Repetition (Loops)

• Want to do some repetitive sequence of actions:
• print vertical line of *s
  *
  *
  *
  *
  *
• Corresponding program:
  printf("*\n");
  printf("*\n");
  printf("*\n");
  printf("*\n");
  printf("*\n");

Outline

II. Program Basics
H. Statements
6. Loops
   Pretest: While
   Posttest: Do-While
   Pretest: For
   Parts: termination condition, initialization, body
   Pretest vs Posttest
   Counter-controlled vs Event-controlled
   Infinite Loops
   Nested Loops

Repetitions with Loops

PreTest vs. PostTest Loops

Types of Loops

• Pretest - a logical condition is checked before each repetition to determine if the loop should terminate
  – while loop
  – for loop
• Posttest - a logical condition is checked after each repetition for termination
  – do-while loop

Terminating Loops

• Counter-controlled loops - a loop controlled by a counter variable, generally where the number of times the loop will execute is known ahead of time
• Event-controlled loops - loops where termination depends on an event rather than executing a fixed number of times
Counter-Controlled Loops

- Generally with for loops
- Can generally infer number of repetitions from code
- Examples:
  - read 5 numbers
  - print 7 items
  - sort n items

Event-Controlled Loops

- Generally with while, do-while loops
- Can’t infer number of repetitions from program
- Examples:
  - read until input ends
  - read until a number encountered
  - search through data until item found

Parts of Loop

- Initialization - commands to set up loop (set counter to initial value, etc.)
- Terminating condition - logical condition that is checked to terminate loop
- Body of loop - commands repeated
  - action(s) - statements to repeat
  - update(s) - statements to update values associated with loop (counters, etc.)

Parts of Loop Example

Init: set counter to 0
Termination: counter < 5
Body:
Action: print *
Update: add 1 to counter

Importance of Update

What if command `counter++;` left out?

Counter never becomes >= 5.
Termination Condition never met.

Infinite Loops

- Loop starts but termination condition never met:
  - you forget to increase counter
  - user never enters terminating data item
  - etc.
- Results
  - program may stop (doing nothing repeatedly)
  - computer may repeatedly print some data out
Termination Conditions in C

• Loops in C always continue when the termination condition is true and end when the condition is false
• Conditions can be rephrased if needed (positive termination conditions can be negated)
• Condition only checked at fixed points (does not have to hold true during body)

Pretest Loop: While

Syntax:

```c
while (condition) 
statement;
```

Corresponds to:
```c
if (!condition) 
DONE 
statement   
... 
```

Example of While

```c
int counter = 0;
while (counter < 5) {
    printf("\n");
    counter++;
}
```

Executing More Than One Stmt

• In C, loops repeat one statement
• Generally we always use a compound statement as that statement:
```
while (condition) {
    /* body */
}
```
• Useful even when body has one or no statements

Empty Statements and Loops

• What’s wrong with this?
```c
counter = 0;
while (counter < 5);
{
    printf("\n");
    counter++;
}
```
• Note the ; after the condition in the while, its an empty statement (which is the body of the while), so the while is an infinite loop

Event-Controlled While

• While loops are often used to test for the occurrence of events that terminate loops
• Example:
  - read in a set of numbers until a particular value is encountered
  - along the way count the set of numbers and the total of the numbers
  - print out the average of the numbers
  - does not terminate after a fixed point
Calculate Average Loop

```
total = 0;
counter = 0;
GetFirstNumber
while (number != -999) {
    total += number;
counter++;
    printf("Please enter next number: ");
    scanf("%d",&number);
} 
printf("Average is %.3f\n",(float) total / count);
```

Using a Sentinel

- The value -999 is sometimes referred to as a sentinel value
- The value serves as the “guardian” for the termination of the loop
- Often a good idea to make the sentinel a constant:
  ```
  #define STOPNUMBER -999
  while (number != STOPNUMBER) ...
  ```

Compound Conditions

- Often the termination condition is compound:
  ```
  ans = 'N';
  while (!(ans == 'Y') || (ans == 'y')) { 
    printf("Enter id# and salary: ");
    scanf("%d %f",&id,&salary);
    printf("You entered id# %d and salary \$%.2f, Is this correct? (Y/N) ",id,salary);
    scanf(" %c",&ans);
  }
  ```

Making Sure Loop is Entered

- Note in previous loop, we had to set variable ans to an initial value, ‘N’
- This is because a while loop tests its condition before entering the loop, and if the condition is already false, the loop never executes
- Sometimes it is useful to have a loop that always executes at least once

Posttest Loop: Do-While

```
Syntax:
do {
    statement(s)
} while (condition);
Corresponds to:
if (!condition) DONE
statement
if (condition) DONE
... 
```
Using the Do-While

do {
    printf("Enter id# and salary: ");
    scanf("%d %.2f", &id, &salary);
    printf("You entered id# %d and salary $%.2f, is this correct? (Y/N) ", id, salary);
    scanf(" %c", &ans);
} while (!(ans == 'Y' || ans == 'y'));

• Loop always executes at least once

Menu Loop

do {
    showOptions();
    printf("Select option:");
    scanf(" %c", &optn);
    execOption(optn);
} while (!((optn == 'Q') || (optn == 'q')));

