CMPT-101 Sample Final Exam Answer Key
3 hours

This is a **closed book exam**. No calculators or similar aids are allowed**. Read each question carefully before answering it**, and list any important assumptions you make. Please ask for clarification if you're unsure of the wording of a question.

Name

Question	Out of	Mark
Short Answer	/22	
Boolean Expressions	/11	
Adding Up Numbers	/16	
What Does this Function Do?	/6	
Leap Years	/8	
Streams and Files	/15	
ADTs and Overloading	/22	
Total	**/100**	

Disclaimer

This exam is from a previous offering of CMPT-101. Your final exam could emphasize slightly different material, pose questions in different formats, or be harder or easier. There could be questions on this exam that assume you know about topics your course didn't cover (e.g. destructors, operator overloading); your final exam will only cover topics from your offering of the course.

Short Answer

a) (2 marks) What is the difference between a class and an object in C++?

An object is an instance of a class. In C++, a class is a data type, while an object is a value of that class.

b) (2 marks) What is the input to a linker? What does it output?

Input: object code (in `.obj` files)
Output: executable code (a `.exe` file)

c) (1 marks) What does ADT stand for? Put your answer in the box.

Abstract Data Type

d) (1 mark) What, exactly, is the maximum `unsigned int` in Borland C++? Put your answer in the box. (Hint: write a simple arithmetic expression)

Any one of the following answers is worth one mark:

- $-1 + 2^{32}$ (`unsigned int`s are 32 bits long, so its range is 0, 1, ..., $-1 + 2^{32}$; 2^{32} alone receives 0 marks)
- `0-1` (if 0 is an `unsigned int`, then subtracting 1 will usually make it “wrap around” to the largest value in its range)
- 11111111111111111111111111111111 (32 bits!)

e) (8 marks) Fill in the table below. Write “unknown” if it is not possible to tell what will be printed on the screen (i.e. because the result is implementation dependent).

```
int x = 5;
int* y = new int(3);
int** z = &y;
int A[5] = {1,2,3,4,5};
```

Code	Printed on `cout`
`cout << *y;`	3
`cout << **z;`	3
`cout << *&x;`	5
`cout << A[4];`	5
`cout << *(A+2);`	3
`cout << *(A+*y);`	4
`cout << A[**z];`	4
`cout << A[x];`	Junk, since `x` is `5`, which is not in A’s range

f) (4 marks) What do the following `cout` statements print? Each row of the table represents a line of code in the same program, so if `i` changes in one row, you should use that new value in the next row(s).

```
int i = 0;
```

Code	Printed on `cout`
`cout << ++i;`	1
`cout << i++;`	1
`cout << i;`	2
`cout << (i=-1);`	-1

g) (4 marks) Consider this class:

```
#include <string>
using namespace std;

class HelloGoodbye {

  string name_;

  HelloGoodbye(const string& s)
  {
    name_ = s;
    cout << "Hi " << name_ << "!\n";
  }

  ~HelloGoodbye()
  {
    cout << "Bye " << name_ << "!\n";
  }

};
```

What will the following code print to `cout`?

Code	Printed on `cout`

`void f()`
`{`
`  HelloGoodbye A("A");`
`  {`
`    HelloGoodbye B("B");`
`  }`
`}`
	`Hi A!`
	`Hi B!`
	`Bye B!`
	`Bye A!`

Boolean Expressions

(11 marks) Suppose the `yes()` and `no()` functions are defined like this:

```
bool yes()                bool no()
{                         {
  cout << "yes ";           cout << "no ";
  return true;              return false;
}                         }
```

Note that whenever one of these functions is called, it prints a message to `cout`.

Boolean expression	Value of expression	Printed on `cout`
	(circle true or false)	
`no() && no()`	true [✓ false]	no
`no() && yes()`	true [✓ false]	no
`yes() && no()`	true [✓ false]	yes no
`yes() && yes()`	[✓ true] false	yes yes
`no() || no()`	true [✓ false]	no no
`no() || yes()`	[✓ true] false	no yes
`yes() || no()`	[✓ true] false	yes
`yes() || yes()`	[✓ true] false	yes
`!(no() && (yes() || !no()))`	[✓ true] false	no
`(no() || !!(true && !no()))`	[✓ true] false	no no
`-1 && 0 && 1 && 2`	true [✓ false]	
`(1 == (-1 + 2)) == (41 && true)`	[✓ true] false	

Adding Up Numbers

(16 marks) Here is one way of calculating 1+2+...+n:

```
// Pre-condition: n is 1 or greater
// Post-condition: this functions returns the value of
//                 1+2+3+...+n
// Examples: sum(1) is 1, sum(2) is 3, sum(3) is 6, ...
int sum(int n)
{
  int sum = 0;
  while (n > 0) {
    sum = sum + n;
    --n;
  }
  return sum;
}
```

Each question below will use the same header, so all you need to write is the code that would appear in the function body. Do *not* change the function header, or the behavior of the function.

