Assignment 1

z5163480 Nabil Shaikh z5162498 Bailey Ivancic

May 2, 2018

1 Task 1 : Specify a function emirp

Define a procedure called emirp that given a natural number n>0 it returns the n^{th} emirp.

The precondition can be as simple as n needing to be greater than 0.

Before specifying the postcondition, define what it means to be prime. $x \in \mathbb{N}_{>0}$ is prime iff

$$\mathsf{prime}(x) \triangleq \forall i \in 2..\sqrt{x}\,(i \nmid x)$$

We use the folliwng mathematical theorem when checking primality in the range of $(2..\sqrt{x})$:
If a number n is not prime, it can be factored into two factors, called a and b;

n = a * b

If both a and b are greater than $\sqrt{n}$, The product a * b would be greater than n. Therefore, at least one of these factors must be less than $\sqrt{n}$, and to check if it is prime we only need to find one of the factors. Therefore, we check for primality up to sqrt(n). Similarly, there is no need to check factors 0 or 1, and so we start testing at 2.

Next we define what is means for the reverse of the number x to be prime. Where r is the reverse of x.

$$\mathsf{res}(x) \triangleq x = \sum_{i=0}^{c(x)}(S_i 10^{c(x)-i}) \wedge r = \sum_{i=0}^{c(x)}(S_i 10^{i}) \wedge \mathsf{prime}(r)$$

Where $c(x) = \lfloor log_{10} x \rfloor$ and S is a sequence containing the digits of x from left to right. This states that there x is made up of digits in S and that there exists an r such that r is made up of reversing the digit order in the sequence S. Furthermore, r is prime and $r \neq n$.

Now, define what it means for x to be the n^{th} emirp.

$$\mathsf{prime}(x) \wedge \mathsf{res}(x) \wedge r \neq x \wedge n = \sum_{\forall j \in \mathbb{A}} (1)$$

Where $\mathbb{A}$ is defined globally as

$$\mathbb{A} = \{y \mid 0 < y \leq x \wedge \mathsf{prime}(y) \wedge \mathsf{res}(y) \wedge r \neq y\}$$

finally, we get

proc *emirp*(**value** $n : \mathbb{N}_{>0}$, **result** $x : \mathbb{N}_{>0}$) **.**

var $x, r : \mathbb{N}$

$\llcorner x \colon [n > 0,\ \mathsf{prime}(x) \wedge \mathsf{res}(x) \wedge r \neq x \wedge n = \sum_{\forall j \in \mathbb{A}} (1)] \lrcorner_{(0)}$

2 Task 2 : Refinement Calculus

$(0) \sqsubseteq$ $\langle$i-loc2x2$\rangle$

$\quad$ **var** $x := 1;$

$\quad$ **var** $r := 0;$

$\quad \llcorner x, r \colon [n > 0 \land r = 0 \land x = 1,\ \mathsf{prime}(x) \land \mathsf{res}(x) \land r \neq x \land n = \sum_{\forall j \in \mathbb{A}} (1)] \lrcorner_{(1)}$

Define an invariant $L \triangleq \mathsf{prime}(x) \land \mathsf{res}(x) \land r \neq x \land k = \sum_{\forall j \in \mathbb{A}}(1)$ to then use during a sequential composition step.

$$L \triangleq \begin{pmatrix} \forall i \in 2..\sqrt{x}\,(i \nmid x) \land x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-i}) \\ \land r = \sum_{i=0}^{c(x)} (S_i 10^i) \land \forall i \in 2..\sqrt{r}\,(i \nmid r) \\ \land r \neq x \land k = \sum_{\forall j \in \mathbb{A}}(1) \end{pmatrix}$$

$(1) \sqsubseteq$ $\langle$seq and definition of L $\Leftrightarrow$post(1)$\rangle$

$\quad \llcorner x, r, k \colon [n > 0 \land r = 0 \land x = 1, L] \lrcorner_{(2)}$

