Asgn 4 Design. PPf.

flow chart

switch () {

- s // print prime
  identify types

- p // Print palindromic

- n // set boundary
  default = 1000 }

Main () {

-s

printf : prime

check - mersenne

- lucas

- fibonacci

printf : /n

-p

Base 9

2 = 2

3 = 3.

↓

decimal = base transform

-n

bound = atoi (optarg(s)

Pre-Lab Part 1

① Fabonacci (int prime)

```
int previsous = 1

int current = 1

while ( cur <= prime) {
    if ( cur == prime)
        printf - fabacci
    int temp = cur
    cur = cur + prev
    prev = temp
    }
```

}

Lucas (int prime)

```
int previsous = 2
int current = 1
while ( cur <= prime) {
    if ( cur == prime)
        printf - fabacci
    int temp = cur
    cur = cur + prev
```

prev = temp

}

Mersenne (int prime)

```
int num = 2 - 1
while ( num <= prime ) {
    if ( num = prime )
     printf : mersenne.
num = (num+1) · 2 - 1
    }
```

(2) -p palindromic

- Base 2
- Base 9
- Base 10
- Base 21 (k)

printpal ()

bool pal ()

bool pal ()

//check if they are pal

1. Only check prime.

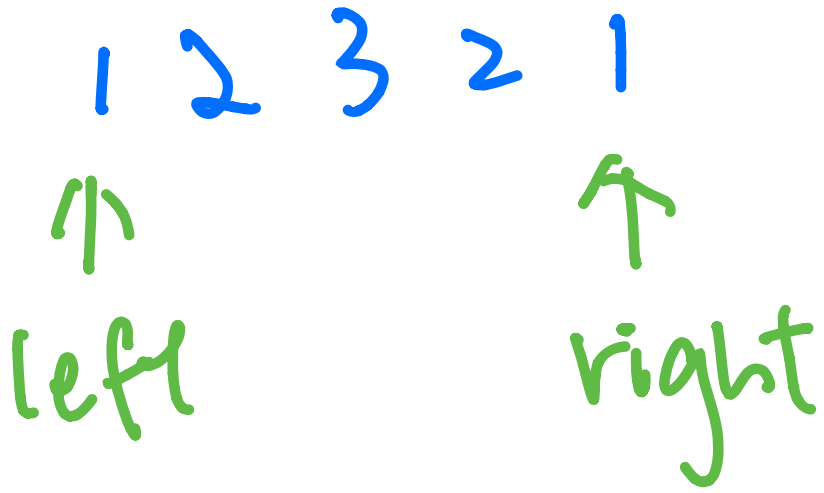
```
if ( bv_get_bit(i) == 1)
```

2. Transform to new base.

Ex. base 2.

```
char* s.
while (num > 0)
        s = num % base + s
num /= base
```

3. Check if the string is palindrome.

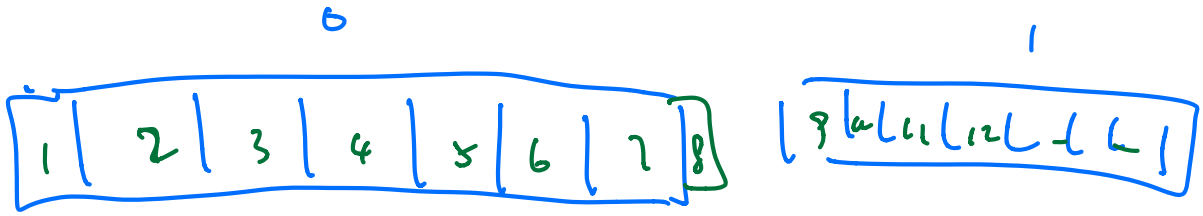



```
while (left <= right)
    if (s[left] != s[right])
        return false;

    left++;

    right--;
```

```
void printpal ()

while (prim) {

    if (pal ())

    printf : prime num

    printf : transform
                 base

}
```

Pre lab Part 2

(1)

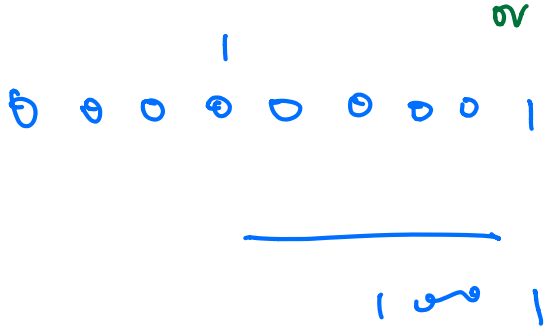


For Ex. Number 9 goes to index 1.

9 / 8 = 1

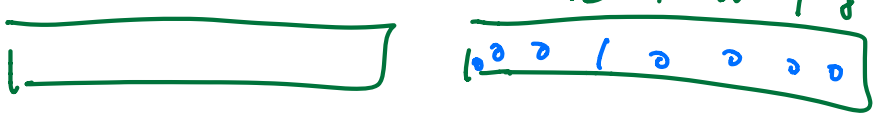
← get into right index first.

set → vector [i >> 3] ← i/3



Ex Number 12.

we want



vector [i >> 3] or (1 << (12 & 7))

12 = 1100
0111

0100 = 4.

bv_ creat ( ) {

malloc size of vector [].

↓

which is length / 8.

}

bv_ delete () {

free (vector)

free (v)

}

bv_get_len () {

return length }

bv_set_bit (v, number i) {

vector[i>>3] or (1<<(i & 7)) }

↑ find index

↑ should be something like 00010000

bv_clr_bit (v, number i) {

vector[i>>3] and [not (1<<(i & 7))]

↑ 11101111

}

bv_get_bit (v, number i) {

return vector[i>>3] and (1<<(i & 7))

↓

00010000

// If Not Zero
then true

}

bv_set_all_bit (v, number i) {

for j in vector[].

vector[j] or 255 (.....)

}

(2) The bv_delete function will free the memory at the end of the main().

(3) Some of the index one getting repeated. I might write another algorithm to see if I can reduce the time of getting In

repeated indices.