Student ID: _____ Lab Section: _____

This test is closed book, closed notes. Points for each question are shown inside [ ] brackets at the beginning of each question. You should assume that, for all quoted program segments, the appropriate header files are included and the standard namespace is used.

Remember to write out and sign the pledge at the end of the test.

1. [1] The parameters in a function invocation are called the _____ parameters. The parameters are represented in the invoked function by the _____ parameters.
2. [1] The values of the formal parameters and other objects defined in a function are kept in its _____.
3. [1] A description of a function's interface, which is a valid C++ statement, is called a _____
4. [1] A _____ object is an object defined outside of any function interface, or any other block structure.
5. [1] C++ supports two forms of parameter passing: _____ and _____.
6. [1] A _____ modifier applied to a parameter declaration indicates that the function may not change the object.
7. [1] _____ parameters can be specified as trailing parameters only.
8. [1] When a function name or an operator symbol can cause diferent actions, depending on its parameters or operands, it is said to be _____.
9. [1] A _____ constructor is a constructor that requires no parameters.

10. [5] Suppose we have a function that is supposed to double the value of an integer variable, and the resulting value is to be used in the invoking expression.

The body of the function consists **only** of the following:

```
{
    value = value * 2;
}
```

Circle all of the following which would make sense as the function prototype.

```
void double(int &value);

int twice(int value);

void int by_two(int &value);

int value();

void double_val(int &value);
```

11. [6] What is the output of the following program segment?

```
int test(int x)
{
     if (x > 3)
          return 1;
     else
          return 0;
}

int main()
{
     int y = 6;
     while (test(y + 1) == 1)
     {
          cout << y << " ";
          y = y - 1;
     }
     cout << endl;
     return 0;
}
```

_____

12. [5] Consider the following function `curse`:

```
int curse(int n) {
   if (n <= 0) {
      return 1;
   }
   else {
      return n * curse(n-2);
   }
}
```

What is the output of the following program fragment?

```
cout << curse(5) << endl;
```

_____

13. [9] What is the output of the following program?

```
void nasty(int &x, int y, int &z);
int main()
{
     int a = 10;
     int b = 20;
     int c = 30;
     nasty(a, b, c);
     cout << a << " " << b << " " << c << endl;
     return 0;
}

void nasty(int &x, int y, int &z)
{
     cout << x << " " << y << " " << z << endl;
     x = 1;
     y = 2;
     z = 3;
     cout << x << " " << y << " " << z << endl;
}
```

_____

_____

_____

14. [2] What output will be produced by the following code, assuming it is embedded in a complete program?

```
int first_choice = 1;
switch(first_choice + 1)
{
     case 1:    cout << "Roast Beef\n";
                break;
     case 2:    cout << "Roast Worms\n";
                break;
     case 3:    cout << "Roast Ice Cream\n";
                break;
     default:   cout << "Eat at Wendy's\n";
}
```

_____

15. [2] What output will be produced by the code above if the first line were replaced by:

```
int first_choice = 3;
```

_____

16. [9] What does the following program segment print?

```
void func();

int i = 2;
int j = 3;
int k = 4;

int m = 5;
int main()
{
     cout << i + j << " " << k << " " << m << endl;
     func();
     cout << i + j << " " << k << " " << m << endl;
     return 0;
}


void func()
{
     int m = i + j;
     j = i + m;
     k = m * m;
}
```

_____

_____

17. [4] What is the output of the following?

```
void one(int &x) {
   x = x + 1;
}

void two(int x) {
   x = x + 2;
}

int main() {
   int x = 3;
   one(x);
   cout << x << " ";
   two(x);
   cout << x << endl;
}
```

_____

The following code will be used in the next 3 questions. In each of these questions show the output produced when the code is executed. If the code segment will not compile, briefly state the reason for this.

