ECS30: `while`, `for`, nested loops

Dr. Rob Gysel
1/25/17

Review Session

Tonight 7pm @ Rock Hall

- Discuss sample midterm
- Write solutions
- Solutions posted to Piazza after

Overview

- Loops in C
  - `while, for, do-while`
  - Printing with `%s`
- String operations: indexing, terminating, `strlen`
- Comparing strings: `strcmp`
- More on `<ctype.h>`

Loops

`whilefactorial.c`

`while (cond) { … statements ... }`

1. Loop condition
2. Body (if condition `true`)

```
int n, result = 1;
printf("Enter non-negative n: ");
scanf("%d", &n);
int i = 1;
while (i <= n) {
    result = result * i;
    i = i + 1;
}
printf("n! Computed. %d! = %d\n", n, result);
```

Loops

`forfactorial.c`

loop structure changes

1. Initialization
2. Loop condition (if true, repeat body)
3. Update (often increment or decrement)

```
int n, i, result = 1;
printf("Enter non-negative n: ");
scanf(" %d", &n);
/* Compute n! */
for (i = 1; i <= n; ++i) {
    result *= i;
}
/* Print result */
printf("n! Computed. %d! = %d\n", n, result);
```

Prefix / Postfix operators

`i++;` and `++i;`

(Almost) equivalent to

```
i = i + 1;
```

Decrement: `i--;` and `--i;`

Prefix / Postfix operators

- Postfix: `i++;`
  - Increments *after* `i` evaluated
- Prefix: `++i;`
  - Increments *before* `i` evaluated

Common loop patterns

Flag-based loops

```
boolean flag = true;
while (flag) {
  ... // do something
  if (...) { // some condition
    flag = false;
  }
}
```

Common loop patterns

Nested loops

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

Common loop patterns

Nested loops

- *One* nesting (i.e. `for (...) { for (...) {} }`) is common
- Two less common
- 3+: There is probably a better approach (ex: write a function!)

Input Validation

`sanitycheckinput.c`

Validating non-negative `int` input

1. `scanf` returns # of variables read successfully
2. Check that `n` is not negative

```
/* Read n and verify input */
while (!readInput) {
    printf("Enter non-negative n: ");
    if (scanf("%d", &n) == 1 && n < 0) {
        readInput = true;
    }
}
```

Input Validation

`dowhilesanitycheckinput.c`

Validating non-negative `int` input

`do-while` executes body *at least once*

```
/* Read n and verify input */
do {
    printf("Enter non-negative n: ");
} while (scanf("%d", &n) != 1 || n < 0);
```

Flushing input (`stdin`)

User input sent to *standard input* (`stdin`)

```
campus-097-225:Lecture Programs RobsMacAir$ ./sanitycheckinput
Enter non-negative n: f
```

Q: What happens if non-`int` entered?

A: `scanf` looks for `int`, sees non-digit `char` `f`, stops. Leaves `f` in *input buffer* (to be read later)

Flushing input (`stdin`)

Causes infinite loop on non-`int` input:

```
/* Read n and verify input */
while (!readInput) {
    printf("Enter non-negative n: ");
    if (scanf("%d", &n) == 1 && n < 0) {
        readInput = true;
    }
}
```

1. Body of `while`: try to read `int` with `scanf`
2. Fail on non-`int` input, leave input in buffer
3. Condition `false`, so go to 1

Flushing input (`stdin`)

```
campus-097-225:Lecture Programs RobsMacAir$ ./sanitycheckinput
Enter non-negative n: f
non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Ent
er non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n:
Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative
n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negati
ve n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-neg
ative n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-
negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter n
on-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Ente
r non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n: E
nter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n
: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negativ
e n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-nega
tive n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-n
egative n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter no
n-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter
 non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n: En
ter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative n:
 Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negative
 n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-negat
ive n: Enter non-negative n: Enter non-negative n: Enter non-negative n: Enter non-ne
```

Flushing input (`stdin`)

Causes infinite loop on non-`int` input:

```
/* Read n and verify input */
while (!readInput) {
    printf("Enter non-negative n: ");
    if (scanf("%d", &n) == 1 && n < 0) {
        readInput = true;
    }
}
```

1. Body of `while`: try to read `int` with `scanf`
2. Fail on non-`int` input, leave input in buffer
3. Condition `false`, so go to 1

Flushing input (`stdin`)

How to flush `stdin`:

```
// Flush stdin buffer
void flushStdin() {
  char c;
  // Skip all characters until end-of-file marker
  // or new line / carriage return
  while ( (c = getchar()) != EOF &&
          c != '\n' &&
          c != '\r' ) {};
}
```

Flushing input (`stdin`)

How to flush `stdin`:

```
// Flush stdin buffer
void flushStdin() {
  char c;
  // Skip all characters until end-of-file marker
  // or new line / carriage return
  while ( (c = getchar()) != EOF &&
          c != '\n' &&
          c != '\r' ) {};
}
```

Read `char`s in input buffer

Flushing input (`stdin`)

How to flush `stdin`:

```
// Flush stdin buffer
void flushStdin() {
  char c;
  // Skip all characters until end-of-file marker
  // or new line / carriage return
  while ( (c = getchar()) != EOF &&
          c != '\n' &&
          c != '\r' ) {};
}
```

Stops if input ends

Flushing input (`stdin`)

How to flush `stdin`:

```
// Flush stdin buffer
void flushStdin() {
  char c;
  // Skip all characters until end-of-file marker
  // or new line / carriage return
  while ( (c = getchar()) != EOF &&
          c != '\n' &&
          c != '\r' ) {};
}
```

Last `char` is user hitting `<enter>`