$\quad \llcorner x, r, k[L,\ k = n \land L] \lrcorner_{(3)}$

$(2) \sqsubseteq$ $\langle$c-frame$\rangle$

$\quad k \colon [n > 0 \land r = 0 \land x = 1, L[{}^{x}/_{x_0}][{}^{r}/_{r_0}]]$

$\sqsubseteq$ $\langle$ass$\rangle$

$\quad k := 0;$

2.1 Proof of (2) $\sqsubseteq k := 0$

$k = k_0 \land pre(2) \Rightarrow post(2)[{}^{0}/_{k}]$ as quoted by the assignment rules

$\quad k = k_0 \land n > 0 \land r = 0 \land x = 1$

$\Rightarrow$ $\langle$Vacuously true$\rangle$

$$\forall i \in 2..\sqrt{x}\,(i \nmid x) \land x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-i}) \land r = \sum_{i=0}^{c(x)} (S_i 10^i) \land \forall i \in 2..\sqrt{r}\,(i \nmid r)$$

$$\land\, r \neq x \land 0 = \sum_{\forall j \in \mathbb{A}} (1)$$

$\Leftrightarrow$ $\langle$Definition of L$\rangle$

$L[{}^{x}/_{x_0}][{}^{r}/_{r_0}][{}^{0}/_{k}]$

Since k is the counter for the number of emirps, k is assigned to 0 - under the consideration of $r = 0 \wedge x = 1$ the post(2) is then vacuously true.

Now, use the negated guard in (3) to initialise a while loop. The negation of $k = n$ is not $k < n$ but since k will only ever be incremented by at most 1 each loop, it is safe to deduce that k will never exceed n.

(3) $\sqsubseteq$ ⟨while loop⟩

while $k < n$ **do**

$\llcorner x, r, k : [L \wedge k < n,\ L] \lrcorner_{(4)}$

od

Lets begin the loop body by defining a sequential composition step.

(4) $\sqsubseteq$ ⟨seq⟩

$\llcorner x, r, k : [L \wedge k < n, L[^{k+1}/_k]] \lrcorner_{(5)}$

$\llcorner x, r, k : [L[^{k+1}/_k],\ L] \lrcorner_{(6)}$

Increment k.

(6) $\sqsubseteq$ ⟨ass and C-frame⟩

$k := k + 1$

Refine step (5) into another sequential composition. Firstly, define an invariant I such that.

$$I \triangleq \left(\begin{array}{l} \forall i \in 2..\sqrt{x}\,(i \nmid x) \wedge x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-i}) \\ \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \forall i \in 2..\sqrt{r}\,(i \nmid r) \\ \wedge r \neq x \wedge k + 1 = \sum_{\forall j \in \mathbb{A}} (1) \end{array} \right)$$

Where the set $\mathbb{A}$ is : $\mathbb{A} = \{y \mid 0 < y \leq x \wedge \mathsf{prime}(y) \wedge \mathsf{res}(y) \wedge r \neq y\}$

(5) $\sqsubseteq$ ⟨seq and i-loc⟩

var $e : \mathbb{B} \cdot \llcorner x, r, e : [L \wedge k < n, e \Leftrightarrow I] \lrcorner_{(7)}$

$\llcorner x, r, e : [e \Leftrightarrow I, L[^{k+1}/_k]] \lrcorner_{(8)}$

Step (7) is refined into a assignment statement.

