

Instructions:

1. Read through the entire examination first and determine the order in which you want to answer them.
2. The exam will be graded out of 48.
3. Turn off all electronic media and store them under your desk.
4. If you are asked to author code, it must be implemented using C++. Unless we specifically ask for syntactically correct C++ code, we may ignore one or two small syntactical errors such as a missing semi colon.
5. You may use `cout` and `endl` instead of `std::cout` and `std::endl`.
6. At all times, you can declare and define helper functions to help simplify or clarify the solution you create.
7. If you need more room for one question, you can continue your solution below your solution to Question 11, but indicate that the solution continues there.
8. You may ask only one question during the examination: “May I go to the washroom?”
9. Asking any other question will result in a deduction of 5 marks from the exam grade.
10. If you think a question is ambiguous, write down your assumptions and continue.
11. Do not leave during first hour or after there are only 15 minutes left.
12. Do not stand up until all exams have been picked up.
13. If a question only asks for an answer, you do not have to show your work to get full marks; however, if your answer is wrong and no rough work is presented to show your steps, no part marks will be awarded.

1. (3 points) The *cardinal sine* function is defined as $\frac{\sin(x)}{x}$ for all non-zero x and equal to 1 if $x = 0$. However, you read that using `double` means that you cannot use this ratio for very small values of x and that the cardinal sine function can be approximated by $1 - \frac{x^2}{3!} + \frac{x^4}{5!}$ if $|x| < 10^{-8}$ or `1e-8`. Write a function that implements

$$\text{sinc}(x) = \begin{cases} \frac{\sin(x)}{x} & |x| \geq 10^{-8} \\ 1 - \frac{x^2}{3!} + \frac{x^4}{5!} & |x| < 10^{-8} \end{cases}$$

You may define a helper function if you wish, or you may use the observation that $\text{sinc}(-x) = \text{sinc}(x)$, if you wish. For $|x| \geq 10^{-8}$, you can use `double std::sin( double x )`, which is implemented in the `cmath` library.

```
#include <cmath>
double sinc( double x );

double sinc( double x ) {
```

2. (3 points) Write a function that returns the largest integer k such that 2^k divides a given unsigned integer $n > 0$ exactly. In other words, 2^k is a factor of n , but 2^{k+1} is not. For example:

- If $n = 40$, then the result is 3 as $40 = 2^3 \cdot 5$.
- If $n = 128$, then $k = 7$ as $128 = 2^7$.
- If $n = 339$, then $k = 0$ as $339 = 2^0 \cdot 3 \cdot 113$.

Your code must work for all unsigned integers, including those greater than 2^{31} . You may not call any standard library functions with the exception of `assert(...)`.

Implement the following function:

```
#include <cassert>
unsigned int power_of_two_factor( unsigned int n );

unsigned int power_of_two_factor( unsigned int n ) {
    assert( n > 0 );
```

3. (4 points) Write a program that prints to the console the scientific notation of a number n with exactly one digit to the left of the decimal point and the appropriate power of ten. Trailing zeros should be removed. For example, if the value of n is 123400, the output must be $123400 = 1.234e5$. If the value of n is 5914, the output must be $5914 = 5.914e3$. If the number contains only a single leading non-zero digit, do not print the decimal point, so $500 = 5e2$ and $7 = 7e0$.

You may use `cout` and `endl` without prefixing them by `std::`, and you may **not** call any other function than ones that you define on this page.

Your solution need not be efficient in any way, but it must be a correctly formed function.

```
int main() {
    unsigned int n{};
    std::cout << "Enter a non-negative integer: ";
    std::cin  >> n;

    // Print the integer as described above...
```

4. (4 points) Write a function that takes an array of double-precision floating-point numbers and its capacity as arguments. The function should rescale the entries so that the smallest element becomes 0.0, the largest becomes 1.0, and all other elements are linearly scaled between these two extremes.

