Beginning C Programming for Engineers

Lecture 4: Iteration
Expressions as Statements

In C, expressions may be used as statements. Most commonly, this is done with function calls or assignments.

Examples:

\begin{verbatim}
1 + 34;
c - 9;
a = 5;
foo(bar);
\end{verbatim}
Assignment as Operator

Assignment is actually an operator in C. The result of the expression

\[ a = b \]

is \( b \), with the side effect of assigning the value of \( b \) to \( a \). Assignment takes place after other arithmetic.

\[ a = b = c = 1; \]

\[ a = b = 2 * (c = 5); \]
Short-cut Assignments

“Short cut” assignment operators combine an operation with an assignment.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>a += b</td>
<td>a = a + b</td>
</tr>
<tr>
<td>a -= b</td>
<td>a = a - b</td>
</tr>
<tr>
<td>a *= b</td>
<td>a = a * b</td>
</tr>
<tr>
<td>a /= b</td>
<td>a = a / b</td>
</tr>
<tr>
<td>a %= b</td>
<td>a = a % b</td>
</tr>
</tbody>
</table>

For instance, instead of writing:

```c
a = a + 1;
```

you could write

```c
a += 1;
```
Postfix Increment and Decrement

Postfix increment and decrement operators return the original value of the variable, then increment or decrement the variable.

```c
int a, b;
a = b = 10;
printf("%d\n", a++);  /* Prints 10 */
printf("%d\n", a);    /* Prints 11 */
printf("%d\n", b--);  /* Prints 10 */
printf("%d\n", b);    /* Prints 9 */
```
Prefix Increment and Decrement

Prefix increment and decrement operators increment or decrement the variable, then return its resulting value.

```c
int a, b;
a = b = 10;
printf("%d\n", ++a);  /* Prints 11 */
printf("%d\n", a);  /* Prints 11 */
printf("%d\n", --b);  /* Prints 9 */
printf("%d\n", b);  /* Prints 9 */
```

Remember: If the ++ comes before the variable, it increments before computing the result.
Iteration

*Iteration* is repeating the same statements in a program. Sometimes this is called *looping*.

```
while (expression) statement
```

The *while* statement executes *statement* as long as *expression* is true. The *expression* is tested before *statement* is ever executed.
Example: Powers of 2

/* Print powers of 2 less than 1500 */

#include <stdio.h>

int main()
{
    int i;
    i = 1;  /* 2^0 */
    while (i <= 1500) {
        printf("%d\n", i);
        i *= 2;
    }
    return 0;
}
/* Read and test for oddness. */

#include <stdio.h>

int getInt(void);

int main()
{
    int i;
    i = getInt();
    while (i % 2 == 0) {
        i = getInt();
    }
    printf("Odd number: %d\n", i);
    return 0;
}

int
getInt(void)
{
    int v;
    printf("Enter number: ");
    scanf("%d", &v);
    return v;
}
/* Read and test for oddness. */

#include <stdio.h>
#include <stdlib.h>

int getint(void);

int
main()
{
    int i;
    while ((i = getint()) % 2 == 0)
    {
        printf("Odd number: %d\n", i);
        return 0;
    }

    int
getint(void)
{
    int v;
    printf("Enter number: ");
    if (scanf("%d", &v) != 1 || v < 0) exit(0);
    return v;
}
The do ...while Statement

Sometimes (rarely) instead of testing a while condition before executing statements, you want to always execute those statements at least once.

```
do statement while (expression)
```

This will execute `statement`. Then, repeatedly test if `expression` is true; if so, loop again.

Example:

```c
int i;
do {
    printf("Enter number: ");
    scanf("%d", &i);
} while (i % 2 == 0);
```
Iteration and Recursion

Every problem that can be solved using recursion can also be solved using iteration (sometimes not as clearly, other times more clearly).

/* n! using while. */

#include <stdio.h>

int
main()
{
    int i, n, f;

    printf("Enter n: ");
    scanf("%d", &n);

    f = 1; /* 0! */
    i = 1;
    while (i <= n) {
        f *= i; /* Now, f = i! */
        ++i;
    }
    printf("%d! = %d\n", n, f);
    return 0;
}
The for Loop

It is common to have a “control variable” step from a starting value to an ending value, stepping through intermediate values (like computing \( n! \)).

\[
\text{for (initExpr ; testExpr ; stepExpr) statement}
\]

The for statement first evaluates \( \text{initExpr} \). Then, as long as \( \text{testExpr} \) is true, execute \( \text{statement} \). After each time \( \text{statement} \) is executed, evaluate \( \text{stepExpr} \).

```c
int i;
for (i = 1;  i < 10;  i += 2) {
    printf("%d\n", i);
}
```
/* n! using for. */

#include <stdio.h>

int
main()
{
    int i, n, f;

    printf("Enter n: ");
    scanf("%d", &n);

    f = 1; /* 0! */
    for (i = 1; i <= n; ++i) {
        f *= i; /* Now, f = i! */
    }
    printf("%d! = %d\n", n, f);
    return 0;
}
The break Statement

Sometimes we want to end a loop in the middle. Use break to immediately end the innermost loop.

```c
int main()
{
    int i, n;
    printf("Enter n: ");
    scanf("%d", &n);
    for (i = 2; i < n; ++i) {
        if (n % i == 0) break;
    }
    if (i == n) {
        printf("%d is prime.\n", n);
    } else {
        printf("%d is not prime (divisible by %d)\n", n, i);
    }
    return 0;
}
```
Nesting Loops

Loops can be *nested*.

```c
/* Sum up primes. */

#include <stdio.h>

int
main()
{
    int i, n, limit, sum = 0;
    printf("Enter limit: ");
    scanf("%d", &limit);
    for (n = 2; n <= limit; ++n) {
        for (i = 2; i < n; ++i) {
            if (n % i == 0) break;
        }
        if (i == n) sum += n;
    }
    printf("Sum of primes \textless= %d is %d\n", limit, sum);
    return 0;
}
```
The continue Statement

The `continue` statement restarts a loop. In a `while` or `do while` loop, the test runs. In a `for` loop, the increment expression runs, then the test.

```c
/* Test n, reverse(n) for primeness. */
#include <stdio.h>
int reverse(int n);

int
main()
{
    int i, limit, n, r;
    printf("Enter limit: ");
    scanf("%d", &limit);
    for (n = 2; n <= limit; ++n) {
        for (i = 2; i < n; ++i) {
            if (n % i == 0) break;
        }
        if (i != n) continue; /* Not prime */
        r = reverse(n);
        for (i = 2; i < r; ++i) {
            if (r % i == 0) break;
        }
        if (i != r) continue; /* Not prime */
        printf("%d and %d are prime.\n", n, r);
    }
    return 0;
}
```
int reverse(int n)
{
    int r;
    for (r = 0; n > 0; n /= 10) {
        r = 10*r + n%10;
    }
    return r;
}