(7) $\sqsubseteq$ ⟨C-frame⟩

$e : [L \wedge k < n, e \Leftrightarrow I]$

$\sqsubseteq$ ⟨ass⟩

var $e :=$ FALSE;

2.2 Proof of (7) $\sqsubseteq e := \textsf{false}$

$e = e_0 \wedge pre(7) \Rightarrow post(7)[^{\text{FALSE}}/_e]$ as quoted by the assignment rules

$$
\begin{aligned}
& e = e_0 \wedge L \wedge k < n \\
\Leftrightarrow \quad & \langle \text{Definition of L} \rangle \\
& e = e_0 \wedge k < n \wedge \forall i \in 2..\sqrt{x}\,(i \nmid x) \wedge x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-i}) \\
& \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \forall i \in 2..\sqrt{r}\,(i \nmid r) \\
& \wedge r \neq x \wedge k = \sum_{\forall j \in \mathbb{A}} (1) \\
\Rightarrow \quad & \langle \text{Since } x \text{ did not change} \rangle \\
& k + 1 \neq \sum_{\forall j \in \mathbb{A}} (1) \\
\Rightarrow \quad & \langle \text{logical implication of or} \rangle \\
& \neg \forall i \in 2..\sqrt{x}\,(i \nmid x) \vee \neg x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-i}) \\
& \vee \neg r = \sum_{i=0}^{c(x)} (S_i 10^i) \vee \neg \forall i \in 2..\sqrt{r}\,(i \nmid r) \\
& \vee \neg r \neq x \vee \neg k + 1 = \sum_{\forall j \in \mathbb{A}} (1) \\
\Leftrightarrow \quad & \langle \text{Definition of I} \rangle \\
& \neg I \\
\Leftrightarrow \quad & \\
& (e \Leftrightarrow I)[^{\text{FALSE}}/_e]
\end{aligned}
$$

Discharge the conjuncts on the LHS separately.The first two conjuncts have no effect on the RHS. All other conjuncts except the last conjunct are all present within the RHS. The last conjunct on the LHS conveys that k is the sum of all the emirps up to x, yet the corresponding statement on the RHS depicts that $k+1$ is equal to this sum. Since x was not in the frame of spec(7), x did not change. Therefore, the statement on the RHS is false. This supports the assignment of $e = \text{FALSE}$ since $e \Leftrightarrow I$.

Now create a sequential composition to set up a while loop.

(8) $\sqsubseteq$ ⟨S-post⟩

$\llcorner x, r, e : [e \Leftrightarrow I, e \Leftrightarrow I \wedge e = \text{TRUE}] \lrcorner_{(9)}$

(9) $\sqsubseteq$ ⟨while loop⟩

while $e = \text{FALSE}$ **do**

$\llcorner x, r, e : [e \Leftrightarrow I \wedge e = \text{FALSE},\ e \Leftrightarrow I] \lrcorner_{(10)}$

od

NOTE 1: exclude the fact that $k = \sum_{\forall j \in \mathbb{A}}(1)$ for steps through (10) to (17) for the purpose of brevity.

2.3 Proof of (8) $\sqsubseteq x, r, e : [e \Leftrightarrow I, e \Leftrightarrow I \wedge e = \textbf{true}]$

$e \Leftrightarrow I \wedge e = \text{TRUE} \Rightarrow L[^{k+1}/_k]$ as quoted by the Stronger postcondition rules

$$
\begin{array}{ll}
& e \Leftrightarrow I \wedge e = \text{TRUE} \\
\Leftrightarrow & \\
& I \\
\Leftrightarrow & \quad \langle \text{Definition of I} \rangle \\
& \\
& \forall i \in 2..\sqrt{x}\,(i \nmid x) \wedge x = \displaystyle\sum_{i=0}^{c(x)} (S_i 10^{c(x)-i}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \mathsf{prime}(r) \wedge r \neq x \\
& \wedge\, k + 1 = \displaystyle\sum_{\forall j \in \mathbb{A}} (1) \\
\Leftrightarrow & \quad \langle \text{substitution} \rangle \\
& \\
& L[^{k+1}/_k]
\end{array}
$$

Now create a sequential composition step to find the next prime after x, assign a value to r and test if it is prime.