a) (4 marks) Rewrite the body of the `sum` function using a single *for-loop* instead of a while-loop.

```
int sum = 0;
for(; n>0; --n)
  sum = sum + n;
return sum;
```

```
int sum = 0;
for(int i = 1; i<=n; ++i)
  sum = sum + i;
return sum;
```

```
int sum;
for(sum = 0; n>0; --n)
  sum = sum + n;
return sum;
```

b) (4 marks) Rewrite the body of the `sum` function using a single *do-while* loop instead of a while-loop.

```
int sum = 0;
do {
  sum = sum + n;
  --n;
} while(n > 0);
return sum;
```

c) (5 marks) Rewrite the body of the `sum` function using *recursion* instead of a while-loop.

```
int sum(int n)
{
  if (n == 1) {
    return 1;
  } else {
    return n + sum(n - 1);
  }
}
```

d) (3 marks) Rewrite the body of the `sum` function using *no* loops or recursion.

```
int sum(int n)
{
  return n*(n+1)/2;
}
```

What Does this Function Do?

```
int F(unsigned int m, unsigned int n)
{
  int sum = 0;
  if (odd(m))   // odd(m) is true if and only if m is an odd number
    sum = n;

  while (m > 1) {
    m = m/2;
    n = 2*n;
    if (odd(m))
      sum += n;
  } // while
  return sum;
}
```

(4 marks) Fill in this table:

Code	Printed on `cout`
`cout << F(4,7);`	28
`cout << F(7,4);`	28
`cout << F(8,2);`	16
`cout << F(3,8);`	24

(2 marks) In English, briefly describe what `F(m,n)` calculates.

`F(m,n)` calculates `m*n`. This is algorithm is called “Russian Peasant Multiplication”.

Leap Years

(8 marks) Write a function that returns `true` if and only if the `unsigned int` Y represents a leap-year. There are two rules for determining leap years (Y is a *century-year* if it’s evenly divisible by 100):

> **Rule 1** If Y is a century-year, then Y is a leap-year just if it’s evenly divisible by 400.
>
> For example, century-years 1700, 1900, 1800, and 3400 *are not* leap-years since they're not divisible by 400. But the century-years 1600, 2000, and 2400 *are* leap years because they are evenly divisible by 400.
>
> **Rule 2** If Y is not a century-year, then it's a leap year if it's evenly divisible by 4.

You can assume that Y is greater than 0. In C++, the `%` operator calculates the remainder when you divide two `int`s. Thus, Y is a century year when `(Y % 100)==0`, and if `(Y % 4)==0`, then Y is evenly divisible by 4 (in general, if `(Y % m)==0`, then Y is evenly divisible by m).

```
bool is_leap_year(int Y)
{
  if ((Y % 100) == 0) {  // Y is a century year
    return (Y % 400) == 0;
  } else {
    return (Y % 4) == 0;
  }
}
```

Streams and Files

(15 marks) Write a complete C++ program that counts the number of times the character `’e’` appears in the text file `story.txt`, and prints that number to `cout`. You should write all the necessary code in `main`; do not define any of your own functions. Make sure to include the necessary `.h` files, and check that `story.txt` has been opened successfully.

```
#include <iostream>
#include <fstream.h>

int main()
{
   ifstream fin("story.txt");

   if (fin.fail()) {
     cout << "Error opening file\n";
     exit(1);
   } // if

   int e_count = 0;
   char next;
   fin.get(next);
   while (!fin.eof()) {
     if (next == ’e’)
       ++e_count;
     fin.get(next);
   } // while
   cout << "e_count = "
        << e_count << ’\n’;
}
```

ADTs and Overloading

(22 marks) Write a class called `Adder` that stores the sum of all the `int`s given to it. Your `Adder` class should allow you to write the following code (and code like it):

```
// sample code

Adder sum1;       // sum1 is initialized to 0
Adder sum2(2);    // sum2 is initialized to 2

cout << “sum1 is ” << sum1 << ‘\n’;     // prints “sum1 is 0”
cout << “sum2 is ” << sum2 << ‘\n’;     // prints “sum2 is 2”

sum1 += 5;   // adds 5 to sum1; now sum1 is 5
sum2 += -3;  // adds -3 to sum2; now sum2 is -1

if (sum1 == sum2)
  cout << “sum1 and sum2 are the same\n”;
```

You should only write the functions that are necessary for `Adder` to be used as in the above program. Use `const` wherever appropriate, and do *not* write or use a cast operator. Make sure to include any necessary header files.

```
class Adder {
private:
  int sum_;
public:
  Adder() {sum_ = 0;}
  Adder(int n) {sum_ = n;}

  void operator+=(int n) {sum_ += n;}

  friend bool operator==(const Adder& x, const Adder& y);
  friend ostream& operator<<(ostream& ostr, const Adder& x);
}; // class Adder

bool operator==(const Adder& x, const Adder& y)
{
  return (x.sum_ == y.sum_);
}

ostream& operator<<(ostream& ostr, const Adder& x)
{
  return ostr << x.sum_;
}
```

Blank Page

Blank Page