CSE 1320 - Intermediate Programming

Loops

Alex Dillhoff

University of Texas at Arlington

Loops

Loops allow us to express multiple *iterations* of statements compactly.

We will cover:

- Loop statements
  - `while`
  - `do-while`
  - `for`
- Exiting gracefully

while Loops

Syntax

```
while (EXPRESSION)
    STATEMENTS
```

Simple Example

```
int count = 0;

while (1) {
    printf("%d\n", count++;);
}
```

`while` Loops

- `EXPRESSION` evaluated at the top of the loop.
- Statements in loop are executed.
- Once the bottom is reached, return to the top.
- **Control the loop through the** `EXPRESSION`.

while Loops

```
/* Count to 10 */
int count = 0;

while (count < 10) {
    printf("%d\n", count++);
}
```

Does this program do what was intended?

while Loops

```
/* Count to 10 */
int count = 0;

while (count < 10) {
    printf("%d\n", count++);
}
```

Does this program do what was intended?
Answer: No! It only counts to 9.

while Loops

Let's modify this slightly.

```
/* Count to 10 */
int count = 0;

while (count <= 10) {
    printf("%d\n", count++);
}
```

Does this program do what was intended?

while Loops

Let's modify this slightly.

```
/* Count to 10 */
int count = 0;

while (count <= 10) {
    printf("%d\n", count++);
}
```

Does this program do what was intended?
Answer: Yes! It now includes 10.

`while` Loops

Example: Is Prime? (`is_prime.c`)

`while` Loops

Example: Guessing Game (`guess.c`)

`while` Loops

Example: System Menu (`menu.c`)

for Loops

for loops provide a convenient syntax for looping a specified number of times.

Syntax

```
for (INIT.; CONDITION; PROCESSING)
STATEMENTS
```

Simple Example

```
/* Count to 10 */
for (int i = 0; i < 10; i++) {
    printf("%d\n", i);
}
```

`for` Loops

- **Initialization** - Allows us to create the loop counting variable.
- **Condition** - Set the test condition for which the loop should continue or stop.
- **Processing** - Defines what should happen after each iteration of the loop.

`for` Loops

Example: Multiples of 3 and 5 (`multiple.c`)

do-while Loops

do-while loops guarantee a single iteration of the loop.

Syntax

```
do
STATEMENTS
while (CONDITION)
```

do-while Loops

Example: Guessing Game Again (`guess2.c`)

Infinite Loops

Infinite loops are most common with `while` loops.

- Make sure the condition can be broken.
- Remember to update your loop counter (if applicable).
- Use **control statements**.

Infinite Loops

With a `while` loop:

```
while (1);
```

Infinite Loops

With a `for` loop:

```
for (;;);
```

Nested Loops

- Any amount of loops can be nested.
- Increases the computation time.
- Useful for having an outer control loop to keep the user in a program.

Nested Loops

EXAMPLE: Prime Factorization
(`prime_factor.c`)

Additional Control

Additional control is available with loops through the following statements.

- `break;`
- `continue;`
- `return;`
- `exit();`

Additional Control – `break`

The `break` statement immediately exits a loop.

If the loop is the inner loop of a nested loop, it will return control to the outer loop.

Additional Control – break

```
while (!found) {

    // Break if target found
    if (input == target) {
        break;
    }

    input++;
}
```

Adding Control – `continue`

The `continue` statement skips to the bottom of the loop.

This is commonly used to skip unnecessary calculations depending on the data.

Adding Control – continue

```
// Don't divide by anything
// that is divisible by 11
for (int i = 0; i < n; i++) {
    if (i % 11 == 0)
        continue;

    input /= i;
}
```

Adding Control – `return`

The `return` statement immediately exits the current function.

If executed in `main`, the program exits.

Adding Control – `exit`

The `exit()` function will immediately exit the program, regardless of where it is executed.

There is typically always a better way to exit the functions and program without it.

Adding Control – exit

```
int main() {
    for (int i = 0; i < 10; ++i) {
        if (i == 5) {
            exit();
        }
    }

    return 0;
}
```