Programming Assignment 3

COMP 206 (T4 - 2020)

Submit a single file named `t3.c0` containing all the solutions, and copy it to the `~/submit` folder to submit it.

Keep the following guidelines with you at all times for future reference. As you gather experience with coding, expand this list with your own observations.

Instructions/Guidelines:

1. Before you start typing your solutions solve the problem using pencil and paper.
2. Write the preconditions and postconditions and decide on the input and output types first.
3. The questions may state some basic pre-conditions and post-conditions, include your own `//@requires`, `//@ensures`, `//@loop_invariant`, and `\\@assert` contracts in your program.
4. If there is a loop, make sure you know the loop invariant before you have written the loop body. It will help you catch bugs as you write the code.
5. To enable contract checking use `coin -d filename.c0`
6. Submit your solutions by copying them to the `~/submit` folder.

Coding Style

1. Give descriptive names to variables.
2. Use comments to explain what you are doing.
    - a single line comment starts with `// single line comment`
    - a multiline comment is enclosed between `/*` and `*/`.

```
int temp;
/* this is a
multiline
comment */
int final;
```

3. Indenting the code properly makes it easier to read and catch bugs.

```
if (a < b) {

   //press TAB key to reach here.
   //Align everything in this block properly.

}//press BACKSPACE to align closing-{ with i of if.

for (int dummy; dummy<10; dummy+=1)
{
    //Even this style of opening and closing parenthesis is good.
}
```

4. Try to keep code as easy to read as possible, by putting extra spaces and empty lines in between, if required.

```
z = f( 10, 10 - y * ( x + z ) / 10 );
```

Introduction to the Tasks

We call an `int` array as a `bitArray` if all its entries as either 0 or 1. A `bitArray B` can be interpreted as a binary number. Use the convention that `B[0]` stores the units place bit (least significant bit) for all the problems below, otherwise you will get wrong answers.

In C_0 or C a new type can be created using the `typedef` keyword. To rename int array to `bitArray` use the following at the beginning of the code outside any function definition.

```
typedef int[] bitArray;
bool funcname(bitArray z) {
    //is it really an array of bits?
}
```

This can make your code readable and easier to debug.

Bitwise operators

Right Shift

If `x` is an int then `x<<1` shifts all the bits in the twos complement representation of `x` by one place to the left and fills a zero as the right-most bit, any overflow bits beyond the 32th place are discarded. To shift `x` by `k` places to the left use the command `x<<k`. For example, `1<<2` equals `4` as $1 = (1)_2$ in twos complement, shifting by 2 places gives us $(100)_2$ whose value is 4.

To find the i^{th} bit, with $1 \leq i \leq 32$, in the C_0 representation of an int `x`, one can use the expression

```
x & (1<<(i-1));
```

Use this information to solve the following.

1. Write a function which takes an int array along with its length as input, and returns true if it is a `bitArray`, false otherwise.

    - Function name: isBinary
    - inputs: int array B, int n
    - returns: boolean
    - requires: length of B must be n.

2. Recall that `int` in C_0 is represented using 32 bits. Write a function which takes a number as input and returns its twos complement representation.

    - Function name: int2tc
    - input argument: int
    - returns: binary number
    - requires: nothing
    - ensures: output array is of size 32, and it is a binary number

3. Write a function which takes a binary number called B which is a 0/1 array and its length n as input. The array B is a twos complement representation of some number, find that number and return it.

    - Function name: tc2int
    - input argument: binary array B, int n
    - returns: int
    - requires: input B must be a binary number with length n.
    - ensures: output int is within the range of an **n-bit** twos complement.

4. Write a function which takes two binary arrays of fixed size `n` and returns their sum, it discards any overflow bits.

    - Function name: addtc
    - input arguments: `bitArray`, `bitArray`, `n` (two binary numbers both having n bits)
    - output: binary array equal to the sum
    - write your own requires and ensures contracts.