Control structures

Stat 580: Statistical Computing

- Theme: Black - White
- Printable version

References

- "The C programming language" by Brain W. Kernighan and Dennis M. Ritchie.
- Part of this slide set is based on *Essential C* by Nick Parlante:

(for copyright reason, this notice is reproduced here.)

`if` statement

Four common forms:

```
if (<expression>) <statement>   /* simple form with no {} or else clause */
```

```
if (<expression>) {     /* simple form with {}'s to group statements */
   <statement>
   <statement>
}
```

```
if (<expression>) {     /* full then/else form */
   <statement>
} else {
   <statement>
}
```

```
if (<expression>) {     /* full then/else form */
   <statement>
} else if (<expression>) {
   <statement>
} else {
   <statement>
}
```

Example

```c
double x, y, min;
x = 1.3;
y = 2.1;

/* if-else */
if (x < y) {
  min = x;
} else {
  min = y;
}
printf("%f\n", min); /* 1.300000 */
```

Conditional expression

The conditional expression can be used as a shorthand for some if-else statements.

```
<expression1> ? <expression2> : <expression3>
```

- expression, not a statement
- If `<expression1>` is true, this expression returns `<expression2>`. Otherwise, it returns `<expression3>`.

```
/* conditional operator */
min = (x<y) ? x : y;
printf("%f\n", min);
```

`switch` statement

- useful when there are many cases (`if ... else if ... else if ... else ...`)

```
switch (<expression>) {
  case <const-expression-1>:
    <statement>
    break;
  case <const-expression-2>:
    <statement>
    break;
  case <const-expression-3>:
    <statement>
    break;
  default:    /* optional */
    <statement>
}
```

`switch` statement

- The switch expression is evaluated, and then the flow of control jumps to the matching `const-expression` case.
- The case expressions are typically `int` or `char` constants.
- `break` is important!

Example 1

```
int n=4;

switch(n) {
  case 3:
    printf("value is 3\n");
    break;
  case 4:
    printf("value is 4\n");
    break;
  case 5:
    printf("value is 5\n");
    break;
  default:
    printf("value is not in any of the cases: %d\n", n);
}
```

Example 2

```
int n=4;

switch(n) {
  case 3:
    printf("value is 3\n");
    break;
  case 4:
  case 5:
    printf("value is 4 or 5\n");
    break;
  default:
    printf("value is not in any of the cases: %d\n", n);
}
```

Example 2

```c
int n=4;

switch(n) { /* n is an integer or a character */
  case 3:   /* error if duplicated */
    printf("value is 3\n");
    break;  /* problematic if break is removed */
  case 4:
  case 5:   /* same statement for case 4 or 5 */
    printf("value is 4 or 5\n");
    break;
  default:
    printf("value is not in any of the cases: %d\n", n);
}
```

`while` loop

```
while (<expression>) {
  <statement>
}
```

- while the `<expression>` is true, the loop continues
- `<expression>` is evaluated before every loop
- may lead to infinite-loop

Example:

```
  j=5;

  while (j--) /* is it an infinite loop? */
    printf("%d\n",j);
```

`do-while` loop

```
do {
  <statement>
} while (<expression>)
```

- `do-while` is a variation of `while` loop, which ensures the loop body is executed at least once.

What are the outputs of the following two loops?

```
  j=5;
  while (j--)
    printf("%d\n", j);


  j=5;
  do
    printf("%d\n", j);
  while (j--);
```

`for` loop

```
for (<initialization>; <continuation>; <action>) {
    <statement>
}
```

- The loop header contains three parts:
  - an initialization
  - a continuation condition
  - an action
- The initialization is executed once before the body of the loop is entered.
- The loop continues to run as long as the continuation condition remains true (like a while).
- After every execution of the loop, the action is executed.

Example

```
int x, y;

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

We can have multiple variables in `<initialization>` and `<action>`:

```
for (x=1, y=5; (x+y<10)&&(y<=5); x++, y++) { /* multiple variables */
  printf("(%d, %d)\n", x, y);
}
```

`break`

```
while (<expression>) {
  <statement>
  <statement>
  if (<"break condition">)
    break;
  <statement>
  <statement>
}
/* control jumps down here on the break */
```

- The `break` statement will move control outside a loop or `switch` statement.
- It's preferable to use a straight `while` with a single test at the top if possible.
- It *does not* work with `if` (does not break `if`), while it usually occurs together with `if`.

continue

- The continue statement causes control to jump to the bottom of the loop, skipping over any code below the continue.

```
while (<expression>) {
  ...
  if (<condition>)
    continue;
  ...
  ...
  /* control jumps here on the continue */
}
```