Exam 1

● Graded

Student

Zhangrui Weng

Total Points

84 / 100 pts

Question 1

Q1 **35** / 50 pts

1.1 Q1.1 **5** / 5 pts

✓ **+ 5 pts** Correct

+ 0 pts Incorrect

1.2 Q1.2 **0** / 5 pts

+ 5 pts Correct

✓ **+ 0 pts** Incorrect

1.3 Q1.3 **0** / 5 pts

+ 5 pts Correct

✓ **+ 0 pts** Incorrect

1.4 Q1.4 **5** / 5 pts

✓ **+ 5 pts** Correct

+ 0 pts Incorrect

1.5 Q1.5 **5** / 5 pts

✓ **+ 5 pts** Correct

+ 0 pts Incorrect

1.6 Q1.6 **0** / 5 pts

+ 5 pts Correct

✓ **+ 0 pts** Incorrect

1.7 Q1.7 **5** / 5 pts

✓ **+ 5 pts** Correct

+ 0 pts Incorrect

1.8 Q1.8 **5** / 5 pts

✓ **+ 5 pts** Correct

+ 0 pts Incorrect

1.9 Q1.9 **5** / 5 pts

✓ **+ 5 pts** Correct

+ 0 pts Incorrect

1.10 **Q1.10** **5** / 5 pts

✓ **+ 5 pts** Correct

+ 0 pts Incorrect

Question 2

Q2

49 / 50 pts

High level

✓ **+ 2 pts** Function signature is unchanged.

✓ **+ 2 pts** Only relies on `<stdexcept>` and possibly `<string>` if used to get **Partial Credit**.

✓ **+ 2 pts** Only uses primitive data types (aside from `std::invalid_argument`).

+ 1 pt Partial Credit: Only uses primitive data types except for an aggregate data type.

+ 0 pts No evidence.

Initializing

✓ **+ 2 pts** All variables initialized to a specific value before being read.

+ 1 pt Partial Credit: At least one variable initialized but others not initialized.

+ 0 pts No evidence.

Parameters and return values

✓ **+ 2 pts** Used values from parameter values. e.g. Did not use standard in (i.e. cin) to get any values.

✓ **+ 2 pts** Returns a long long with a calculated value (even if the value is not correct)

+ 1 pt Partial Credit: returns another integer type

+ 0 pts Hard coded answers from test cases.

+ 0 pts Got values from source other than the parameters.

+ 0 pts No evidence.

Error Handling

✓ **+ 4 pts** Correctly throws a std::invalid_argument if and only the input is negative

+ 3 pts Partial Credit Attempt to throw an exception but there is a problem (exception is thrown for some valid inputs OR exception is not throw for some invalid inputs OR wrong type is thrown)

+ 2 pts Partial Credit Attempt to throw an exception but there are problems (exception is thrown for some valid inputs OR exception is not throw for some invalid inputs OR wrong type is thrown)

+ 1 pt Partial Credit Attempted

+ 0 pts No evidence

Iteration

✓ **+ 4 pts** Appropriate structure to iterate through each digit.

✓ **+ 4 pts** Iteration is structured to exit at the appropriate time.

✓ **+ 4 pts** Changes made (e.g. dividing the number by 10) such that the iteration ends at the appropriate time.

+ 3 pts Partial Credit: Appropriate structure to iterate through each digit but with an error.

+ 2 pts Partial Credit: Attempt to iterate through each digit.

+ 3 pts Partial Credit: Appropriate way to exit iteration with an error.

+ 2 pts Partial Credit: Attempt to exit iteration with major problems.

+ 2 pts Partial Credit: Attempt to make changes such that the iteration ends at the appropriate time.

+ 0 pts No evidence.

Digit Extraction

✓ **+ 5 pts** Appropriate way to correctly slice digits from integer. (can use modulus and integer division)

✓ **+ 2 pts** Avoid extracting a digit twice (e.g. update number to no longer contain the extracted digit or update index into string) - all or none

+ 4 pts Partial Credit: Approprite way to slice digits from integer but there is a problem

+ 3 pts Partial Credit: Attempt to slice digits, but there are problems.

+ 2 pts Partial Credit: Appropriate way to extract digits but manually rather than iteratively.

+ 3 pts Partial Credit: Relies on an aggregate datatype (e.g. string) - Appropriate way to correctly slice digits.

+ 2 pts Partial Credit: Relies on an aggregate datatype (e.g. string) - Attempt to slice digits from integer but there are problems.