Menu Options

void showOptions() {
    printf("A)dd part to catalog
    R)emove part from catalog
    F)ind part in catalog
    Q)uit
");
}

Executing Options

void execOption( char option ) {
    switch (option) {
        case 'A': case 'a': addPart(); break;
        case 'R': case 'r': delPart(); break;
        case 'F': case 'f': fndPart(); break;
        case 'Q': case 'q': break;
        default: printf("Unknown option %c\n", option); break;
    }
}

While vs Do-While

• Differences
  – where condition tested:
    • while (first) - may execute 0 times
    • do-while (last) - must execute at least one time

• Similarities
  – one statement executed
  – initialization before loop
  – update during loop
Pretest Loop: For

- Initialization included in loop header
- Update included in loop header
- Header also includes update
- Syntax:
  
  ```
  for (init; condition; update) 
  statement;
  ```

- Generally for loops expected to execute fixed number of times (counter-controlled)

For Loop

- Syntax:
  
  ```
  for (init; condition; update) 
  statement;
  ```

- Init: assignments to counter variables
- Update: changes to counter variables

For Example

- Printing vertical line of stars:
  
  ```
  counter = 0;
  counter < 5; 
  counter++;
  printf("\n");
  ```

For Example - Sum of 1 to N

```
printf("Number to sum to: ");
scanf("%d", &N);
total = 0;
for (I = 1; I <= N; I++)
total += I;
/* total is now 1 + 2 + ... + N */
```  

For Example - Max of N Scores

```
printf("Number of students: ");
scanf("%d", &NumStudents);
for (I = 0; I < NumStudents; I++) {
    printf("Enter student score %d: ");
    scanf("%d", &score);
    if (score > max)
        max = score;
}
/* max is highest score entered */
```  

The Comma Form

- Possible to evaluate more than one expression (assignment) in initialization or update by separating each by commas
- Syntax: expression, expression, ...

```
printf("Number to sum to: ");
scanf("%d", &N);
for (total = 0, I = 1; 
    I <= N;
    total += I, I++);
/* total is now 1 + 2 + ... + N */
```
Directions of Counting

```c
for (I = 10; I >= 1; I--)
    printf("%d\n", I);
printf("0 BLASTOFF!\n");
printf("Enter start, end, inc values: ");
scanf("%d%d%d", &lstart, &lend, &linc);
for (I = lstart;
    (!linc < 0) && (I < lend) ||
    (!linc > 0) && (I > lend));
    I += linc)
    printf("%d\n", I);
```

Nesting Loops

- It is often useful to have a loop within a loop (called nested loops)
- For example:

```c
printf("Max N! to print: ");
scanf("%d", &N);                /* 1 */
for (I = 1; I <= N; I++) {     /* 2 */
    fact = 1;                    /* 3 */
    for (J = 2; J <= I; J++)     /* 4 */
        fact *= J;                 /* 5 */
    printf("%d! = %d\n", I, fact); /* 6 */
}
```

Tracing Nested Loops

<table>
<thead>
<tr>
<th>Stmt</th>
<th>N</th>
<th>I</th>
<th>J</th>
<th>fact</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

More Efficient Implementation

```c
printf("Max N! to print: ");
scanf("%d", &N);
fact = 1;
for (I = 1; I <= N; I++) {
    fact *= I;
    printf("%d! = %d\n", I, fact);
}
```

- But why does this work?

Another Nesting Example

- How to print: for (I = 6; I >= 1; I--) {
  for (J = 1; J <= I; J++)
    printf("*");
  printf("\n");
}

- Could have 6 be a variable

More Complex Nesting

- How to print: for (I = 0; I <= 5; I++) {
  for (J = 0; J < (11 - 2 * I); J++)
    printf("*");
  printf("\n");
}

- Note 2 (sequential) inner loops
Another Nesting Example

```
Size: 5
*****
****
***
**
*

Size: 7
******
*****
****
***
**
*

Size: 0
```

Nesting and Functions

- With deep nesting it's often a good idea to have inner loops in separate functions.

```c
void drawTri (int N) {
    int I;
    int J;
    if (N > 0) {
        for (I = N; I >= 1; I--) {
            for (J = 1; J <= I; J++)
                printf("*");
            printf("\n");
        }
    } while (N > 0);
}
```

break statement

- The break statement can be used to halt a loop, though it is generally better to use a flag to signal loop should end.

```c
total = 0;
do {
    scanf("%d", &num);
    if (num < 0) break; /* Loop ends */
} while (num != 0);
```

Flags

- Better to use a flag variable - a variable that is set to a value when a condition occurs.

```c
total = 0;
done = 0; /* done is set to 0 state */
do {
    scanf("%d", &num);
    if (num < 0) done = 1; /* done 1 */
} while ((num != 0) && (!done));
```

continue statement

- A continue statement says jump to the next end of the statement list.

- Example:

```c
for (I = 0; I < 100; I++) {
    if ((I % 2) == 1) continue;
    printf("%d is even", I);
}
```

- Again, stylistically, better not to use

Avoiding continue

- Can generally use an if to avoid a continue:

```c
for (I = 0; I < 100; I++) {
    if ((I % 2) == 1) continue;
    printf("%d is even", I);
}
```

- Becomes

```c
for (I = 0; I < 100; I++) {
    if (1 - ((I % 2) == 1))
        printf("%d is even", I);
}
```