# Data Structures in C

Prof. Georg Feil

## Branching

Summer 2018

## Acknowledgement

- These lecture slides are based on slides and other material by Professor Magdin Stoica
- Additional sources are cited separately

## Reading Assignments

- C for Programmers (supplementary textbook)
  - Chapter 2 section 2.5: Equality and Relational Operators
  - Chapter 3: Control Statements Part I
  - Chapter 4 section 4.11

## If / Else Statement

condition

```
if (test > 100) {
      // positive outcome statements
      …
}
else {
      // negative outcome statements
      …
}
```

The “if” block

The “else” block
(optional)

Be Proactive!

## Always use blocks with all control statements

## If / Else Common Errors

```
if (number = 77) {
    …
}
```

Not a syntax error in C. This is an assignment operator which is treated as **true if the value is non-zero**.

Always remember to use `==` instead to test for equality

## Statements inside if/else blocks

- The statements inside the “if” or the “else” blocks can be any statements
- Declaring variables inside “if” or “else” blocks
  - Variables declared inside if/else blocks are local to those blocks
  - A variable declared inside an “if” block is not available in the else block, nor outside of the if/else control structure
  - A variable declared inside an “else” block is not available in the if block, nor outside of the if/else control structure
  - If you need a variable to be defined in both blocks, define it before the if/else statement

## Example Prog 1: ‘if’ statement with ‘else’ (chars)

```
#include <stdio.h>

int main (int argc, char** argv)
{
    printf("Please enter a letter of the alphabet: ");

    char letter;
    scanf("%c", &letter);

    if (letter == 'A') {
        printf("You entered A\n");
    }
    else {
        printf("Letter entered was not A\n");
    }

    printf("This runs no matter what\n");  }
}
```

## Advanced Branching

Alternative if, nested if, switch

## Alternative ‘if’ statements

- To test more than one alternative or condition put an ‘if’ statement after the ‘else’ keyword of another ‘if’ statement
  - Note this is not considered nesting

- An ‘else’ block at the very end will run if none of the ‘if’ conditions was true

## Alternative “if” Statements (pseudocode)

```
if (<condition expression 1>) {
   // outcome 1
}
else if (<condition expression 2>) {
   // outcome !1 && 2
}
else if (<condition expression 3>) {
   // outcome !1 && !2 && 3
}
else {
   // outcome !1 && !2 && !3
}
```

## Example Prog 2: ‘if’ statement with ‘else-if’

```
printf("Please enter a letter of the alphabet: ");
char letter;
scanf("%c", &letter);

if (letter == 'A' || letter == 'a') {
    printf("You entered A\n");
}
else if (letter == 'B' || letter == 'b') {
    printf("You entered B\n");
}
else if (letter == 'C' || letter == 'c') {
    printf("You entered C\n");
}
else {
    printf("Letter entered was not A, B, or C\n");
}
printf("This runs no matter what\n");
```

## Nested ‘if’ statements

- Putting an ‘if’ statement in the body of another ‘if’ statement is called nesting

- Nested ‘if’ statements may appear either in the “if” block or the “else” block of another ‘if’ statement

- There is no practical limit to the number of levels of nesting, but try to keep your program readable!

## Nested 'if' statements

```
if (<condition expression 1>) {
    if (<condition expression 2>) {
        // outcome 1 && 2
    }
}
else {
    if (<condition expression 3>) {
        // outcome !1 && 3
    }
}
```

## Example Prog 3a: Nested ‘if’ statement

```
printf("Please enter two letters of the alphabet: ");
char letter1, letter2;
scanf("%c%c", &letter1, &letter2);

if (letter1 == 'A') {
    if (letter2 == 'B') {
        printf("Letter one is A and letter two is B\n");
    }
}
else {
    if (letter2 == 'B') {
        printf("Letter one is not A and letter two is B\n");
    }
}
printf("This runs no matter what\n");
```

## Ternary conditional operator

- The ternary operator ? is a special ‘if’ expression that gives you one value if the condition is true, and another value if the condition is false, example:

```
  int num = (inp > 30 ? 2 : 1);
```

- This is a handy shortcut for:

```
int num;
if (inp > 30) {
    num = 2;
}
else {
    num = 1;
}
```

## The switch statement

## The switch Statement (pseudocode)

```
switch (<expression>) {

        case <value 1>:
                <statement 1.1>;
                <statement 1.2>;
                …
                break;

        case <value 2>:
                <statement 2.1>;
                <statement 2.2>;
                …
                break;

        // as many cases as you need

        default:
                <default statement 1>;
                <default statement 2>;
                …
                break;
}
```

Keyword introduces the value to test

Keyword to end the case

Keyword to introduce each possible value and its outcomes

Notice the colon

Keyword to define what happens if expression value does not match any case ‘switch’ requires

an integer expression

(includes characters ‘char’)

## Example: switch by month

```
int month = …;

switch (month) {
case 1:
    printf("January\n");
    break;

case 2:
    printf("February\n");
    break;

case 3:
    printf("March\n");
    break;
…

case 12:
    printf("December\n");
    break;

default:
    printf("Invalid month number\n");
    break;
}
```

## Fall-Through switch Cases

- Fall-through cases are cases that are not terminated by a “break” statement.
- The statements are executed beginning with a matching case and continue  until the first “break” statement is encountered
- If a “case” block doesn’t contain a break, *program flow will continue with the statements of the next case in the sequence*

## Example: Switch with fall-through

```
int month = …;

switch (month) {

case 1:
case 2:
case 3:
    printf("First Quarter");
    break;

case 4:
case 5:
case 6:
    printf("Second Quarter");
    break;

…

default:
    printf("Invalid month number");
    break;
}
```

## Common switch Errors

- Forgetting a “break” for a case
  - Causes a logic error since the program flow continues with the next case

- Not having a “default” case
  - Errors happen! If there is no “default” then no statements inside the switch will execute when no case matches, hiding the error
  - Always have a default even if all it does it prints an error message or signals an error through other means

- Using a string or double/float expression instead of an integer expression
  - Causes a syntax error or runtime error

Always define a “default” case

## How to choose between if-else and switch

- Use a switch statement in the following situations:
  - You are testing specific numbers
  - You are testing specific characters, for example letters of the alphabet
  - There are many integer values to test (more than 3)
- Use if-else in the following situations:
  - You are testing conditions that involve >, <, or !=
  - The same action is needed for a range of values, for example all integers between 100 and 200
  - You are testing data types which can’t be used in switch
    - e.g. double, float, strings
  - There are only a few values to test (3 or fewer)

## Commenting Control Structures

- All control structures should contain at least a comment above explaining WHY is the logic branched
  - Explain the reason in your own words, don’t just state the obvious
  - Good Example: // Test the number of hours worked to check for
    // overtime
  - Bad Example: // Compare _hoursWorked with 40
- Comments inside the if / else / case blocks:
  - Each case can have a comment to reiterate what condition caused those statements to execute
  - Examples: “regular hours, paid according to hourly pay”, “overtime hours, paid 1.5 times regular hourly pay”.
- Control structure comments are mandatory for good programming style (see coding standards)