(10) $\sqsubseteq$ ⟨seqx4⟩

con c $\cdot \llcorner x, r, e : [\neg I \wedge c = x_0,\ x > c \wedge \mathsf{prime}(x) \wedge \forall k \in \mathbb{N}\,(c < k < x \Rightarrow \neg \mathsf{prime}(k)) \wedge e = \text{FALSE}] \lrcorner_{(11)}$

$\llcorner x, r, e : [\ \text{post}(11)\ ,\ x = \displaystyle\sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \mathsf{prime}(x) \wedge e = \text{FALSE}] \lrcorner_{(12)}$

$$\textbf{var}\ \mathrm{t} : \mathbb{B} \cdot \llcorner x, r, e, t : [\ \mathrm{post}(12), t \Leftrightarrow \mathsf{prime}(r) \wedge e = \textsc{false} \wedge\ \ \mathrm{pre}(13)]_{\lrcorner(13)}$$
$$\llcorner x, r, e, t : [\ \mathrm{post}(13), e \Leftrightarrow I]_{\lrcorner(14)}$$

Assign a constant c to freeze the previous value of x, in order to use the gmp function gmpFindNextPrime.

$$\begin{aligned}
(11) \sqsubseteq \quad & \langle \text{W-pre, C-frame and i-con} \rangle \\
& \textbf{con}\ \mathrm{c}\ \cdot x : [c = x_0,\ x > c \wedge \mathsf{prime}(x) \wedge \forall k \in \mathbb{N}\,(c < k < x \Rightarrow \neg\mathsf{prime}(k) \wedge e = \textsc{false})] \\
\sqsubseteq \quad & \langle \text{Definition of function gmpFindNextPrime[3]} \rangle \\
& \textbf{proc}\ \mathrm{gmpFindNextPrime}(\textbf{value}\ x\colon \mathbb{N})] \\
& \quad \textbf{con}\ \mathrm{c}\ \cdot x : [c = x_0,\ x > c \wedge \mathsf{prime}(x) \wedge \forall k \in \mathbb{N}\,(c < k < x \Rightarrow \neg\mathsf{prime}(k))]
\end{aligned}$$

The post(11) has the statement $e = \textsc{false}$. However, e is no longer in the frame, therefore this fact will be preserved throughout the proc call.

Introduce a sequence S, defined in the spec, to define the reverse of the number x and assign its value to r.

$$\begin{aligned}
(12) \sqsubseteq \quad & \langle \text{W-pre, C-frame and i-con} \rangle \\
& \textbf{con}\ \mathrm{S}\colon 10^* \cdot r : [x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}),\ x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge e = \textsc{false} \wedge \mathsf{prime}(x)] \\
\sqsubseteq \quad & \langle \text{Definition of proc reversen[3]} \rangle \\
& \textbf{proc}\ \mathrm{reversen}(\textbf{value}\ x\colon \mathbb{N},\ \textbf{result}\ r\colon \mathbb{N}) \\
& \quad \textbf{con}\ \mathrm{S}\colon 10^* \cdot r : [x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}),\ r = \sum_{i=0}^{c(x)} (S_i 10^i)]
\end{aligned}$$

The post(13) on the LHS includes $\mathsf{prime}(x) \wedge e = \textsc{false} \wedge x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1})$. However, this is irrelevant since x and e is excluded from the frame and will never change. Therefore, this statement is preserved.

Introduce a boolean variable t to hold the result of gmpPrimeTest, which will asses whether the reverse of x, r, is prime or not.

$$\begin{aligned}
(13) \sqsubseteq \quad & \langle \text{W-pre, C-frame and i-loc} \rangle \\
& \textbf{var}\ \mathrm{t}\colon \mathbb{B} \cdot t : [\textsc{true},\ t \Leftrightarrow \mathsf{prime}(r) \wedge x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge e = \textsc{false} \wedge \mathsf{prime}(x)] \\
\sqsubseteq \quad & \langle \text{Definition of function gmpPrimeTest[3]} \rangle \\
& \textbf{proc}\ \mathrm{gmpPrimeTest}(\textbf{value}\ r\colon \mathbb{N},\ \textbf{result}\ t\colon \mathbb{B}) \\
& \quad \textbf{var}\ \mathrm{t}\colon \mathbb{B} \cdot t : [\textsc{true},\ t \Leftrightarrow \mathsf{prime}(r)]
\end{aligned}$$

