Lecture 4

More playing with numbers and how to think about loops in programs.

Practical review

- We know that the 1st of January 1900 was a Monday, we want to construct a program which will take in a date since then and output what day of the week it was.

- Remember we already have some nice functions, isleap and maxdays, maybe we will be able to use them in our solution.

Practical review

- If we could calculate the number of days that had passed from the 1st of January 1900 until the day we entered then the remainder of the program would be quite easy.

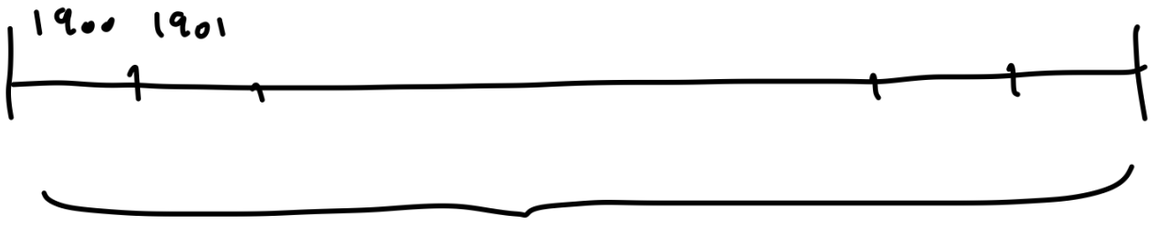
- We decide what to output using the If statement on the next slide

```
if (days_elapsed % 7 == 1)
      { printf("that day was a Monday \n"); }
else if (days_elapsed % 7 == 2)
      { printf("that day was a Tuesday \n"); }
else if (days_elapsed % 7 == 3)
      { printf("that day was a Wednesday \n"); }
else if (days_elapsed % 7 == 4)
      { printf("that day was a Thursday \n"); }
else if (days_elapsed % 7 == 5)
      { printf("that day was a Friday \n"); }
else if (days_elapsed % 7 == 6)
      { printf("that day was a Saturday \n"); }
else if (days_elapsed % 7 == 0)
      { printf("that day was a Sunday \n"); }
```

Practical review

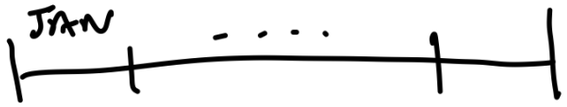
- Now let us concentrate on how to put the current value into the variable days_elapsed. Suppose the user has given us values for d, m, and y.

- It will be the sum of
  - The number of days in each of the complete years since 1900.
  - The number of days in each of the complete months in the current year.
  - The number of days since the start of the current month.



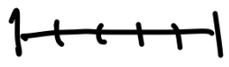
FULL YEARS

+



FULL MONTHS

+



DAYS

THIS IS A NICE SIMPLE WAY TO DECOMPOSE (BREAK UP) THE PROBLEM.

EACH of THIS SMALLER TASKS CAN BE PROGRAMMED EASILY.

- **The number of days in each of the complete years since 1900.**

```
int days_in_full_years( int y)
  {
  int total;
  int i;

  i = 1900 ;
total = 0;
while (i != y)
  {
  if (leap(i))
    { total = total + 366 ;}
  else if (!(leap(i)))
    { total = total + 365 ;}
  i = i + 1 ;
  }
return total ;
}
```

- **The number of days in each of the complete months in the current year.**

```
int days_in_full_months(int m, int y)
   {
   int total;
   int i;

   total = 0;
   i = 1;
   while (i != m )
      {
      total = total + maxdays(i, y);
      i = i + 1 ;
      }
   return total ;
   }
```

Calculate days_elapsed

```
days_elapsed = days_in_full_years(year) + days_in_full_months (month,year) + day ;
```

A problem

- Given an integer, we would like to have a function which every time we find the digit 7 in it we would replace it by the digit 3

- Replace (1273727) = 1233323

- A more general function would be one which had 3 parameters x and y (single digit integers) and n an integer, and returned n with every occurrence of the digit x replaced by the digit y.

replace (int x, int y, int n)

Suppose n was just a single digit
then if x == n we return y,
But if x != n then we return x

We could write this as a small function

```
single_replace (int x, int y, int n)
{
   if (n == x)
        { return y ; }
   else if (n != x)
        { return n ; }
}
```

replace (int x, int y, int n)

But if the number was bigger than 10 we could “split” the number into the right hand digit and all of the left hand digits,
solve the problem for all of the left hand digits,
multiply that by 10 and add on the solution for the single right hand digit.

```
replace (int x, int y, int n)
{
  if (n < 10)
    { return single_replace (x, y, n) ; }
  else
    { return ( replace (x, y, n/10) * 10 ) + single_replace (x, y, n%10) ; }
}
```

A Challenge

- We would like to have a function which when we give it an integer it would return the reverse of the integer.

