDB if the code fragment's execution causes undefined behavior at run-time. Otherwise, write the ***exact*** value printed to standard output by the code fragment.

```
int b[][3] = {5, 3, 9, 4, 1, 2, 6, 7, 8}, m = 0;
for (int k = 0; k < 3; ++k) {
  int j = 0;
  for (int i = k; i >= 0; i--) {
    m += b[i][j++];
  }
}
printf("%d", m);
```

Answer: 28 ✓

Question **8**

Correct

2.00 points out of 2.00

Write CTE if the following code fragment doesn't compile. Write UDB if the code fragment's execution causes undefined behavior at run-time. Otherwise, write the ***exact*** value printed to standard output by the code fragment.

```
int j = 0;
if (j++ == ++j) {
  printf("1");
} else {
  printf("2");
}
```

Answer: UDB ✓

Question **9**

Correct

2.00 points out of 2.00

Write CTE if the following code fragment doesn't compile. Write UDB if the code fragment's execution causes undefined behavior at run-time. Otherwise, write the ***exact*** value printed to standard output by the code fragment.

```
int j = 0;
if (j++ && ++j) {
  printf("1");
} else {
  printf("2");
}
```

Answer: 2 ✓

Question **10**

Correct

3.00 points out of 3.00

Given the following definitions

```
int i = 5, j = 6;
int const ci = 10, cj = 11;
int const *pci;
int *pi;
```

determine whether the subsequent assignment statements are legal or not.

`pci = &i;`	Assignment is legal ✓
`ci = 8;`	Assignment is illegal ✓
`*pci = 5;`	Assignment is illegal ✓
`j = ci;`	Assignment is legal ✓
`pi = &ci;`	Assignment is legal ✓
`pci = &cj;`	Assignment is legal ✓

Your answer is correct.

Question **11**

Correct

3.00 points out of 3.00

Fill the boxes with the values printed to standard output by the corresponding **printf** statements.

```
int a = 3, b = 8, c = 4, d = 1, e = 5;

printf("%d", a > b && b < 10 || ++a == 4);         1 ✓

printf("%d", a > b && b < 10 && ++a == 4);         0 ✓

printf("%d", a < b || b < c && c < d || d < e);    1 ✓
```

Question **12**
Correct
2.00 points out of 2.00

Determine whether the following identifiers can be used to define variables in a C program.

`extern`	The identifier is invalid	✓
`Const`	The identifier is valid	✓
`sizeof`	The identifier is invalid	✓
`printf`	The identifier is valid	✓
`myname50`	The identifier is valid	✓
`_tmp`	The identifier is valid	✓
`include`	The identifier is valid	✓
`is-integer`	The identifier is invalid	✓

Your answer is correct.

Question **13**
Correct
2.00 points out of 2.00

Write CTE if the following code fragment doesn't compile. Write UDB if the code fragment's execution causes undefined behavior at run-time. Otherwise, write the ***exact*** value printed to standard output by the code fragment.

```
int bar(int x) {
  return x % 3;
}

printf("%d", bar(bar(bar(19))));
```

Answer: 1 ✓

Question **14**
Correct
2.00 points out of 2.00

Fill the boxes with the values printed to standard output by the corresponding `printf` statements. Write -1 if you believe the following code fragment has compile-time or run-time error(s).

```
int flag = 1;
printf("%d", 5==3>6+11*7.5||1&&flag);    1 ✓

flag = 0;
printf("%d", 5==3>6+11*7.5||1&&flag);    0 ✓
```

Question **15**
Correct
3.00 points out of 3.00

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

```
char s[] = "HSjodi", *p = s;
for (; *p; ++p) {
  --*p;
}
printf("%s", s);
```

Answer: GRinch ✓

Question **16**
Correct
2.00 points out of 2.00

Given the following definitions:

```
int a[75] = { /* various comma-separated initializers */ };
int *p = a + 44;
```

rewrite the following expressions using array name **a** and subscript `[]` operator. ***Note that your answers should not contain any extraneous whitespace characters***.

