DPST1092 23T2 — Extra C

https://www.cse.unsw.edu.au/~dp1092/23T2/

Extra C Not Covered in Live Lectures

Please see Moodle for pre-recorded lecture video that cover this material

Extra C lecture video

Fine Control

Good programming style often dictates ...

- control loops only through termination condition
- use `if` to control if only part of loop body needed
- have one `return` per function
- exit program only via `return` in `main()`

C has constructs to violate all of these

- `break` and `continue` for fine loop control
- multiple `return` statements within a function
- `exit` and `assert` terminate program early
- but, used carefully, can make code easier to understand

Fine Control Example

Example: scan a [N], stop on zero or at end, ignore negative values

```
for (i = 0; i < N; i++) {
   if (a[i] == 0) break;
   if (a[i] < 0) continue;
   sum += a[i];
}

/* vs */

for (i = 0; i < N && a[i] != 0; i++) {
   if (a[i] > 0) {
      sum += a[i];
   }
}
```

Assignment as Expression

Assignments can be treated as expressions returning a value

```
x = 5;           // returns 5
```

Can be useful

```
x = y = z = 0;  // equiv to x = (y = (z = 0))
```

Can be dangerous

```
if (x = 0)       // "test" always fails
```

Assignment as Expression Example

Assignment-as-expression often used to simplify loops, e.g.

```
// read stdin one char at-a-time
while ((ch = getchar()) != EOF) {
   // do something with ch
}
```

rather than

```
ch = getchar();
while (ch != EOF) {
   // do something with ch
   ch = getchar();
}
```

Assignment as Expression Exercise

In the example above, we wrote:

```
// read stdin one char at-a-time
while ((ch = getchar()) != EOF) {
   // do something with ch
}
```

which reads a char into ch and tests it against EOF

What would the following code do?

```
// read stdin one char at-a-time
while  ( ch = getchar() != EOF)  {
   // do something with ch
}
```

Hint: Check precedence table in the C Reference Card or by typing the command

```
man 7 operator
```

Switch Statements

`switch` encapsulates a common type of selection:

```
if (v == C1) {
   S1;
} else if (v == C2) {
   S2;
}
...
else if (v == Cn) {
   Sn;
}
else {
   Sn+1;
}
```

```
switch (v) {
case C1:
   S1; break;
case C2:
   S2; break;
...
case Cn:
   Sn; break;
default:
   Sn+1;
}
```

Note: The expression v must be an integral type

Warning: `break` is critical; if not present, falls through to next case.

Switch Statements

```
if (colour == 'r') {
   printf("Red");
} else if (colour == 'g') {
   printf("Green");
} else if (colour == 'b') {
   printf("Blue");
}
else {
   printf("Invalid Colour");
}
```

```
switch (colour) {
case 'r':
   printf("Red"); break;
case 'g':
   printf("Green"); break;
case 'b':
   printf("Blue"); break;
default:
   printf("Invalid Colour");
}
```

Conditional Expressions

Encapsulates a common usage for an `if`:

```
if (cond) {
   v = Expr1;
} else {
   v = Expr2;
}
```

can be written as

```
v = cond ? Expr1 : Expr2;
```

Examples:

```
x = (y > 0) ? z+1 : z-1;
```

or

```
x = z + ((y > 0) ? 1 : -1);
```

Conditional Exercise

For each of the following

- rewrite it using a conditional expression or
- state why it cannot be rewritten in this way

```
// (a)
if (x > 0)
   y = x - 1;
else
   y = x + 1;
```

```
// (b)
if (x > 0)
   y = x - 1;
else
   z = x + 1;
```

```
// (c)
if (x > 0) {
    y = x - 1;  z = x + 1;
} else {
    y = x + 1;  z = x - 1;
}
```