- So we want reverse(1234) = 4321

- We can use the operators “/“ and “%” to break the number apart and look at the individual digits, so it should be easy to do.

A Challenge

We can break 1234 into 123 4
we would expect 4 to be at the start of the reversed number

4XXX where XXX stands for reverse(123)

We can break 123 into 12 3
we would expect 3 to be at the start of the reversed number

43YY where YY stands for reverse(12)

We can break 12 into 1 2
we would expect 2 to be at the start of the reversed number

432Z where Z stands for reverse(1)

But reverse(1) is just 1.

So we get 4321

A Challenge

To do this we actually need to write the function as a function which takes 2 parameters.

The first parameter is what remains to be reversed.
The second parameter is what has already been reversed.

It will behave something like this

reverse(1234, 0)
=
reverse(123, 4)
=
reverse(12, 43)
=
reverse(1, 432)
=
4321

A Challenge

```
int reverse( int n, int x);
        {
        if ( n < 10)
        { return x*10 + n; }
else
        { return reverse(n/10, x*10 + n%10) ; }
}
```

The second parameter is often called an accumulator, as it is used to accumulate the answer as the function calls itself.

This shape of function is quite common.

Review of functions

We have seen some simple functions such as isleap, maxdays, days_in_full_years etc...

These allow us to decompose the solution to a problem into smaller parts.

When writing these functions we use parameters to identify the data which the function will use to produce the answer. We can ignore other data used in th only concentrate on what is important.

We also give a function a type which describes what sort of value will be produced at the end. You must remember to return this value to the main program.

Review of functions

Some of the functions we have written have had a special shape. These functions have called themselves in solving the problem.

A function which calls itself is called a **Recursive** function.

A recursive function always has at least 2 parts.

the “base case” where we do not need to use the function again to get the solution

the “recursive case” where we call the function again with a smaller problem and use that answer to get the solution.

Recursive solutions are usually much shorter to write and much more elegant.

Reasoning about loops

- Almost all of the programs you write will involve using Loops of Recursion.

- You already have written many Loop programs. Now I want to look in some detail at how we reason about Loop programs.

- We will just consider While Loops.

In a program we often want to repeat a set of actions. C provides us with a few different ways to express this but the best way is by using a while statement. Here is the shape of a while statement

```
while ( condition )
    {
    action ;
    }
```

Some terminology.

The condition is often called the **Loop Guard**.
The action is often called the **Loop Body**.

If the condition is **true**, then we enter the loop and perform the action. When we finish we once again check the condition, if it is still true we perform the action again. We continue to do this.

```
while ( condition )
    {
    action ;
    }
```

When the condition is **false** we do not enter the loop, instead we move on to the next statement in the program.

It is important that some action within the ***loop body eventually makes the condition false***. Otherwise, the loop will continue forever.

Suppose we are asked to write a program to sum the first 100 natural numbers.