+ 1 pt Partial Credit: Attempted

+ 0 pts No Evidence

Result Construction

✓ **+ 10 pts** Appropriate way to construct the answer using the extracted digits.

+ 8 pts Partial Credit: Appropriate way to construct the answer with a minor problem (e.g. off by one error or use incorrect digits)

+ 5 pts Partial Credit: Appropriate way to construct the answer with a few problems (e.g. off by one error or use incorrect digits)

+ 5 pts Partial Credit: Relies on an aggregate datatype (e.g. string) - Appropriate way to construct final number.

+ 3 pts Partial Credit: Relies on an aggregate datatype (e.g. string) - Appropriate way to construct final number with a minor problem.

+ 2 pts Partial Credit: Relies on an aggregate datatype (e.g. string) - Appropriate way to construct final number with a few problem.

+ 2 pts Attempted

+ 1 pt Attempted using an aggregate datatype.

+ 0 pts No Evidence

Overall Correctness

+ 5 pts The overall algorithm works to give a correct return value excluding items already accounted for in prior rubric items.

✓ **+ 4 pts** **Partial Credit**: The overall algorithm works but a minor logic error can result in an incorrect return value.

+ 3 pts **Partial Credit**: Overall algorithm structure is in the right direction but details are unclear or does not work.

+ 1 pt **Partial Credit**: Major problems with the algorithm, but there are some elements that could be part of a correct solution.

+ 0 pts Not at all in the right direction

Partial Credit: Attempted and less than 5 points earned.

+ 5 pts **Partial Credit**: Attempted and no other points earned.

+ 4 pts **Partial Credit**: Attempted and 1 point earned.

+ 3 pts **Partial Credit**: Attempted and 2 points earned.

+ 2 pts **Partial Credit**: Attempted and 3 points earned.

+ 1 pt **Partial Credit**: Attempted and 4 points earned.

+ 0 pts No Evidence

Exam 1

Version: R0530

Academic Integrity

- Aggies do not lie, cheat, or steal, nor tolerate those who do.
- We hope you have fun solving these problems, even though this is an exam ☺
- You can do this!

Exam Guidelines

- You will have 50 minutes to complete and submit this exam.
- Do not write outside of the margin box. We scan the exams and anything outside those lines will likely get cut off.
- You cannot use any electronic devices (including calculators, phones, smart watches, and computers)
- You may use
    - A writing utensil (e.g. pen/pencil)
    - Scratch Paper that we provide.
        - § Scratch paper may not have the margin box, so leave a margin around the edge of your scratch paper to avoid having information cut off when scanned.
    - Up to 5 pages of exam aids.
        - § Exam aids can be pages up to 8.5X11 inches and can be handwritten or printed on both sides.
        - § Do not use exam aids for scratch work.
            - If you use an exam aid for scratch work, submit with exam.
- Start Exam when prompted.
- Stop exam when you are finished or when time expires.
    - Attach any scratch work to the end of your exam. Including exam aids used for scratch work.
    - Submit your exam.
        - § You may keep your exam aids if they were not used for scratch work.

Fill in name and UIN before exam starts

Name: Zhayu Weng

UIN: 832009830

Name:________________________________

Q1 - Code Tracing (50 points)

Overview

- In the following parts you will be given small pieces of code. Determine what each piece of code prints, and fill in the blank.
- For each piece of code, assume that the file has `#include <iostream>` and `using std::cout;`

Q3.1

x=
y=10
z=20

```
int sum(int x, int ~~y=10~~, int z=20);

int main(){
      cout<< sum(1, 2);
      return 0;
}
int sum(int x, int y, int z){
      return x+y+z;
}
```

This code prints: 23

Q3.2

```
int main() {
      const int SIZE = 3;
      int arry[SIZE] = {2, 0, 1};
      char letters[SIZE] = {'a', 'b', 'c'};

      for (size_t i = 0; i < SIZE; ++i) {
            cout << letters[arry[i]]<< ",";
      }
      return 0;
}
```

This code prints: c, b, a

Q3.3

012345

```
int main(){
      char s[] = "C++IsFun";
      s[5] = '\0';
      cout << s;
}
```

