CSC 347 - Concepts of Programming Languages

Statements and Expressions

Instructor: Stefan Mitsch

Learning Objectives

❓ What should be the basic building blocks of computations?

- Understand different ways of expressing computations
- Understand the importance of sequencing in computations

PL Development

- Assignment: `x := 5`, comparison `x >= y`, conditional `if (x<4) then 2 else 3`
- Develop syntax for picking a random value from an interval
- `x := * from [0,10]`
- Develop syntax for picking a value that satisfies some property
- `x := * s.t. 0 <= x`
- Express ordinary assignment in terms of the property-satisfying assignment above
- `x := 5` is shortcut for `x := * s.t. x=5`
- What else do popular programming languages provide?

Is this C?

```
int f (int x) {
    int y;
    if (x) y=1; else y=2;
    return y;
}
int main() { printf ("%d\n", f(5)); return 0; }
```

- Compile to find out

```
gcc conditional.c & ./a.out
```

Is this C?

```
int f (int x) {
    int y;
    y = if (x) 1 else 2;
    return y;
}
int main() { printf ("%d\n", f(5)); return 0; }
```

Is this C?

```c
int f (int x) {
    int y;
    y = x ? 1 : 2;
    return y;
}
int main() { printf ("%d\n", f(5)); return 0; }
```

Is this C?

```
int f (int x) {
    int y;
    y = { int z=0; while (x>0) {x--; z++;} z }
    return y;
}
int main() { printf ("%d\n", f(5)); return 0; }
```

- How can we make it C?

Is this C?

```
int g (int x) {
  int z=0;
  while (x>0) {
    x--;
    z++;
  }
  return z;
}
int f (int x) {
    int y;
    y = g(x);
    return y;
}
int main() { printf ("%d\n", f(5)); return 0; }
```

Statements and Expressions

Assembly language consists of *statements*

Expressions are a more abstract way of expressing computations

- Roughly:
    - Statements change memory
    - Expressions: should not

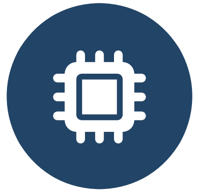
Pure vs Side-Effecting Expressions

- A mathematical function takes arguments and gives results
- An expression is *pure* if that is all it does
- Anything else is a *side effect*
    - Assignment to a variable
    - Change of control (goto)
    - I/O (console, network)
    - etc.

Expressions

- Literals (boolean, character, integer, string)
- Operators (arithmetic, bitwise, logical)
- Function calls

```
f (1 + (2 * strlen ("hello")))
```



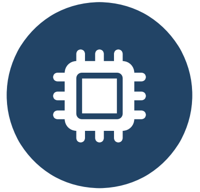
Statements

- Expression statements (including assignment)

```
printf("hello");
^^^^^^^^^^^^^^^    expression
^^^^^^^^^^^^^^^^   statement
```

- Return statements

```
return 1+x;
       ^^^    expression
^^^^^^^^^^^   statement
```



Statements

- Selection statements (if-then-else; switch-case)
- Iteration statements (while; do-while; for)

```
int count = 0;
while (1) {
  int ch = getchar();
  switch (ch) {
    case -1: return count;
    case 'a': count = count + 1;
    default: continue;
  }
}
```

Side-Effecting Expressions

- Post-increment

```
x++
```

- Add and assign

```
x += 2
```

- Assignment

```
x = (y = 5)
```

- Combined

```
x -= (y += 5)
```

Side-Effecting Expressions

```
int x = 1;
printf ("%d\n", ++x);    //
//
```

```
int x = 1;
printf ("%d\n", x++);    //
//
```

```
x = 1 + (y = 5);    //
```

```
int x = 1;
printf ("%d\n", (x = x + 1) + x);    //
```

Side-Effecting Expressions

```c
int x = 1;
printf ("%d\n", ++x);   // pre increment, prints 2
// value of x is now 2
```

```c
int x = 1;
printf ("%d\n", x++);   //
//
```

```c
x = 1 + (y = 5);   //
```

```c
int x = 1;
printf ("%d\n", (x = x + 1) + x);   //
```

Side-Effecting Expressions

```
int x = 1;
printf ("%d\n", ++x);    // pre increment, prints 2
// value of x is now 2
```

```
int x = 1;
printf ("%d\n", x++);    // post increment, prints 1
// value of x is now 2
```

```
x = 1 + (y = 5);    //
```

```
int x = 1;
printf ("%d\n", (x = x + 1) + x);    //
```

Side-Effecting Expressions

```
int x = 1;
printf ("%d\n", ++x);   // pre increment, prints 2
// value of x is now 2
```

```
int x = 1;
printf ("%d\n", x++);   // post increment, prints 1
// value of x is now 2
```

```
x = 1 + (y = 5);  // assigns 5 to y and 6 to x
```

```
int x = 1;
printf ("%d\n", (x = x + 1) + x);   //
```

Side-Effecting Expressions

```
int x = 1;
printf ("%d\n", ++x);   // pre increment, prints 2
// value of x is now 2
```

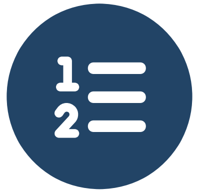
```
int x = 1;
printf ("%d\n", x++);   // post increment, prints 1
// value of x is now 2
```

```
x = 1 + (y = 5);  // assigns 5 to y and 6 to x
```

```
int x = 1;
printf ("%d\n", (x = x + 1) + x);   // no "sequence point", undefined!
```

Side-Effecting Expressions

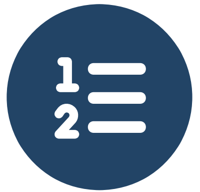
```
int global = 0;

int post_inc () {
  return global++;
}

int main () {
  printf ("%d\n", post_inc () + post_inc ());
}
```



Sequencing

- `(e1, e2, ..., en)`
- `e1` … `en-1` executed for side effect
- Result is the value of `en`
- Sequential execution of expressions!
- An example:

```
string s;
while(read_string(s), s.len() > 5) {
  // do something
}
```



Sequencing

```
int main () {
  int x = 5;
  x *= 2;
  printf ("%d\n", x);
}
```

```
int main () {
  int x = 5;
  printf ("%d\n", (x *= 2, x));
  // behavior defined because comma operator introduces sequence point
}
```

Summary

Statements

- Change memory
- Are executed in sequence

Expressions

- Pure vs. side-effecting
- Sequencing by operator `e1, e2, ... en`
- Conditional by operator `e1 ? e2 : e3`

❓ Should variable declarations be statements or expressions? What are they in C?

❓ Should loops be statements or expressions? What are they in C?