`p + 5`	&a[49]	✓
`*(p+2)+4`	a[46]+4	✓
`*(p-1)-2`	a[43]-2	✓
`&*(p-12)`	&a[32]	✓
`(p-1)-2`	&a[41]	✓

Question **17**
Correct
3.00 points out of 3.00

Given the following definitions:

```
int i = 3, k[] = {2, 4, 6, 8, 10, 12}, *x = &i, *y = k;
double d = 1.5;
struct {
  int x, y;
  char *name;
} pt[] = {{200, 40, "begin"}, {300, 100, "end"}}, *pp = pt;
```

provide the precise type [that is, the type you would use to define a variable in a C program] of each expression and the value resulting from the expression's evaluation. If an expression's value cannot be specified, select NA to denote this fact.

Assume the expressions are non-cumulative, that is, every expression's evaluation will only consider the values that variables are initialized with.

Expression to evaluate	Type resulting from evaluation		Value resulting from expression	
`++pp->x`	int	✓	201	✓
`pt[1].x*i/5`	int	✓	180	✓
`k[i++]`	int	✓	8	✓
`k[++i]+d`	double	✓	11.5	✓
`pt[i--].y+50`	Undefined behavior	✓	NA	✓
`*(*pp.name+2)`	Compile-time error	✓	NA	✓
`*((*pp).name+2)`	char	✓	'g'	✓

Question **18**

Correct

7.00 points out of 7.00

Define functions **find_max** and **find_min** that satisfy the following use cases:

```
// use cases for functions find_min and find_max
int a1 = 10, b1 = 11;
printf("%d %d ", *find_max(&a1, &b1), find_min(a1, b1)); // prints 11 10
int a2 = 11, b2 = 1;
printf("%d %d ", *find_max(&a2, &b2), find_min(a2, b2)); // prints 11 1
int const *pa1 = &a1, *pb1 = &b2;
printf("%d %d\n", *find_max(pa1, pb1), find_min(*pa1, *pb1)); // prints 10 1
*find_max(&a1, &b1) *= 2; // doesn't compile!!!
```

You must write the definition of functions **find_max** and **find_min** without including any C standard library headers.

```
// define functions find_max and find_min without including any C standard library headers!!!
const int* find_max(const int*a, const int* b)
{
    if(*a>*b)
    {
        return a;
    }
    return b;
}
int find_min(const int a, const int b)
{
    if(a<b)
    {
        return a;
```

Correct

Evaluation details:

Evaluation:

-Summary of tests

```
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
int const* find_max(int const *lhs, int const *rhs) {
  return (*lhs > *rhs) ? lhs : rhs;
}

int find_min(int lhs, int rhs) {
  return lhs < rhs ? lhs : rhs;
}}
```

Question **19**

Correct

7.00 points out of 7.00

Define a function **nonvowel_count** that takes a read-only null-terminated character string and returns the number of non-vowel characters in the character string. For this exercise, assume lowercase Latin characters a, e, i, o, and u represent vowels.

Your definition of function **nonvowel_count** must not include any C standard library headers.

```
// Define your function here.
// NOTE: Including standard library functions will prevent your code from compiling!!!
int nonvowel_count(const char* str)
{
    int count = 0, i =0;
    while(str[i]!='\0')
    {
        if(str[i]!='a'&&str[i]!='e'&&str[i]!='i'&&str[i]!='o'&&str[i]!='u')
        {
            count++;
        }
        i++;
    }
    return count;
```

Correct

Evaluation details:

Evaluation:

-Summary of tests

```
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
int nonvowel_count(char const *str) {
  int count = 0;
  while (*str) {
    count = (*str != 'a' && *str != 'e' && *str != 'i' && *str != 'o' && *str != 'u') ?
count+1 : count;
    ++str;
  }
  return count;
}
```

Question **20**

Correct

7.00 points out of 7.00

Define a function **expand_string** that expands a null-terminated character string by replacing all tab characters in the string with spaces. Use the following use cases to guide your definition:

```
// use cases for function expand_string
char dst[1024] = {'\0'}, src1[] = {"today\tis\ta\tgood\tday"};

expand_string(dst, src1, 4);
// dst will now contain the characters "today    is    a    good    day"

printf("%s", dst);
printf("\n");

char const src2[] = {"never\tsay\tnever\tagain"};
expand_string(dst, src2, 2);
// dst will now contain the characters "never  say  never  again"

printf("%s", dst);
printf("\n");

char src3[] = {"C\tto\tshining\tC++"};
expand_string(dst, src3, 3);
// dst will now contain the characters "C   to   shining   C++"

printf("%s", dst);
```