Here is the definition of a class of rectangular objects - the class `Rect` (don't confuse this class with class `RectangleShape` from the textbook):

```
class Rect
{
     public:
          Rect();
          Rect( int j, int k);
          int get_l();
          int get_w();
          int area();
     private:
          int length;
          int width;

};

// now the definitions of the member functions

Rect::Rect() {
     length = -234;
     width = 0;
}

Rect::Rect(int j, int k) {
     length = j;
     width = k;
}

int Rect::get_l() {
     return length;
}

int Rect::get_w() {
     return width;
}

int Rect::area()
{
     return (length * width);
}
```

18. [3] Consider the following main function:

```
int main()
{
     Rect a(4, 7);
     cout << a.length << endl;
     return 0;
}
```

_____

19. [4] Here is another main function:

```
int main()
{
     Rect box(6, 9);
     int x = box.area();
     cout << x << " " << box.get_l() << endl;
     return 0;
}
```

_____

20. [4] And here is a third:

```
int main()
{
     Rect alpha;
     Rect beta(1, 2);
     int one = alpha.get_l() + beta.get_l();
     int two = alpha.get_w() + beta.get_w();
     cout << one << " " << two << endl;
     return 0;
}
```

_____

The definitions in the following header file, named `rational.h`, are in effect for the remaining questions

```
#ifndef RATIONAL_H
#define RATIONAL_H
class Rational {
   public: // member functions and operator
      Rational();
      Rational(const int numer, const int denom);
protected:
      int GetNumerator() const;
      int GetDenominator() const;
      void SetNumerator(const int numer);
      void SetDenominator(const int denom);
   private:
      int Numerator;
      int Denominator;
};
Rational operator+(const Rational &r, const Rational &s);
Rational operator*(const Rational &r, const Rational &s);
bool operator!=(const Rational &r, const Rational &s);
ostream& operator<<(ostream &sout, const Rational &r);
istream& operator>>(istream &sin, Rational &r);
#endif
```

21. [3] In no more than two short sentences, explain why the `#ifndef`, `#define`, and `#endif` directives appear in the include file. Don't explain how they work

22. [2] The **`const`** in the prototypes of functions `GetNumerator()` and `GetDenominator()` in class `Rational` indicates that:

    (a) The functions will not modify their actual parameters.

    (b) The functions will return a constant value.

    (c) The functions will not modify the invoking object.

    (d) None of the above.

23. [2] The **`const`** in the prototypes of functions `SetNumerator()` and `SetDenominator()` in class `Rational` indicates that:

    (a) The functions will not modify their actual parameters.

    (b) The functions will return a constant value.

    (c) The functions will not modify the invoking object.

    (d) None of the above.

24. [5] The code below defines the overloading of operator `!=` so that it returns **`true`** if and only if the rational numbers represented by its two operands are not equivalent. Otherwise it is to return the value **`false`**.

```
bool operator!=(const Rational &r, const Rational &s) {
  int a = r.GetNumerator();
  int b = r.GetDenominator();
  int c = s.GetNumerator();
  int d = s.GetDenominator();
  return X;
}
```

Write here the code to replace X which will make this function perform correctly:
(Note that 3/6 is equivalent to 1/2)

25. [4] The overloading of `!=` in the question above defines that its its parameters are to be **`const`** `Rational &.` In one sentence each, explain:

(a) the parameter is a reference (&) parameter. Why do we pass non-fundamental objects by reference instead of by value?

(b) why, in addition, is it made a **`const`** parameter?

26. [10] Using the functions whose interfaces are provided in the include file `rational.h`, write a complete member function `Reciprocal` that returns the `Rational` reciprocal of the invoking object. Recall that for a `Rational` object, the reciprocal of `a/b` is `b/a`. Do *not* compute the reciprocal using the division operation! Function `Reciprocal` is to be a member function. Write your function below.

27. [12] Using the class `Rational` as defined, together with the auxilliary functions, write a **complete program** that computes the sum of the rational numbers `1/2` to `1/`*n* where *n* is an integer that is prompted for and read by the program. Your program should use an assert statement to ensure that the input typed by the user is greater than 1 and less than 51. Assume that the `<<` operator, defined as an auxilliary function, works as described in the text. The final output of the program should be the line:

The sum of 1/2 through 1/n is **X**

where `n` is the value inputted from the keyboard and **X** is the computed sum.

Your program will be graded based on its correctness and style. Do not include comments in your program.

Write your program on the **next page**. We have included all necessary header files. Also, remember to write out and sign the pledge at the bottom of the next page.

Write your code here:

```
#include <iostream>
#include <assert.h>
#include "rational.h"
using namespace std;
int main( )
{




}
```

Write out and sign the Pledge here: