Loops

Chapter 5

for Loop
Loop controls in one location

- Test at top
  - May not execute
- Any expression may be omitted
- Expression 1 is the initializer
  - Executed only once
- Expression 2 is the loop test
  - Loops while expression 2 is true
  - Tested after expr 1
  - Tested after expr 3
- Expression 3 is the update
  - for (expr-1; expr-2; expr-3) for (i = 0; i < 10; i++)
  - statement;

```
ct 2200, Chapter 5 Slide 2 of 8

for Loop Examples
Simple counting
int i, j;

for (i = 0; i < 10; i++)
  printf("%d\n", i);  /* prints 0 - 9 */

for (i = 0; i < 10; i += 2)
  printf("%d\n", i);  /* prints 0, 2, 4, 6, 8 */

for (i = 0, j = 0; i < 10 && j < 5; i += 2, j++) /* comma operator */
  printf("%d%d\n", i, j);

for (i = 79; i >= 0 && s[i] == '\n'; i--) /* null statement */
  ;
```

```
ct 2200, Chapter 5 Slide 3 of 8

for Loop Variations
Non-"standard" loops

for (;;)
  statement; /* infinite loop */

for (i = 0; s[i] == '\n'; i++)
  statement; /* initializes i */

for (; s[i] != 't'; i++)
  statement; /* empty init */

List* L;
for (L = root->next; L != root; L = L->next)
  statement; /* loop doesn't */
  /* have to work */
  /* with integers */
```

while Loop
Controlled repetition

- Test at the top
  - May not be executed
- Loops while expression is true
  - while (expression)
  - statement;

```
ct 2200, Chapter 5 Slide 5 of 8

int n = 100;
while (n > 0)
  printf("%d\n", n--);

while((c = fgetc(stdin)) != EOF)
  {
    ...
  }
```

do-while Loop
Less common loop

- Test at the bottom
  - Executed at least once
- Loops while expression is true
  - Opposite of Pascal's repeat-until
- Useful when the test expression is initialized by a statement in the loop body

```
ct 2200, Chapter 5 Slide 6 of 8

while (n != 0)
  {
    scanf("%d", &n);
    while (n != 0);
  }
```
Loop Interruption

Used to simplify program structure

- **break**
  - Terminates the inner most loop (execution resumes with the statement following the loop)
- **continue**
  - Skips remaining code in inner loop (from the continue statement to the end of the loop)
  - Starts next loop iteration
    - For loops resume at the update expression followed by the test expression
    - Do and do-while loops resume at the test expression
  - Usually in an **if**-statement

```c
for (avfil = optind; avfl < argv; avfl++)
    if (!fopen(argv[avfil], "r")) == NULL)
        printf("ERROR: unable to open "argv[avfl]");
        continue; /* try to open the next file */
    }
```

Some common idioms

```c
for (; ;)
    /* from qsort partition */
    while (avl++ < v)
        while (avl[++] > v)
            if (avl <= i)
                break;
            temp = avl[ ];
            avl[ ] = avl[ ];
            avl[ ] = temp;
    }
```

for (avfl = optind; avfl < argn; avfl++)
    if (!fopen(argv[avfl], "r") == NULL)
        printf("ERROR: unable to open "argv[avfl]");
        continue; /* try to open the next file */
    }
```