Note: You must write the definition of function **expand_string** without including any C standard library headers.

```
// define function expand_string without including any C standard library headers!!!
void expand_string(char*dst, const char*src, int no)
{
    int i = 0, k=0;

    while(src[k]!='\0')
    {
        if(src[k]!='\t')
        {
            dst[i] = src[k];
        }
        else if (src[k]=='\t')
        {
            for(int i = 0; i<no; i++)
```

Correct

Evaluation details:

Evaluation:

-Summary of tests

```
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
void expand_string(char *dst, char const *src, int tab_size) {
  while (*src) {
    if (*src == '\t') {
      for (int i = 0; i < tab_size; ++i) {
        *dst++ = ' ';
      }
    } else {
      *dst++ = *src;
    }
    ++src;
  }
  *dst = '\0';
}
```

Question **21**

Correct

7.00 points out of 7.00

Define a function **count_whitespace** that counts the number of space, tab, and newline characters in a character string. Use the following use cases to guide your definition:

```
// use cases for function count_whitespace
char const src1[] = "today\tis a good  day\n";
int space, tab, newline;
count_whitespace(src1, &space, &tab, &newline);
printf("%d %d %d", space, tab, newline); // prints to standard output: 4 1 1

char src2[] = "  tomorrow\t\twill be a\nbetter day\n";
count_whitespace(src2, &space, &tab, &newline);
printf("%d %d %d", space, tab, newline); // prints to standard output stream: 5 2 2
```

Note: You must write the definition of function **count_whitespace** without including any C standard library headers.

```
// define function count_whitespace without including any C standard library headers!!!
void count_whitespace(const char*src, int*space, int*tab, int*newline)
{
    *newline =0;
    *tab =0;
    *space =0;
    int i=0;
    while(src[i]!='\0')
    {
        if(src[i] == '\t')
        {
            (*tab)++;
        }
        if(src[i] == '\n')
```

Correct

Evaluation details:

Evaluation:

-Summary of tests

```
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
void count_whitespace(char const *stm, int *space, int *tab, int *newline) {
  *space = *tab = *newline = 0;
  for (int i=0; stm[i]; ++i) {
    *space = (stm[i] == ' ') ? *space+1 : *space;
    *tab   = (stm[i] == '\t') ? *tab+1  : *tab;
    *newline = (stm[i] == '\n') ? *newline+1 : *newline;
  }
}
```

Question **22**

Correct

7.00 points out of 7.00

Define a function **my_tolower** that returns the lowercase equivalent of its parameter if the parameter is an uppercase Latin character and has a lowercase equivalent. If no such conversion is possible, the parameter is returned unchanged. Use the following use cases to guide your definition:

```
// use cases for function my_tolower
printf("%c", my_tolower('A')); // prints a
printf("%c", my_tolower('a')); // prints a
printf("%c", my_tolower('#')); // prints #
printf("%c", my_tolower('3')); // prints 3
printf("%c", my_tolower('M')); // prints m
```

Note: You must write the definition of function **my_tolower** without including any C standard library headers.

```
// define function my_tolower without including any C standard library headers!!!
char my_tolower(char var)
{
    if(var>='A'&& var <='Z')
    {
        return (char)(var-('A'-'a'));
    }
    return var;
}
```

Correct

Evaluation details:

Evaluation:

-Summary of tests

```
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
int my_tolower(int ch) {
  return (ch >= 'A' && ch <= 'Z') ? (ch-'A' + 'a') : ch;
}
```

Question **23**

Correct

7.00 points out of 7.00

Define a function `reverse` that reverses in-place [that is, without copying values to a temporary array] `double` values in a half-open range. Note that the function must neither use C standard library functions nor the subscript operator `[]`. Use the following use cases to guide your definition:

```
double d[9] = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};
reverse(d, d+9); // elements of d are now: 9.9 8.8 7.7 6.6 5.5 4.4 3.3 2.2 1.1

double d2[1] = {9.9};
reverse(d2, d2+1); // elements of d2 are now: 9.9

double d3[3] = {100.1, 99.2, 109.3};
reverse(d3, d3+2); // elements of d3 are now: 99.2 100.1 109.3
```

```
// Define function reverse here.
// NOTE: Using subscript operator or including standard library functions will prevent your code from compiling
void reverse(double*array, double* end)
{
    long int diff = (end--)- array;
    double tmp;
    diff/=2;

    for(int i=0; i<diff; i++)
    {
        tmp =*array;
        *array = *end;
        *end = tmp;
```

Correct

Evaluation details:

Evaluation:

-Summary of tests

```
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
void reverse(double *first, double *last) {
  while (first != last && first != --last) {
    double tmp = *first;
    *first = *last;
    *last = tmp;
    ++first;
  }
}
```

Question **24**

Correct

2.00 points out of 2.00

Define a structure type with structure tag **pos2d_tag** that has members named **x** and **y**, both of type **float**.

