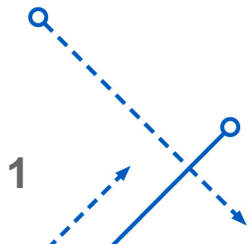
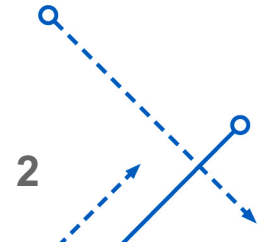
C – Control Flow

Karthik Dantu
Ethan Blanton
Computer Science and Engineering
University at Buffalo
kdantu@buffalo.edu



Conditionals

- True and False in C
  - 0 is false
  - Anything else in true
- Boolean expressions are more predictable
  - True results in 1
  - False results in 0



Boolean Logic

- Boolean operators

  !: Logical NOT; inverts the following expression

  &&: Logical AND: true if LHS and RHS are true

  ||: Logical OR: true if LHS or RHS is true

- C uses short circuit evaluation

  Evaluation of Boolean sentence stops as soon as the final value is known

- For example, `x && y`

  If x is false, y is not evaluated

Short Circuit Consequenes

- While logical, some consequences are surprising
- If terms in an evaluation have side effects, those side effects may not run

```
if (i < len && (array[i]=VAL)) {
   /* code */
}
```

Equality Operators

- Two equality operators

  ==: Compares value equality; returns true if equal

  !=: Compares value equality; returns false if equal

- Operators compare values, not logical truth

  i.e., many values might equate to true, but `true = 1`

  Therefore two logical true values might be unequal

Conditional Examples

```
bool x = true;
int y=2;


if (x)
      printf("x is true\n");
if (y)
      printf("y is true\n");
if (x==y)
      printf("x and y are true\n");
```

Output

```
x is true
y is true
```

stdbool

- The header `#include <stdbool.h>` helps better define Booleans

  Defines type bool which holds only 0 or 1

  The values to true and false respectively

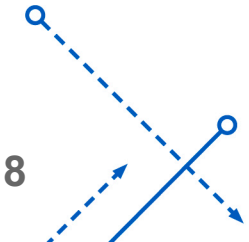
```
bool b = 2;
printf("%d\n", b);
```

OUTPUT

1

Control Flow

- Control flow is the path that execution takes through a program
- The C model is linear by default
- Control flow statements can change the order of execution
- This is how our program makes decisions

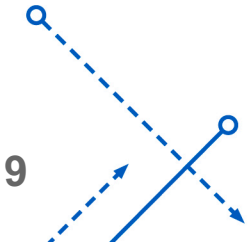


The `if` statement

- The simplest control statement in C is `if`
- Its syntax is

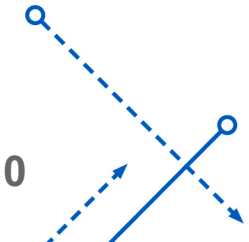
```
if(condition) {
   body;
}
```

- If the expressions condition evaluates to `true`, body runs
- Otherwise, `body` is skipped



Implementing `if`

- The `if` statement is compiled to machine instructions
- Those machine instructions must encode the condition check
- This is normally implemented as a conditional branch instruction
- NOTE: You don't need to learn assembly for this class, but occasionally we'll look at some machine instructions



Condition Example - C

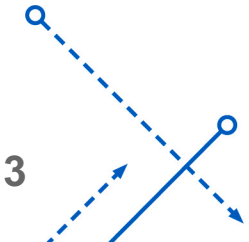
```
int main(int argc, char *argv[]) {
    if (argc ==2 && argv[1][0] == '-')
    {
        printf("negative\n");
    }
    return 0;
}
```

Condition Example - Assembly

```
        cmpl $2, %edi           ; compare argc to 2
        je   .L8                ; jump to .L8 if ==
.L4:
        xorl %eax, %eax         ; set up return value
        ret
.L8:
        movq 8(%rsi), %rax      ; load argv[1][0] into %rax
        cmpb $45, (%rax)        ; compare %rax to 45 ('-')
        jne .L4                 ; jump to .L4 if !=
        leaq .LC0(%rip),%rdi    ; load "negative" into %rdi
        subq $8, %rsp           ; make roon on stack
        call printf@PLT         ; call printf("negative")
```

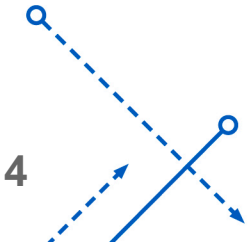
Control Flow in C

- Control flow was lost in machine code
- One of the advantages of high-level languages is structure
- The computer can generally only:
  Make simple comparisions (sometimes only to 0)
  Jump to a program location
- Anything more complicated is a software construction



The `else` clause

- The `else` clause is simply either
  - The next instruction after a jump
  - The jump destination
- Which layer the compiler uses depends on the code and architecture



else Gotchas

- Marking blocks is important in conditionals

```
if (modify_x)
    if (negate)
        x = x * -1;
else
    y = -x;
```

else Gotchas

- What that code actually means is

```
if (modify_x)
    if (negate)
        x = x * -1;
    else
        y = -x;
```

else Gotchas

- The correct code would be

```
if (modify_x) {
    if (negate) {
        x = x * -1;
    }
} else {
    y = -x;
}
```

else if

- Unlike some languages, C does not have an else if statement.
- Instead, we can concatenate `else` and `if`
- This is because `if` is a statement that forms the `else` body.
- Therefore, `else if (...)` is actually `else { if (...)}`!

The `switch` statement

- C provides a convenient multi-case condition statement: `switch`.
- It compares an integer with a set of values.
- The first matching integer value begins execution.

```
switch (integer) {
      case value1:
            body_for_value1 ;
            break;
      case value2:
            body_for_value2 ;
            break;
      default:
            else_body;
}
```

Switch Gotchas

- The break keyword is never implied

```
int i = 0, value = 1;
switch (value) {
      case 1:
            i++;
      case 2:
            i++;
      default:
            i++;
}
printf("%d\n", i);
```

OUTPUT

3

Summary

- All nonzero values are true conditions in C
- All Boolean expressions use 1 for true
- The bool keyword holds only 0 or 1
- C uses short-circuit evaluation of Boolean logic
- if and switch implement conditionals
- Use blocks for if and else
- Control flow is implemented with comparisons and jumps

References

- K&R: 3.1 – 3.4