The post(14) on the LHS includes $x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge e = \textsc{false} \wedge \mathsf{prime}(x)$. However, this is irrelevant since x and r and e are excluded from the frame and will never change.

Therefore, these statements are preserved.

Introduce a if statement to check if r is prime.

(14) $\sqsubseteq$ $\langle$if condition and C-frame for x, r, $t\rangle$

if t=TRUE **then**

$$\llcorner e : [x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \mathsf{prime}(r) \wedge \mathsf{prime}(x) \wedge e = \text{FALSE}, e \Leftrightarrow I] \lrcorner_{(15)}$$

else

$$\llcorner e : [x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \neg\mathsf{prime}(r) \wedge \mathsf{prime}(x) \wedge e = \text{FALSE}, e \Leftrightarrow I] \lrcorner_{(16)}$$

fi

(16) $\sqsubseteq$ $\langle$W-pre and C-frame and e-frame$\rangle$

$$e : [x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \neg\mathsf{prime}(r) \wedge \mathsf{prime}(x) \wedge e = \text{FALSE}, e \Leftrightarrow I]$$

$\sqsubseteq$ $\langle$skip$\rangle$

$SKIP$

Since e was not changed from it's original value, step (16) can be refined into SKIP.

2.4 Proof of (16) $\sqsubseteq SKIP$

$pre \Rightarrow post[^{e}/_{e_0}]$ as quoted by the Skip rules

$$x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \neg\mathsf{prime}(r) \wedge \mathsf{prime}(x) \wedge e = \text{FALSE}$$

$\Rightarrow$ $\langle$definition of $\mathsf{prime}(r)\rangle$

$$\neg\forall i \in 2..\sqrt{r}\,(i \nmid r) \wedge e = \text{FALSE}$$

$\Rightarrow$ $\langle$logical implication of or$\rangle$

$$\neg\forall i \in 2..\sqrt{x}\,(i \nmid x) \vee \neg x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-i})$$

$$\vee \neg r = \sum_{i=0}^{c(x)} (S_i 10^i) \vee \neg \forall i \in 2..\sqrt{r}\,(i \nmid r)$$

$$\vee \neg r \neq x \vee \neg k + 1 = \sum_{\forall j \in \mathbb{A}} (1) \wedge e = \text{FALSE}$$

$\Leftrightarrow$ ⟨Definition of I⟩

$$e \Leftrightarrow I \wedge e = \text{FALSE}$$

$\Rightarrow$ ⟨Definition of I⟩

$$e \Leftrightarrow I$$

(15) $\sqsubseteq$ ⟨if condition⟩

if $r \neq x \ \wedge$ **then**

$\llcorner e : [\text{pre}(15) \wedge r \neq x,\ e \Leftrightarrow I]_{\lrcorner(17)}$

else

$\llcorner e : [\text{pre}(15) \wedge r = x,\ e \Leftrightarrow I]_{\lrcorner(18)}$

fi

Only logical step is that e must be assigned to TRUE.