Your answer must not use the storage specifier **typedef**.

```
// NOTE: Including standard library functions will prevent your code from compiling!!!
// NOTE: Using storage specifier typedef will prevent your code from compiling!!!
// Provide your answer here.
struct pos2d_tag
{
    float x;
    float y;
};
```

Correct

Evaluation details:

```
Evaluation:
-Summary of tests
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
struct pos2d_tag {
  float x, y;
};
```

Question **25**

Correct

2.00 points out of 2.00

Define a structure type with structure tag **pos2d_tag** that has members named **x** and **y**, both of type **float**. Continue the definition statement [where you are defining the structure type] by defining and initializing two variables named **start** and **end** with Cartesian coordinates $(1.1, 2.2)$ and $(4.6, 8.7)$, respectively.

Your answer must not use the storage specifier **typedef**.

```
// NOTE: Including standard library functions will prevent your code from compiling!!!
// NOTE: Using storage specifier typedef will prevent your code from compiling!!!
// Provide your answer here.
struct pos2d_tag
{
    float x;
    float y;
};
struct pos2d_tag start ={1.1, 2.2};
struct pos2d_tag end ={4.6, 8.7};
```

Correct

Evaluation details:

```
Evaluation:
-Summary of tests
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
struct pos2d_tag {
  float x, y;
} start = {1.1f, 2.2f}, end = {4.6f, 8.7f};
```

Question **26**

Correct

4.00 points out of 4.00

First, define a structure type with structure tag **`pos2d_tag`** that has members named **`x`** and **`y`**, both of type **`float`**.

Next, define a function named **`set_position`** that takes two parameters both of type **`float`** and returns an object of the structure type defined in the first step whose **`x`** and **`y`** members are initialized by the function's first and second parameter, respectively.

Your answer must not use storage specifier **`typedef`**.

```
// NOTE: Including standard library functions will prevent your code from compiling!!!
// NOTE: Using storage specifier typedef will prevent your code from compiling!!!
// Provide your answer here.
struct pos2d_tag
{
    float x;
    float y;
};

struct pos2d_tag set_position(float x, float y)
{
    struct pos2d_tag h = {x, y};
    return h;
}
```

Correct

Evaluation details:

```
Evaluation:
-Summary of tests
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
struct pos2d_tag {
  float x, y;
};

struct pos2d_tag set_position(float x, float y) {
  struct pos2d_tag tmp = {x, y};
  return tmp;
}
```

Question **27**

Correct

4.00 points out of 4.00

First, define a structure type with structure tag **pos2d_tag** that has members named **x** and **y**, both of type **float**.

Next, define a function named **mid_point** that computes the midpoint of two points and returns the computed midpoint as a value of the structure type defined earlier. The function takes two parameters both of type "pointer to read-only structure type defined earlier".

Your answer must not use storage specifier **typedef**.

```
// NOTE: Including standard library functions will prevent your code from compiling!!!
// NOTE: Using storage specifier typedef will prevent your code from compiling!!!
// Provide your answer here.
struct pos2d_tag
{
    float x;
    float y;
};

struct pos2d_tag mid_point(const struct pos2d_tag* one, const struct pos2d_tag* two)
{
    struct pos2d_tag h;
    h.x = (one->x + two->x)/2;
    h.y = (one->y + two->y)/2;
```

Correct

Evaluation details:

Evaluation:

-Summary of tests

```
>+------------------------------+
>|  1  test run/ 1  test passed |
>+------------------------------+
```

```
struct pos2d_tag {
  float x, y;
};

struct pos2d_tag mid_point(struct pos2d_tag const *first, struct pos2d_tag const *last) {
  struct pos2d_tag tmp = {(first->x + last->x)/2.f, (first->y + last->y)/2.f};
  return tmp;
}
```