CS 31 Discussion 1A, Week 3

Zengwen Yuan (zyuan [at] cs.ucla.edu)
Humanities A65, Friday 10:00—11:50 a.m.

Today's focus

- Control flow
  - Boolean expression
  - Branching mechanism
  - Loop
- Answer your questions
  - types, variables, assignments, operators, string

Boolean Expression

- What and why?
  - e.g. `(x < 5) && (y != x)`
  - most branching stmts are controlled by boolean exprs
- logical operators
- precedence rules

Quiz: Valid boolean expression or not?

1. `x = y`
2. `x != y || x`
3. `x < 5 && > 3`
4. `3 < x < 5`

Quiz: Evaluate the following boolean expressions

```
int x = 1, y = 2, z = 3;
```

1. `x >= y`
2. `x == y`
3. `(x - y) > 10`
4. `x < y && y < z`
5. `((x != y) || (x > y)) && (y == z)`
6. `(x != y) || ((x > y) && (y == z))`
7. `x >= !y`

Branching: `if-else` statements

```
if (<expr>)
  stmt1
else
  stmt2
```

```
if (<expr>) {
  // stmts
} else {
  // stmts
}
```

```
if (<expr>) {
  // stmts
} else if (<expr>) {
  // stmts
} else if (<expr>) {
  // stmts
}
// more else ifs...
else {
  // stmts
}
```

Quiz: What's the output?

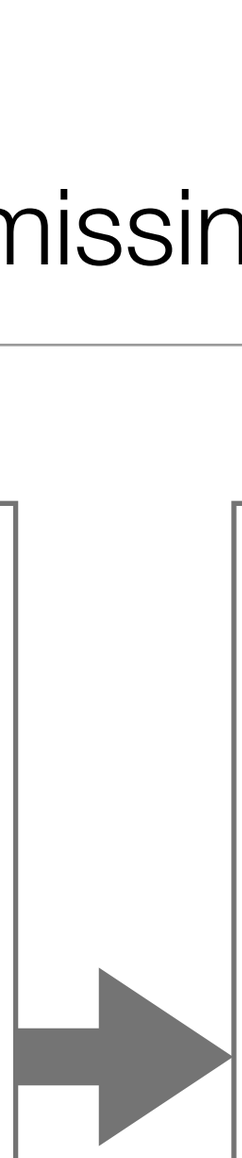
```
int x = 0;
if (x = 0)
  cout << "here\n";
else
  cout << "there\n";
```

Quiz: What's the output?

```
int x = 5;
int y = 1;
if (5 == (x && !y))
  cout << "here\n";
else
  cout << "there\n";
```

Quiz: What are the missing conditions?

```
if (<condA>) {
  /* do A; */
  if (<condB>) {
    /* do B; */
  } else {
    /* do C; */
  }
} else {
  /* do D; */
}
```



```
if (__?__) {
  /* do A; */
}
if (__?__) {
  /* do B; */
} else if (__?__) {
  /* do C; */
} else {
  /* do D; */
}
```

Branching: `switch` statement

```
switch (<expr>) {
  case <const1>:
    // stmts
    break;
  case <const2>:
    // stmts
    break;
  default:
    // stmts
}
```

Quiz: Rewrite the following snippet using `switch`

```
if (code == 281)
    cout << "bigamy";
else if (code == 321 || code == 322)
    cout << "selling illegal lottery tickets";
// other cases...
else
    cout << "some other crime";
```

Loop: `while` statement

- Syntax

```
while (<expr>)
  // stmt
```

```
while (<expr>) {
  /* code block */
}
```

- Be aware of the infinite loop

Loop: `do-while` statement

- Syntax

```
do {
  /* code block */
} while (<expr>);
```

- What's the difference from while loop?
  - code block gets executed AT LEAST ONCE, regardless whether <*expr*> holds
  - semicolon at the end

Quiz: What's the output?

```
int x = 3;
while (x > 0) {
    cout << "hello\n";
    x--;
}
```

Quiz: What's the output?

```
int x = 3;
do {
    cout << "hello\n";
    x--;
} while (x > 0);
```

Quiz: What's the output?

```
int x = 3;
do {
    cout << "hello\n";
} while (x-- > 0);
```

Quiz: What's the output?

```
int x = 3;
do {
    cout << "hello\n";
} while (--x > 0);
```

Loop: `for` statement

- Syntax

```
for (initialization; cond; de/increase)
    // stmt
```

```
for (initialization; cond; de/increase) {
    /* code block */
}
```

- a real example (but a more cliche one…)

```
for (int x = 3; x > 0; x--) {
    cout << x << endl;
}
```

Loop: `for` statement (cont'd)

- Trickier ones…

¯_(ツ)_/¯

```
for (int x = 3; x > 0;) {
    cout << x << endl;
}
```

```
int x = 5;
for (x = 3; x > 0; x--);
    cout << x << endl;
```

```
int x = 5;
for (; x > 0; x--) cout << x << endl;
```

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

Iterate strings

- Example

```
string s = “Hello";
char c = s[4];  // 'o'
for (int k = 0; k != s.size(); k++)
        cout << s[k] << endl;
```

- Operations on `string` type
  - string::size()
  - string::length()

http://www.cplusplus.com/reference/string/string/