The natural numbers start at 0

Here is a list of the numbers

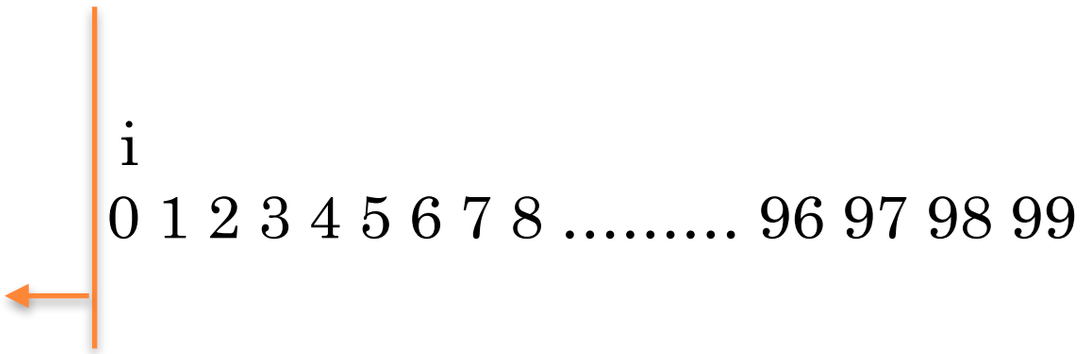
0 1 2 3 4 5 6 7 8 ..... 96 97 98 99

I am going to make use of 2 integer variables

r will store the sum of the values
i will store the current value

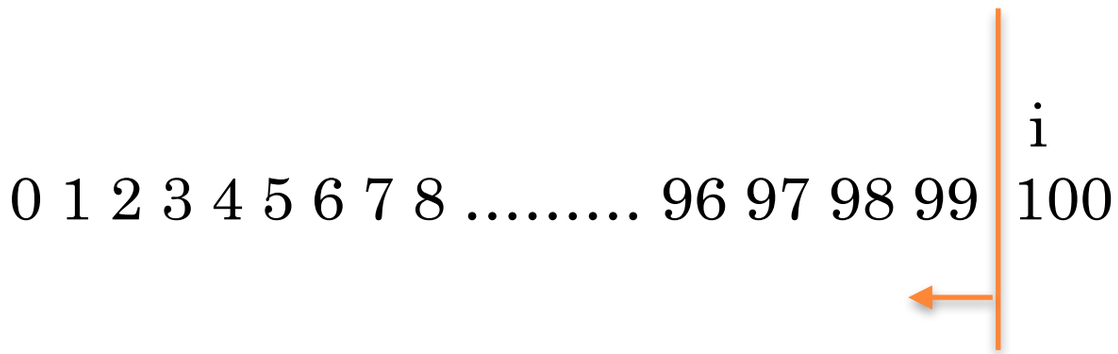
now let us study this problem

Here is a picture at **the start**.



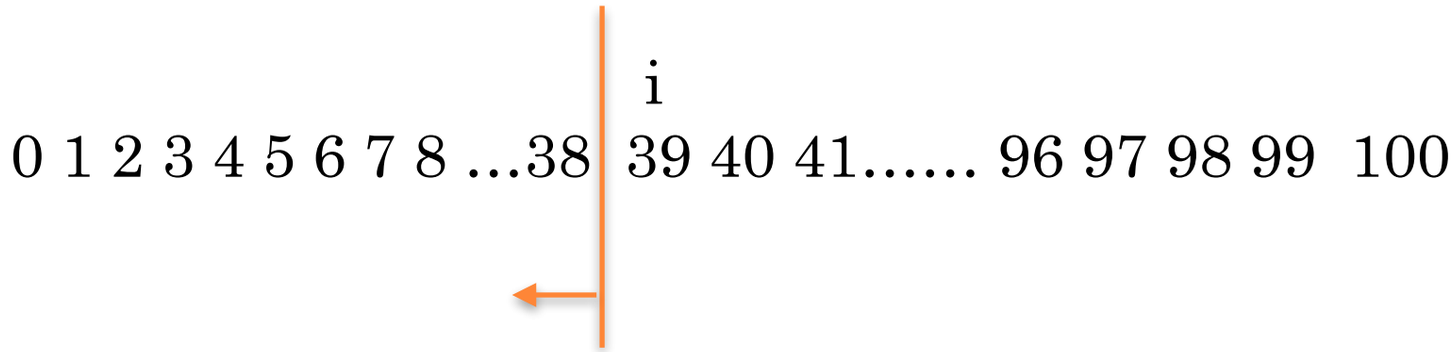
everything to the left of the line has been summed
and stored in variable r

Here is a picture at **the end**.



everything to the left of the line has been summed
and stored in variable *r*

Here is a picture at **some point in between**.