Formally, if the original array contains values x_k for $k = 0, \dots, n - 1$, let

$$x_{\min} = \min\{x_0, x_1, \dots, x_{n-1}\} \text{ and } x_{\max} = \max\{x_0, x_1, \dots, x_{n-1}\}$$

Then replace each entry by

$$x_k \leftarrow \frac{x_k - x_{\min}}{x_{\max} - x_{\min}}$$

so that the smallest value becomes 0.0 and the largest becomes 1.0.

If all values are the same (that is, $x_{\max} = x_{\min}$), then set every entry to 0.0.

For example, if the array originally contains

7.0, 5.0, 11.0, 9.0

then after calling `normalize_array` the array should contain

0.3333333, 0.0, 1.0, 0.6666667

because the smallest value is 5.0 and the largest is 11.0.

Implement the function:

```
void normalize_array( double array[], unsigned int cap ) {
```

5. (4 points) Implement the function

```
void draw_diagonal_pattern( int m, int n );
```

which prints an $m \times n$ character grid using only the characters space ' ', 'x', and 'o', according to the following rule for the entry in row i and column j (zero-based indices):

- 'x' if $i = j$ (that is, on the main diagonal).
- 'o' if $i < j$ (that is, above the main diagonal).
- '.' if $i > j$ (that is, below the main diagonal).

Print rows from $i = 0$ to $i = m - 1$; within each row print columns from $j = 0$ to $j = n - 1$, then end the line with `endl`. When $m \leq 0$ or $n \leq 0$, print nothing.

You may use `cout` and `endl` without the `std::` prefix, and you may define helper functions if you wish.

Examples. For $m = 1$ and $n = 1$, the output is:

```
x
```

For $m = 3$ and $n = 5$, the output is:

```
xoooo
.oxxx
..xoo
```

For $m = 5$ and $n = 3$, the output is:

```
xoo
.xo
..x
...
...
```

```
void draw_diagonal_pattern( int m, int n ) {
```

6. (2 points) Describe what this function tries to accomplish. You may assume that `unsigned int f( unsigned int n )` is defined elsewhere. Your description should not just repeat what the statements do, but rather, your description must help the reader understand the intent of the function.

```
void g( unsigned int n ) {
    unsigned int f0{ f( 0 ) };

    unsigned int m{ 1 };

    while ( f( m ) != f0 ) {
        ++m;
    }

    std::cout << f0;

    for ( int k{ 1 }; k < m; ++k ) {
        std::cout << ", " << f( k );
    }

    std::cout << std::endl;
}
```

7. (5 points) Draw the call stack the first time the program gets to the comment that says `// Draw the call stack....`. Your call stack need only identify which blocks of memory are used for which function, what are the parameters and local variables on the call stack, together with their values. **Note, you don't have to write down addresses, and you don't have to show the intermediate call stacks, so for example, if a call to *f* does not result in a call to *g*, just do the computation and use the return value.** If you have multiple call stacks, be sure to identify which is your solution.

```
int main();
bool f( int n );
int g( int a, int b );

int main() {
    int data[5]{ 5, 13, 91, 17, 29 };

    for ( unsigned int k{ 0 }; k < 5; ++k ) {
        if ( f( data[k] ) ) {
            std::cout << data[k] << std::endl;
        }
    }

    return 0;
}

bool f( int n ) {
    if ( n < 2 ) {
        return false;
    } else if ( n == 2 ) {
        return true;
    }

    bool even{ n%2 == 0 };
    if ( even ) {
        return false;
    } else {
        for ( int m{ 3 }; m < n; m += 2 ) {
            if ( g( n, m ) != 1 ) {
                return false;
            }
        }
        return true;
    }
}

int g( int a, int b ) {
    assert( ( a >= 0 ) && ( b >= 0 ) );
    assert( ( a > 0 ) || ( b > 0 ) );

    int rem{ a%b };

    while ( rem != 0 ) {
        a = b;
        b = rem;
        rem = a%b;
    }

    // Draw the call stack the first time 'b' is not equal to 1
    // - Draw it before the return statement is executed...
    // - This *will* be true at least once for this program.
    return b;
}
```

8. (3 points) If m and n are both unsigned short, so 16 bits, we calculate $m - n$ by calculating $m + n2s$ where $n2s$ is the two's complement of n . Using the binary representation of 4, 5 and 6, show that $6 - 5$ does in fact correctly calculate the result 1 while $4 - 5$ equals the largest possible unsigned short.

