CSC 347 - Concepts of Programming Languages

Strict and Nonstrict Evaluation

Instructor: Stefan Mitsch

Learning Objectives

❓ How much of an expression should be evaluated when computing a result?

- Understand expression evaluation
- Understand function call evaluation
- Understand macro evaluation

Strict and Nonstrict Operators

- A *strict* construct evaluates all of its operands before it runs
- Name some **strict constructs** in C
  - Arithmetic operators: `+`, `-`, `*`, `/`, ...
  - Comparison operators: `<`, `<=`, `==`, `!=`, ...
  - Bitwise operators: `|`, `&`, `~`, `^`
  - Function calls: `f(x,y)`

Strict and Nonstrict Operators

- A *strict* construct evaluates all of its operands before it runs
- Name some **non-strict constructs** in C
  - `e1 && e2` is strict in `e1`, but not `e2`, useful e.g., `p && p->m()`
  - `e1 || e2` is strict in `e1`, but not `e2`, useful e.g., `p==NULL || p->m()`
  - `e1 ? e2 : e3` is strict in `e1`, but not `e2` or `e3`
  - Macro expansion

Ternary Operator

Conditional expression `(e1 ? e2 : e3)`

- Evaluate `e1`; if `true` then evaluate `e2`, else evaluate `e3`

```c
int main () {
  for (int i=0; i<10; i++) {
    int x = 0;
    int y = 0;
    int z = (rand()%2) ? (x=1,111) : (y=2,222);
    printf ("x=%d, y=%d, z=%d\n", x, y, z);
  }
}
```

```
$ gcc statements-01.c && ./a.out
x=1, y=0, z=111
x=0, y=2, z=222
...
```

Conditionals

- Conditional statement vs conditional expression

```
int fact (int n) {
  if (n <= 1) {
    return 1;
  } else {
    return n * fact (n - 1);
  }
}
```

```
int fact (int n) {
  return (n <= 1) ? 1 : n * fact (n - 1);
}
```

What Happens?

- Function calls are strict

```
int fcond (int b, int t, int f) { return b ? t : f; }
int main () {
  for (int i=0; i<10; i++) {
    int x = 0;
    int y = 0;
    int z = fcond (rand()%2,  (x=1,111), (y=2,222));
    printf ("x=%d, y=%d, z=%d\n", x, y, z);
  }
}
```

```
$ gcc statements-02.c && ./a.out
x=1, y=2, z=111
x=1, y=2, z=222
...
```

What Happens?

- Macro calls are non-strict

```
#define mcond(b, t, f) (b)?(t):(f)
int main () {
  for (int i=0; i<10; i++) {
    int x = 0;
    int y = 0;
    int z = mcond (rand()%2,  (x=1,111), (y=2,222));
    printf ("x=%d, y=%d, z=%d\n", x, y, z);
  }
}
```

```
$ gcc statements-03.c && ./a.out
x=1, y=0, z=111
x=0, y=2, z=222
...
```

What Happens?

- Macro calls are evaluated in the compiler, by textual substitution

```c
int ftriple (int i) { return i + i + i; }
#define mtriple(i) (i)+(i)+(i)
int main () {
  int x = 10;
  int rx = ftriple(x=x+1);
  int y = 10;
  int ry = mtriple(y=y+1);
  printf ("x=%d, rx=%d, y=%d, ry=%d\n", x, rx, y, ry);
}
```

```
$ gcc statements-04.c && ./a.out
x=11, rx=33, y=13, ry=36
```

Summary

- Strict evaluation: evaluates **all operands** when determining the value of an expression
- Nonstrict evaluation: evaluates **only necessary operands** when determining the value of an expression