CS11001/CS11002
Programming and Data Structures (PDS)
(Theory: 3-1-0)

Loops

The *for* loop

- for (initialize loop; continuation condition ; loop increment)

  { execute loop body; }
- The for loop can be equivalently described in terms of the following while loop:
  - initialize loop;

    while (continuation condition is true)

    { execute loop body;

    loop increment; }

Example

- One can compute gcds using for loops as follows:

```
for ( ; b > 0 ; )
{
        r = a % b; /* Compute the next remainder */
        a = b; /* Replace a by b */
        b = r; /* Replace b by r */
 }
```

Computing the Harmonic Numbers

- Computation of harmonic numbers using for loops is quite simple:

```
H = 0;
for (i=1; i<=n; ++i)
    H += 1.0/i;
printf("H(%d) = %f\n", n, H);
```

For loops with multiple initialization and incrementation statements

- If more than one statements need be executed during the initialization or increment step, they should be separated by commas, since semi-colons indicate separation of the three parts of the loop control area.
- for (i = 2, p1 = 1, p2 = 0; i <= n; ++i , p2 = p1 , p1 = F)

```
    F = p1 + p2; /* Compute Fi from Fi-1 and Fi-2 */
printf("F(%d) = %d", n, F);
```

Loop Invariants

- For verifying the correctness of loops one often uses the concept of **loop invariance**.
- A loop invariant refers to a statement that is true at all instants when the loop condition is checked.
- It may be expressed in terms of one or more variables controlling the flow of the loop.

Example

- Consider the while loop implementation of the computation of H_n.
- i = 0; H = 0;

```
while (i < n)
   { ++i; /* Incremet i */
     H += 1.0/i; /* Update the harmonic number accordingly */ }
```

Here the loop invariant is the statement **"H stores the value Hi for all i=0,1,2,...,n".**

The correctness of this statement can be proved using induction on i.

Another example

```
/* Initialize */
   r2 = a; u2 = 1; v2 = 0; /* Previous-to-previous values */
   r1 = b; u1 = 0; v1 = 1; /* Previous values */
/* Extended gcd loop */
  while (r1 > 0) {
 /* Compute values for the current iteration */
q = r2 / r1; /* Compute the next quotient */
r = r2 - q * r1; /* Compute the next remainder */
 u = u2 - q * u1; /* Identically compute the next u value */
 v = v2 - q * v1; /* Identically compute the next v value */
 /* Prepare for the next iteration */
r2 = r1; u2 = u1; v2 = v1; /* Let the previous-to-previous values be the
     previous values */
r1 = r; u1 = u; v2 = v; /* Let the previous values be the current values */ }
 printf("gcd(a,b) = %d = (%d) * a + (%d) * b\n", r2, u2, v2);
```

Loop Invariant

- Whenever the continuation condition for the above loop is checked, we have:
  - gcd(r2,r1) = gcd(a,b), (1)
  - u2 * a + v2 * b = r2, (2)
  - u1 * a + v1 * b = r1. (3)
- Convince your self that the initial values satisfy these 3 equations.
- Prove the results by induction:
  - gcd(r1,r)=gcd(r2,r1)=gcd(a,b)

Inductive Reasoning

- Moreover,
  - u = u2 - q * u1, and v = v2 - q * v1, and so
  - u * a + v * b

    = (u2 - q * u1) * a + (v2 - q * v1) * b

    = (u2 * a + v2 * b) - q * (u1 * a + v1 * b)

    = r2 - q * r1 = r.

Let us look at the trace of the values stored in different variables for a sample run with a=78 and b=21.

Iteration No	**r2**	**r1**	**u2**	**u1**	**v2**	**v1**	**q**	**r**	**u**	**v**	**u2*a+v2*b**
Before loop	78	21	1	0	0	1	-	-	-	-	78
1	78 21	21 15	1 0	0 1	0 1	1 -3	3 3	15 15	1 1	-3 -3	78 21
2	21 15	15 6	0 1	1 -1	1 -3	3 4	1 1	6 6	-1 -1	4 4	21 15
3	15 6	6 3	1 -1	-1 3	-3 4	4 -11	2 2	3 3	3 3	-11 -11	15 6
4	6 3	3 0	-1 3	3 -7	4 -11	-11 26	2 2	0 0	-7 -7	26 26	6 3

gcd(78,21) = 3 = (3) * 78 + (-11) * 21

The *break* statement

- A loop may be forcibly broken from inside irrespective of whether the continuation condition is satisfied or not. This is achieved by the break statement.
- while (1)

```
{ if (b == 0) break;
  r = a % b; a = b; b = r;
 } printf("gcd = %d\n", a);
```

Infinite loops with break

- The do-while loop:

  do { execute loop body; } while (continuation condition is true);

  is equivalent to
- do { execute loop body;

  if (continuation condition is false) break; } while (1);
- while (1) { execute loop body;

  if (continuation condition is false) break; }

Cmputing sum of gcd(a,b), a<=b<=20

```
/* Initialize sum */
sum = 0;
for (i=1; i<=20; ++i)
   { for (j=i; j<=20; ++j)
      { /* Now we plan to compute gcd(j,i) */
       /* But we must not disturb the loop variables */
      /* So we copy j and i to temporary variables a and b and change those copies */
       a = j; b = i;
     /* The Euclidean gcd loop */
   while (1)
    { if (b == 0) break; /* gcd computation is over, so break the while loop */
     r = a % b; a = b; b = r; }
  /* When the while loop is broken, a contains gcd(j,i). Add it to the accumulating
     sum. */
    sum += a;
  }/*end inner for loop*/
}/*end outer for loop*/
printf("The desired sum = %d\n", sum);
```

An obfuscated code

```
sum = 0; /* Initialize sum to 0 */
 i = 0; /* Initialize the outer loop variable */
while (1 != 0) { /* This condition is always true */
 j = ++i; /* Increment i and assign the incremented value to j */
if (j == 21) break; /* Break the outermost loop */
while (3.1416 > 0) { /* This condition is always true */
 a = j; b = i; /* Copy j and i to temporary variables */
while ('A') { /* This condition is again always true, since 'A' = 65 */
 r = a % b; /* Compute next remainder */
  if (!r) break; /* Break the innermost loop */
    a = b; /* Adjust a and b and */
    b = r; /* prepare for the next iteration */ }
 /* End of innermost loop */
```

```
sum += b; /* Add gcd(j,i) to the accumulating
   sum */
    if (j == 20) break;
    /* Break the intermediate loop */
     ++j; /* Prepare for the next value of j */ }
     /* End of intermediate loop */ }

     /* End of outermost loop */
   printf("The desired sum = %d\n", sum);
```