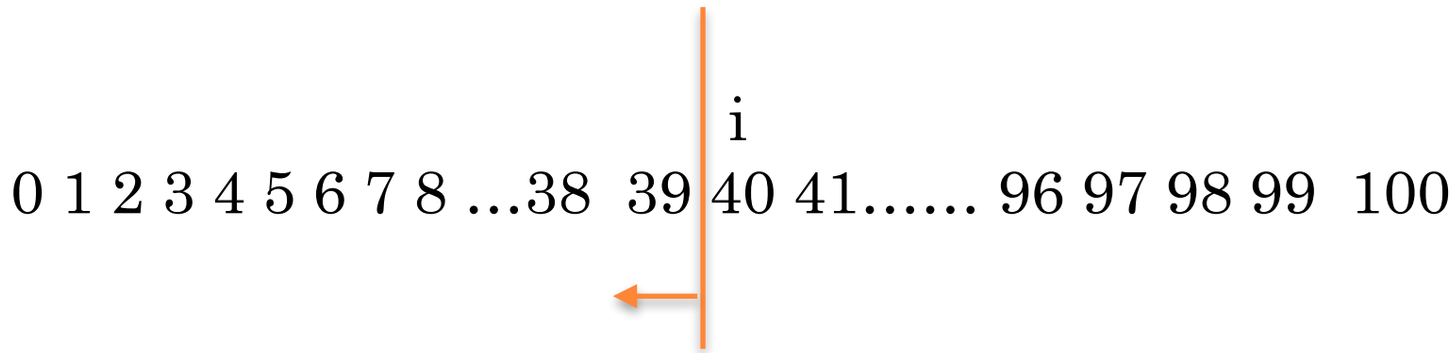


everything to the left of the line has been summed
and stored in variable r

Now what do you think you should do here to make progress?

Add the current value 39 to the sum in variable r
Move the line one place to the right.

Here is a picture at **some point in between**.



everything to the left of the line has been summed
and stored in variable r

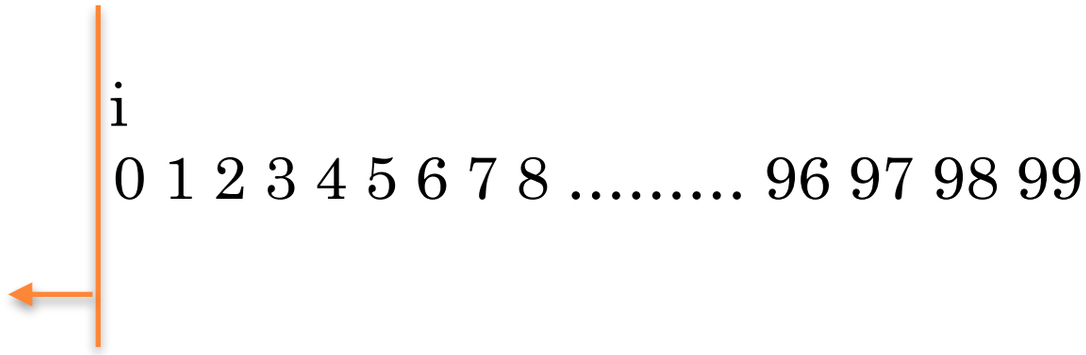
We have added in the value 39 and moved the line one place to the right.
Now we are getting closer to the end.

Reasoning about Loops

- No we are going to go through that example again but at each stage we will add in the C statements that make the pictures true.

- So our picture of the state of the solution and the actual program will be the same

Here is a picture at **the start**.

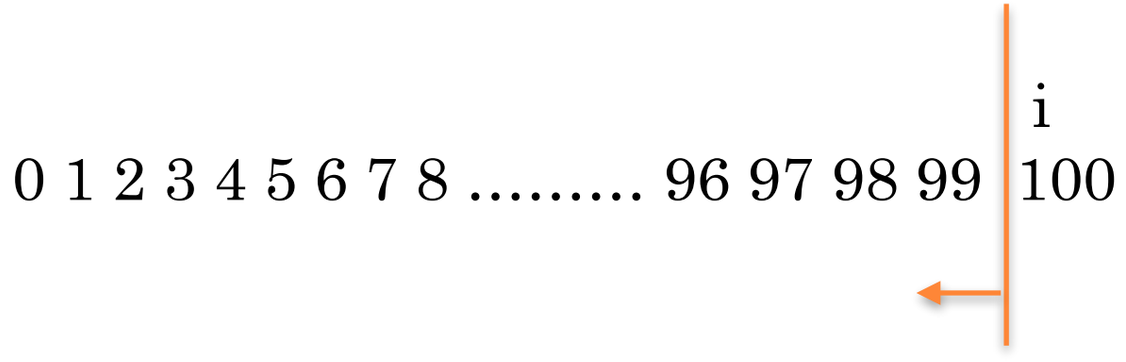


everything to the left of the line has been summed
and stored in variable r

Expressed in C

```
r = 0 ;
i = 0 ;
```

Here is a picture at **the end**.