This code prints: C++Isun

Q3.4

```
int main() {
      int number=1;
      switch(number){
            case 1:
            cout << "A";
            case 2:
            cout << "B";
            break;
            default:
            cout << "C";
      }
      return 0;
}
```

number = 1

This code prints: AB

Q3.5

```
int main() {
      size_t width = 2;
      size_t height = 3;
      for (size_t y = 0; y < height; ++y) {
            for (int x = 0; x < width; ++x) {
                  cout << (y * width) + x + 1 << ",";
            }
      }
      return 0;
}
```

width = 2
height = 3

This code prints: 1, 2, 3, 4, 5, 6

y < 3
x < 2

~~0~~
1
2

y = 0 → x < 2
x

y = 1 →

1, 2, 3, 4, 5, 6

1) (0 × 2) + 0 + 1
2) (0 × 2) + 1 + 1
3) (1 * 2) + 0 + 1 = 2 + 1
4) (1 * 2) + 1 + 1 = 2 + 2
5) (2 * 2) + 0 + 1 = 5

5) (2*2) + (1) = 5

6) (2 * 2) + (+) = 6

Name:______________________________

Q3.6

```
int main(){
      int lines=3, i,j;
      ~~i = 1;~~
      do {
         j = 1;
           do{
                ~~cout << j;~~
                ~~j++;~~
           } while(j <= i);
           i++;
      } while(i <= lines);
      return 0;
 }
```

line = 3

i = ~~1~~, 2

line=3

i = ~~1~~ 2

j=1 → cout : 1

j=2

This code prints: 1 2, 3

Q3.7

```
int main() {
      int i=12;
      int j=5;
      cout << i/j;
      return 0;
}
```

12/5 =

This code prints: 2

Q3.8

```
int main() {
      try {
            int w = 90;
            if (w >= 100) {
                  cout << "???";
            } else {
                  throw (w);
            }
      }
      catch (int fit) {
            cout << fit;
      }
      return 0;
}
```

This code prints: 90

Name:____________________________________

Q3.9

```
bool foo(int a, int b);

int main() {
      if(foo(8, 13)){
            cout << "A";
      } else {
            cout << "B";
      }
      return 0;
}

bool foo(int a, int b){
      do {
            if(a == 1 && b == 1){
                  return true;
            }
            int c = a;
            ~~a = b - a;~~
            ~~b = c;~~
      } while(a > 0);
      return false;
}
```

X

a=8 ≠1 b=13≠1

1) c=8;
a =13-8 =5
b =8

2) c=5;
a= 8-5=3
b=5;

3) c=3
a=5-3=2
b=3

4) c=2
a=3-2=1
b=2

5) c=1
a= 2 -1
b=c
1=1

This code prints: A

Q3.10

```
int main() {
      bool a = true;
      bool b = true;
      int i = 10;
      ~~int j = 2;~~
      ~~a = i % j == 1;~~
      if(a != b) {
            cout << i/j;
      } else {
            cout << i--;
      }
      return 0;
}
```

a: ~~true~~ false
b: true
i=10;
j=2;

10%2 = =

10/2 since a is false

This code prints: 5

Name:_______________________________

Q2 - Combining Digit Pairs (50 points)

Overview

Given a positive integer, calculate the product of adding adjacent digits of an integer with each other.

Requirements

Write the function `long long combineDigitPairs(long long number)` which computes the sum of each pair of adjacent digits in number and then returns the product of these sums.

- If the input is negative, your program should throw an instance of std::invalid_argument
- Note that long long can have up to 19 digits.
- long long is a signed integer with a max value of 9,223,372,036,854,775,807
- You may assume that the answer fits in a long long (that is, you do not need to worry about overflow)
- You may not use any libraries besides stdexcept
- For full credit, you may not use any type of string or array as part of your solution.

Examples

- `combineDigitPairs(99)` should return 18, because 9+9 = 18
- `combineDigitPairs(21469)` should return 2250, because (2+1) * (1+4) * (4+6) * (6+9) = 2250. Note this is the same as (9+6) * (6+4) * (4+1) * (1+2).
- `combineDigitPairs(320440)` should return 1280, because (3+2) * (2+0) * (0+4) * (4+4) * (4+0) = 1280. Note this is the same as (0+4) * (4+4) * (4+0) * (0+2) * (2+3)
- `combineDigitPairs(-123)` should throw an instance of `std::invalid_argument`

Name:_______________________________ **7 / 8**

Q2 Answer (Combining Digit Pairs)

```
#include <stdexcept>
long long combineDigitPairs(long long number)
   if(number < 0) {
    std::throw invalid_argument("Negative numbers
   are not allowed"));
  }
   long pair;
   long sum =0;
   long factor =10;
  while( number> 9) { // More than two digits
     pair = number % 100; // get the last two dig
     sum += (pair/10) * (pair%10);
     number /=100;
  }
  if (~~number >0)~~ (number> 0)
  {
      sum += num;
   }
  return sum;
}
```

/pair/10 remove the last digit

Name:______________________________