(17) $\sqsubseteq$ ⟨ass⟩

$e :=$ TRUE

2.5 Proof of (17) $\sqsubseteq e :=$ true

$w = w_0 \wedge pre \Rightarrow post[^{\text{TRUE}}/_e]$ as quoted by the Assignment rules

$$x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \mathsf{prime}(r) \wedge \mathsf{prime}(x) \wedge e = \text{FALSE} \wedge r \neq x$$

$\Rightarrow$ ⟨exclude the fact that e=false since this isn't relevant⟩

$$x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \mathsf{prime}(r) \wedge \mathsf{prime}(x) \wedge r \neq x$$

$\Rightarrow$ ⟨NOTE 1 : statement of brevity [page 6] and the fact that $x > c$ from step (11) on page 6⟩

$$x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i) \wedge \mathsf{prime}(r)$$
$$\wedge\, \mathsf{prime}(x) \wedge r \neq x \wedge k = \sum_{\forall j \in \mathbb{A}} (1) \wedge x > c$$

$\Rightarrow$ ⟨Definition of I⟩

$$\text{TRUE} \Leftrightarrow (x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-i}) \wedge r = \sum_{i=0}^{c(x)} (S_i 10^i)$$
$$\wedge\, \mathsf{prime}(x) \wedge \mathsf{prime}(r)$$
$$\wedge\, r \neq x \wedge k + 1 = \sum_{\forall j \in \mathbb{A}} (1))$$

$\Leftrightarrow$ ⟨Definition of I⟩

$$(e \Leftrightarrow I)[^{\text{TRUE}}/_e]$$

Discharge all conjunct on the LHS separately. 1-5 conjuncts on the LHS are all present in the RHS. To better understand what we are looking at we expand the 6 and the 7 conjunct on the LHS.

$$k = \sum_{\forall j \in \mathbb{A}} (1) \wedge x > c \wedge \mathsf{prime}(x)$$
where $\mathbb{A} = \{y \mid 0 < y \leq c \wedge \mathsf{prime}(y) \wedge \mathsf{res}(y) \wedge r \neq y\}$

This is justified since k has not changed from it's state on page 6 and x and c have not been change since page 6
Now expand the RHS

$$k + 1 = \sum_{\forall j \in \mathbb{A}} (1) \wedge \mathsf{prime}(x)$$
where $\mathbb{A} = \{y \mid 0 < y \leq x \wedge \mathsf{prime}(y) \wedge \mathsf{res}(y) \wedge r \neq y\}$

On the LHS k is equal to the sum of all the emirps up to the old value of x, which is the value c, and on the RHS $k + 1$ is equal to the sum of all the emirps up to the new value of x. Therefore clearly, LHS $\Rightarrow$ RHS.

(18) $\sqsubseteq$ ⟨C-frame and ass⟩
$e := \text{FALSE}$
$\sqsubseteq$ ⟨skip⟩
$SKIP$

Proof similar to refinement step (16)

The full program here:

```
var x := 1;
var r := 0;
var k := 0;
while k < n do
    var e := FALSE;
    while e = FALSE do
        x := gmpFindNextPrime(x);
        reversen(x, r);
        var t := gmpPrimeTest(r)
        if t = TRUE then
            if r ≠ x then
                e := TRUE;
            fi
        fi
    od
    k := k + 1;
od
```

With the C implementation following:

```
1  void emirp(unsigned long n, mpz_t x)
2  {
3          //Declare variables
4          mpz_t k;
5          mpz_t r;
6
7          //Initialise variables (Both will be set to 0)
8          mpz_init(k);
9          mpz_init(r);
10
11         //Set values
12         mpz_set_ui(x, 1);
13
14         while (mpz_cmp_ui(k, n) < 0) //GMP comparison function, returns negative if op1 >= op2
15         {
16                 bool e = false;
17                 while (e == false)
18                 {
```

```
19                          // Uses temp to hold previous value of x
20                          mpz_t temp;
21                          mpz_init(temp);
22                          mpz_set(temp, x);
23
24                          mpz_nextprime(x, temp);
25                          reversen(x, r); //r now contains reversed order of x's digits
26
27                          int t = mpz_probab_prime_p(r, 30); //gmp primality testing function
28                          if (t != 0) // t == TRUE
29                          {
30                                  if (mpz_cmp(x, r)) //Guard against palindromic primes
31                                  {
32                                          e = true;
33                                  }
34                          }
35                  }
36                  mpz_add_ui(k, k, 1);
37          }
38  }
```

