ESc 101: Fundamentals of Computing

Lecture 13

Jan 28, 2010

Algorithms

- An algorithm is a stepwise description of operations to solve a problem.
- It is usually in a natural language, not a computer language.
- It also includes the description of the data structures to be used.
- The first step for creating a program to solve a problem should be to design a suitable algorithm for it.

Algorithms and Functions

- Functions provide a very convenient way of implementing algorithms.
- Using functions, we can often mimic the steps of algorithms closely.

Example: Adding Numbers Algorithm

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

Example: Adding Numbers main Program

```
main()
{
   char number1[SIZE]; /* stores first number */
   char number2[SIZE]; /* stores second number */
   char number3[SIZE]; /* stores the result */

   /* Read first number */
   if (read_number(number1) == 0) /* error */
      return;
   /* Read second number */
   if (read_number(number2) == 0) /* error */
      return;
   /* Add the two numbers */
   if (add_numbers(number1,number2,number3) == 0) /* error */
      return;
   output_number(number3); /* output result */
}
```

Example: Generating Prime Numbers Algorithm

```
Input: number n /* First n primes to be generated */

1. Read number n
2. For every number between 2 and n do:
          output if it is prime.
```

Example: Generating Prime Numbers Program

```
main()
{
   int n; /* upper limit */
   int i;

   printf(''Input  n: '');
   scanf(%d, &n); /* read n */

   /* Output all prime numbers <= n */
   printf(''Prime numbers <= n are:\n'');
   for (i = 2; i <= n; i++)
      if (is_prime(i))
         printf(''%d '', i);
}
```

Example: Generating Prime Numbers Program

```
int is_prime(int m)
{
   int i;

   for (i = 2; i < m; i++)
      if (m % i == 0) /* m is composite */
         return 0;

   return 1; /* m is prime */
}
```

Example: Computing GCD Algorithm

```
1. Read numbers n and m.
2. Compute GCD of n and m.
3. Output the gcd.
```

Example: Computing GCD Program

```
main()
{
   int n; /* first number */
   int m; /* second number */

   /* Read n and m */
   printf(''Input two numbers:'');
   scanf(''%d'', &n);
   scanf(''%d'', &m);

   /* Find gcd */
   t = compute_gcd(n, m);

   /* Output gcd */
   printf(''The GCD is: %d\n'', t);
}
```

Computing GCD: First Method

```
Strating from n, and subtracting one each time,
find the largest number that divides both n and m.
```

Corresponding Program

```
int compute_gcd(int n, int m)
{
   int t; /* stores GCD */

   for (t = n; 1; t--)
      if ((n % t == 0) && (m % t == 0)) /* both are divisible
*/
         return t;

/* No need to worry about other cases,
 * because when t = 1, it will divide both n and m
*/
}
```

Computing GCD: Eucind's Method

```
1. Make n the larger number, swapping if required.
2. if m divides n, gcd is m.
3. Otherwise, replace n by n (mod m).
4. Go to 1.
```

Corresponding Program

```
int compute_gcd(int n, int m)
{
   int t; /* needed for swapping */

   for (; 1; ) {
      if (n < m) { /* swap */
         t = m;
         m = n;
         n = t;
      }
      if (n % m == 0) /* m is gcd */
         return m;
      else
         n = n % m;
   }
}
```

Two Algorithms for GCD

- The first algorithm for computing gcd goes through all numbers between `n` and 1 when the gcd of `n` and `m` is 1.
- The second algorithm, on the other hand, proceeds much faster – in a single iteration, the value of `n` goes from being larger than `m` to being smaller than `m`.
- Hence, the second algorithm is faster than the first one – which can be observed by running the two algorithms on large inputs.
- Thinking carefully about the problem and writing down the algorithm before writing a program is important for this reason too: We may be able to discover a faster way of solving the problem.