Introduction to C

Tutorial 5: conditional statements
Last week…

- Arithmetic expressions
- Logical expressions
- Order of operations
Agenda

- ‘If’ statements
- Switch statements
If statements
If statements: motivation

• Up until this point, we have used logical statements to store/compute values (0 and 1, or false and true, respectively.)

• However, we want to make our program perform differently based on different checks.

• For example: if $x > 54$, then print “success”. Else, print “failure”.

• In C, this can be executed with an if statement. Also called a “conditional” statement
The structure of an ‘if’ statement is simple:

- **expression** can be any expression
  - The parentheses are required. They are part of the structure of an if statement.
- If the expression is not zero, then the statement will get executed.
- If the expression is zero, then the statement will not get executed and the program will continue after the if statement.
- Example:
  
  ```c
  if (x < 55) printf("You have failed!");
  ```

Important to remember that this results in a 0 or 1. In general any expression can appear here!
float temp;
int is_liquid;
printf(“Enter temperatue: ”);
scanf(“%f”, &temp);
is_liquid = (0 < temp) && (temp < 100);
if (is_liquid) printf(“In liquid state.”);
if (temp < 0) printf(“Frozen.”);
if (temp > 100) printf(“Boiled.”);
Another example

```c
int grade;
printf("Enter test grade: ");
scanf("%d", &grade);
if (grade < 55) printf("Failed.");
if (grade >= 55) printf("Passed.");
```

• Both of these checks (comparing grade to 55) complement each other:
  – If one condition is satisfied, then the other won’t be – and vice versa..
  – This (If...else...) is very common.
Because complementary conditions are so common, C has a special feature for this that eliminates the need for 2 tests.

If \( expression \) is true, then \( statement_1 \) will be executed.
Else \( statement_2 \) will be executed.
Either way, **exactly one** of those statements will be executed.
```c
int grade;
printf("Enter test grade: ");
scanf("%d", &grade);
if (grade < 55)
    printf("Failed.");
else
    printf("Passed.");
```

The check only happens once!
More than one instruction

- Based on a condition, we’ll often want to execute more than one line of code.
- What will the next program print?

```c
int grade = 25;
printf("The grade is %d\n", grade);
if (grade > 54)
    printf("Passed.\n");
    printf("Very good!\n");
```

- What happened here?
Still, more than one instruction

• How can we execute more than one statement based on an if statement?
• The answer – using braces.
• Using braces, we can “group” several statements together that will only get executed according to the if statement.
• Example:
  ```c
  int grade = 25;
  printf("The grade is %d\n", grade);
  if (grade > 54)
    {
      printf("Passed.\n");
      printf("Very good!\n");
    }
  ```
Within the “else” portion

if (grade > 54) {
    printf(“Passed.\n”);
    printf(“Very good!\n”);
} else {
    printf(“Failed.\n”);
    printf(“What a pity!\n”);
}

A group of statements within braces is called a block.
Also for a single statement...

- Braces can also be used for a single statement
- Most say that this is a much clearer and more readable way.
- Example:

```c
if (grade < 55) {
    printf("Failed.");
} else {
    printf("Passed.");
}
```
Incorrect Indentation

- What will the next program print when grade = 40?

```c
if (grade < 55)
    if (grade > 52)
        grade = 55; /* factor */
else
    printf("Passed without factor.");
```

- The `else` belongs to the last `if` (of `grade>52`)!
- