# ESc 101: Fundamentals of Computing

Lecture 11

Jan 25, 2010

- Why did we end up with the messy program for something as simple as adding integers?
- We blindly started writing the program, patching it whenever we found a mistake.
- This is poor programming technique, and invariably ends up with:
    - messy programs,
    - many errors along the way, and
    - hard-too-understand programs.

## Revisiting the Problem

Let us write down the steps we wish the program to perform, at a high level:

```
1. Read a number.
2. Read another number.
3. Add the two numbers.
4. Output the result.
```

## Fixing Data Structure

A data structure is an organization of memory locations in which we store the data needed for the program.

Let us now fix the data structure needed for storing large numbers.

## Fixing Data Structure

- Assume that the largest number will be 100 digits long.
- As we have already discussed, we can store it in an array of size 100.
- However, there are still two issues to be resolved:
    - How do we handle different number of digits in numbers?
    - In what sequence should we store the digits of a number?

## Handling Size

- One way is to store the digits in sequence and then store a non-digit to signify end of number:
  - This means that we should declare an array of size 101 instead of 100.
- Another way is to store the size in a separate variable:
  - This means that the number would require an array plus an additional variable to store.

Let us choose the first method.

## Most or Least Significant Digit First?

- Storing most significant digit first creates problem during addition:
    - In case the result has an extra digit, we have to shift the entire sequence by one.
    - When the number of digits are different for two numbers, we have to start the addition at different indices.
- Storing least significant digit first creates problem during input / output:
    - The input is read with most significant digit first, we have to reverse the order for storage.
    - Same issue during output.

Let us choose to store least significant digit first.

## An Advantage of Storing LSD First

- We do not need to end the digit sequence of a number with a non-digit!
- Just fill in 0's after the digits are over up to `SIZE` digits.
- Now the size of the array needs to be `SIZE` only.

## Back to Program: Declarations

```
#define SIZE 100
main()
{
   /* Digits of numbers are stored in reverse order */
   char number1[SIZE]; /* first number */
   char number2[SIZE]; /* second number */
   char number3[SIZE]; /* stores result */
```

## Reading a Number

```
char symbol; /* stores current input symbol */
int size; /* counts the digits in input number */
char temp[SIZE]; /* temporary storage for numbers */

symbol = getchar(); /* read first symbol */
for (size = 0; (symbol >= ‘0’) && (symbol <= ‘9’)
                    && (size < SIZE); size++) {
   temp[size] = symbol - 48;
   symbol = getchar(); /* read next symbol */
}
```

## Reading a Number

```
   if (size == SIZE) { /* input may be too large */
      symbol = getchar();
      if ((symbol >= ‘0’) && (symbol <= ‘9’)) { /* too large
*/
         printf(’’Input too large: number should be at most
                     %d digits’’, SIZE);
         return;
      }
   }

   /* store in  number1 in reverse order */
   int i;
   for (i = 0; i < size; i++)
      number1[i] = temp[size-1-i];
   for (i = size; i < SIZE; i++)
      number1[i] = 0;
```

## Reading a Number: Alternative

```
char symbol; /* stores current input symbol */
int size; /* counts the digits in input number */
char temp[SIZE]; /* temporary storage for numbers */

symbol = getchar(); /* read first symbol */
for (size = 0; 1; size++) {
   if ((symbol < ‘0’) || (symbol > ‘9’)) /* not a digit */
      break;
   if (size == SIZE) { /* number too large */
      printf(’’Input too large: number should be at most
                    %d digits’’, SIZE);
      return;
   }
   temp[size] = symbol - 48;
   symbol = getchar(); /* read next symbol */
}
```

## Reading a Number: Alternative

```
/* store in  number1 in reverse order */
int i;
for (i = 0; i < size; i++)
   number1[i] = temp[size-1-i];
for (i = size; i < SIZE; i++)
   number1[i] = 0;
```

## Adding Two Numbers

```
int i;
int carry; /* stores the carry value */

for (i = 0, carry = 0; i < SIZE; i++) {
   number3[i] = number1[i] + number2[i] + carry;
   if (number3[i] > 9) { /* new carry */
      number3[i] = number3[i] - 10;
      carry = 1;
   }
   else /* no carry */
      carry = 0;
}
```

## Adding Two Numbers

```
if (carry == 1) { /* sum exceeds the size */
  printf(''Sum too large!\n'');
 return;
}
```

This is better but does not take care of negative numbers!