Limitations of the current program implementation:
Since we are using the GMP supplied primality testing function, an element of error is present within the program. This function provides three sets out return values; definitely not prime, definitely prime and could be prime. In our implementation, we have interpreted any return value apart form definitely not prime as prime, which means there is a possibility that a composite number could be interpreted as a prime. This chance is highly unlikely however, as we used a fairly high constant within the function, resulting in better accuracy. Time and resources are also a limiting factor, as calculating emirps in excess of 100,000 gets extremely costly in terms of both, but this should be expected with any prime searching function.

Justification of choice of representation:
Our choice to use the GMP functions was done in order to make a more universal, robust program. We wanted to avoid the limitation of unsigned long sizes for the numbers, and instead used the mpz-t data structure provided by GMP in order to allow for bigger numbers. We also used the gmp functions for prime comparison and prime retrieval, as these functions are fast, robust and well documented. This is demonstrated from the times we obtained from our inputs, with no large time or resource penalty until reaching in excess of the 50,000th emirp. Keeping in mind that any prime retrieval program will be large in terms of complexity, we believe we have done well in building a solution that is efficient and practical

3 Appendix

Use of the gmp function $< mpz_n extprime() >$ for derivations in the proof. Assumed the spec of the function conforms to this spec.

func gmpFindNextPrime (**value** $x : \mathbb{N}$)
  con $c : \mathbb{N} \cdot x : [c = x_0,\ x > c \wedge \mathsf{prime}(x) \wedge \forall k \in \mathbb{N}\,(c < k < x \Rightarrow \neg\mathsf{prime}(k))]$
  return x;

Use of the proc reversen given to us by Kai in assignment 1 spec.

proc reversen (**value** x $: \mathbb{N}$, **result** $r : \mathbb{N}$)
  con $S : [10]^* r : [x = \sum_{i=0}^{c(x)} (S_i 10^{c(x)-1}),\ r = \sum_{i=0}^{c(x)} (S_i 10^i)]$

Use of the gmp function $< mpz_p robab_p rime_p >$ for derivations in the proof. Assumed the spec of the function conforms to this spec.

func gmpPrimeTest (**value** x $: \mathbb{N}$)
  var $p : \mathbb{B} \cdot p : [\text{TRUE},\ p \Leftrightarrow \mathsf{prime}(x)]$
  return p;

Timing data was collated using the function to find the n'th emirp, with n from 10-800,000. This is included below:

```
1  // user times recorded from Unix "time" command
2  // n size | time recorded (seconds)
3
4  // Emirps from 10 − 9000 are verified to be correct
5  10 0.000
6  100 0.000
7  500 0.000
8  1000 0.012
9  2000 0.024
10 5000 0.080
11 9000 0.212
12
13 // Emirps from 50 000 − 800 000 likely to be correct, but are not verified
14 50 000 13.156
15 100 000 22.404
16 500 000 3m 21.360s
17 800 000 4m 18.584s
```

The following funciton was used to test the function's output. The output was sent to file "output2.txt", where it was then compared with a list of verified emirps. This was verified correct up to the 9000th emirp, where further testing was stopped due to time constraints, but further output is implied true.

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <gmp.h>
4  #include "reverse.h"
5
6  void testEmirps()
7  {
8          int i = 1;
9
10         FILE *fp = fopen("output2.txt", "w");
11         while (i < 10001)
12         {
13                 //printf("yo\n");
14                 mpz_t temp;
15                 mpz_init(temp);
16                 emirp(i, temp);
17                 unsigned long temp2 = mpz_get_ui(temp);
18                 fprintf(fp, "%lu\n", temp2);
19                 i++;
20                 printf("%d\n", i);
21         }
22         fclose(fp);
23 }
```