9. (5 points) Recall that the least significant bit of an `unsigned int` is Bit 0, and the most significant bit is Bit 31.

Write a function that takes two arguments `unsigned int m` and `unsigned int n`, and sets all bits from Bit m to Bit $n - 1$ of the parameter `value` to the bit passed as a fourth argument `unsigned int bit`, which is either 0 or 1.

Recall that `n |= 4` sets Bit 2 to 1 and that `n &= ~4` sets Bit 2 to 0.

```
void set( unsigned int &value, unsigned int m, unsigned int n, unsigned int bit ) {  
    assert( (bit == 0) || (bit == 1) );
```

10. (4 points) The array with entries {m0, n0, m1, n1, m2, n2, ..., mL} represents the sequence

$$\underbrace{m_0, \dots, m_0}_{n_0 \text{ times}}, \underbrace{m_1, \dots, m_1}_{n_1 \text{ times}}, \dots, \underbrace{m_{L-1}, \dots, m_{L-1}}_{n_{L-1} \text{ times}}, m_L, m_L, m_L, \dots$$

For example, the array { 3, 5, 2, 4, 7 } represents

$$3, 3, 3, 3, 3, 2, 2, 2, 2, 7, 7, 7, \dots,$$

where the last value is repeated. You should return the entry at index k of this sequence, where the first entry is with $k = 0$. Thus, entries $k = 0$ through $k = 4$ are 3, entries $k = 5$ through $k = 8$ are 2, and all entries after and including $k = 9$ are 7.

You should assert that the capacity of the array is odd.

```
unsigned int index( unsigned int k, unsigned int array[], unsigned int cap ) {
```

11. (3 points) Are any of the following six conditions equivalent: that is, one is true if and only if the other is true. Write your answers as pairs of integers for those that are equivalent, and for those that are not equivalent to any others, list them separately. Assume that x and y are of type `int`.

1. `((x > 0) && (x < 10)) || ((y > 0) && (y < 10))`
2. `(x >= 0) && (y >= 0)`
3. `(x < 0) || (x > 10) || (y < 0) || (y > 10)`
4. `!((x >= 0) && (x <= 10) && (y >= 0) && (y <= 10))`
5. `!((x < 0) || (y < 0))`
6. `!(((x >= 10) && (y >= 10)) || ((x >= 10) && (y <= 0)) || ((x <= 0) && (y >= 10)) || ((x <= 0) && (y <= 0)))`

Enter your answers here:

1. These pairs are equivalent:
2. These are not equivalent to any of the others:

12. (3 points) Write one function that goes through a C-style string and replaces any tab character ('`\t`'), new line character ('`\n`') and carriage return character ('`\r`') with a blank space ('').

```
void clean( char str[] ) {
```

13. (4 points) Write a function that takes three local variables as arguments and reassigns them so that the first argument becomes assigned the smallest of the three, the second is assigned the middle of the three, and the third is assigned the largest of the three. Give your function a reasonable name. As for all questions on this examination, you may author a helper function, but in this case, it could be very beneficial to do so.

```
int main() {
    int r{ 45 }, s{ 5 }, t{ 21 };
    std::cout << r << " " << s << " " << t << std::endl; // prints 45 5 21
    your_function( r, s, t );
    std::cout << r << " " << s << " " << t << std::endl; // prints 5 21 45
    return 0;
}
```

Hint: Swapping when necessary, compare the first two, then the second two, and then compare the first two again.

14. (1 point) What is the output when this program is compiled and executed?

```
#include <iostream>

int main();
int f( int a, int b = 1, int c = 2 );

int main() {
    f( 1, 2, 3 );
    f( 4, 5 );
    f( 6 );
    return 0;
}

int f( int a, int b, int c ) {
    std::cout << a << " " << b << " " << c << std::endl;
}
```

This blank page is for extra room. If you continue a solution on this page, be sure to indicate that your solution continues on the last page.