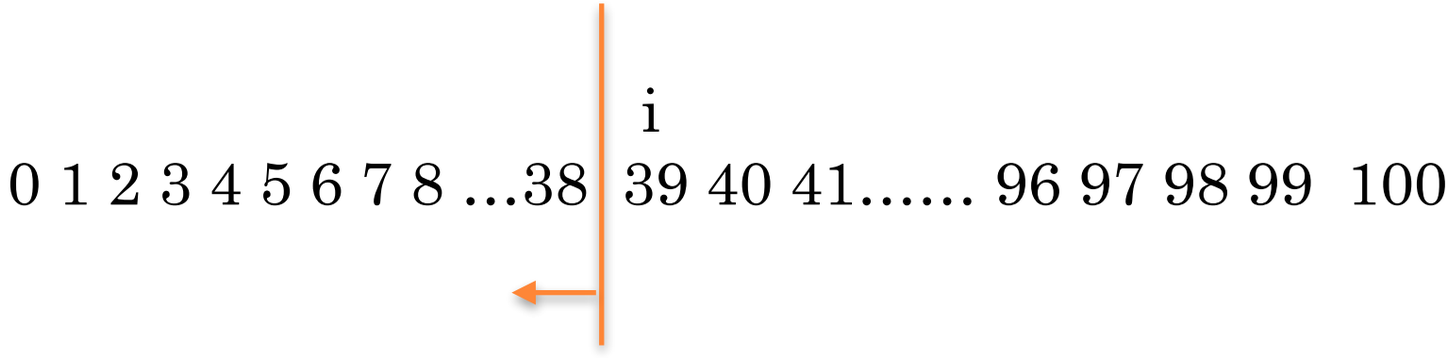
everything to the left of the line has been summed and stored in variable r

At this point we are finished so i == 100

So when we are not finished the guard on the loop will be i != 100

```
While (i != 100)
    {
    }
```

Here is a picture at **some point in between**.



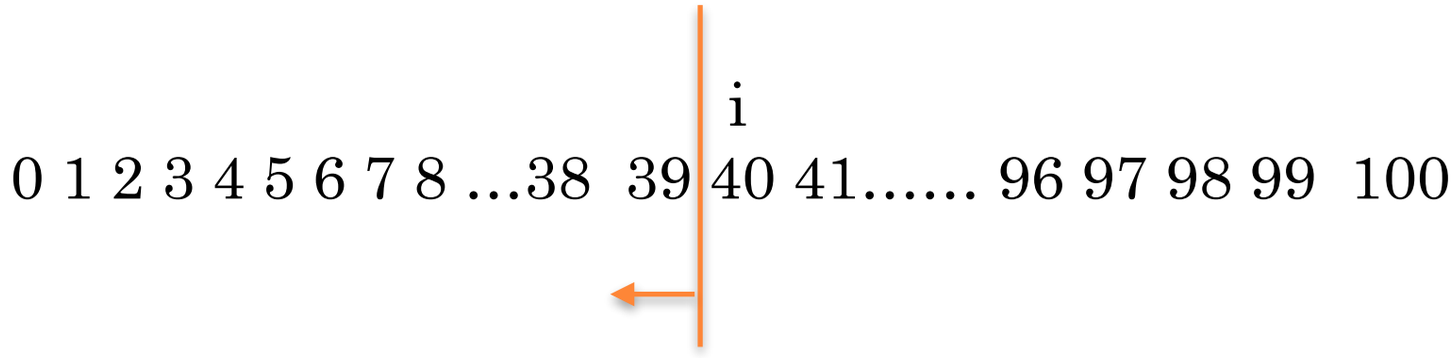
everything to the left of the line has been summed
and stored in variable r

Add the current value 39 to the sum in variable r
Move the line one place to the right.

Expressed in C

```
r = r + i ;
i = i + 1 ;
```

Here is a picture at **some point in between**.



everything to the left of the line has been summed
and stored in variable r

Add the current value 39 to the sum in variable r
Move the line one place to the right.

Expressed in C

```
r = r + i ;
i = i + 1 ;
```

Finished program

```
#include <stdio.h>
main ()
    {
    int r ;
    int i ;

    r = 0 ;
i = 0 ;
while ( i != 100)
    {
    r = r + i ;
    i = i + 1 ;
    }
printf ( “the sum of the first 100 naturals : %d \n”, r) ;
}
```