CS240: Programming in C

Lecture 4: Operators and Expressions. Control Flow.

PURDUE UNIVERSITY

```
#include<stdio.h>

int main() {
  int fahr, celsius;
  const int lower = 10, upper = 300, step = 10;

  fahr = lower;
  while (fahr <= upper) {
    celsius = 5* (fahr - 32) / 9;
    printf("%d\t%d\n", fahr, celsius);
    fahr = fahr + step;
 }

  return 0;

}
```

Operators

- Arithmetic: +, -, *, /, %
- Relational: <, >, <=, >=, !=, ==
- Logical: ||, &&, !
- Increment/decrement: ++, --
- Bitwise: |, &, >>, <<, ^, ~
- Assignment: =, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=

Bit operators example

```
int bitcount(unsigned x) {
  int b;

  for (b=0; x != 0; x >>= 1)
     if (x & 01)
          b++;
  return (b);
}
```

Conditional expressions

```
if (a > b)
    z = a;
else
    z = b;

z = (a > b) ? a : b;
```

$expression_1$? $expression_2$:$expression_3$

$expression_1$ evaluated first, then

- if true $expression_2$ is evaluated
- if false $expression_3$ is evaluated

Precedence and associativity

Operators	Associativity
`()  []  ->  .`	Left to right
`!  ~  ++  --  +  -  *  &  (type)  sizeof`	Right to left
`*  /  %`	Left to right
`+  -`	Left to right
`<<  >>`	Left to right
`<  <=  >  >=`	Left to right
`==  !=`	Left to right
`&`	Left to right
`^`	Left to right
`|`	Left to right
`&&`	Left to right
`||`	Left to right
`?:`	Right to left
`=  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=`	Right to left
`,`	Left to right

Unpredictable results

- `s[i] = i++;`
- **Is the subscript the old one or the new one?**
- It is unspecified so different compilers treat this issue differently

```
x = f() + g()
```

It is not specified who is evaluated first f or g

If else

```
if (a > b)
  max = a;
else
  max = b;
```

```
if (expression)
        statement1
else
        statement2
```

If expression is true (non-zero) $statement_1$ is executed
Otherwise $statement_2$ is executed

Nested if-else

- To avoid ambiguity the else is associated with the closest else-less if

```
if (n > 0)
   if (a > b)
      z = a;
   else
      z = b;
```

Always safer to use braces if you're not sure!

Nested if-else: when things go wrong

```
if ( n >=0)
      printf (…);
      some_function(…);
      if(n % 2 == 0) {
       ….
      }
else  /* you mean n is negative*/
```

OOPS! The compiler will associate the else with the closest if (n%2)

Use braces to fix it!

Else-if

```
if (expression1)
      statement1
else if (espression2)
      statement2
else if (espression3)
      statement3
else if (espression4)
      statement4
else
      statement5
```

Example else-if

```
int binsearch (int x, int v[], int n) {
   int low, high, mid;

   low = 0;
   high = n-1;
   while (low <= high) {
        mid = (low+high) / 2;
        if (x < v[mid])
               high = mid - 1;
        else if (x > v[mid])
               low = mid + 1;
        else
               return mid;
   }
   return -1;
}
```

Switch

```
switch (expression) {
   case const_expr: statements
   case const_expr: statements
   case const_expr: statements
   default: statements
}
```

Statements often ends with a **break** statement which causes the exit from **switch.**

Example switch

```
switch (day) {
  case MONDAY:
  case WEDNESDAY:
          prepare_class();
          break;
  default:
          other_stuff();
          break;
}
```

Break and continue

- **`break`**: provides early exist from a **`for`**, **`while`**, **`do`**, and **`switch`**

- **`continue`**: continues the next iteration for a **`for`**, **`while`** or **`do`** to begin

```
for (i=0; i<n; i++) {
 if(a[i]<0)
     continue /* skips negative elements*/
 …
}
```

`switch` and `break`

- **`break`** jumps to the end of the **`switch`** statement.
- If you forget a break statement, the flow of execution will continue right through past the next **`case`** clause.

What will this code print?

```
#include <stdio.h>
int main() {
  char c = 'a';

  switch(c) {
    case 'b': printf("case b: %c\n", c); break;
    case 'a': printf("case a: %c\n", c);
    case 'd': printf("case d:\n");
    default: printf("default \n");
  }
  return 0;
}
```

Go to

- **`goto label`**: makes the program jump to the **`label`**

```
for ()
      for ()
          …
              if(failure)
                goto error;
  error:
            clean up the mess
```

Why you should not use goto in your program
http://www.u.arizona.edu/~rubinson/copyright_violations/Go_To_Considered_Harmful.html

Exam practice questions

- What is the result of the following:

```
   14++ >> 5
4 % 12 & 3
```

- What is the result of

```
printf(" %d    %d\n", x++, x+5);
```

If **x** is 6 before the **printf**

Good coding habits

- **Do not rely on order of operators or function evaluation when writing code.**
- **Always use braces for if else.**
- **Check if you need break in the case statements.**
- **Do not use goto inside your code.**

Readings for this lecture

K&R Chapter 2 and 3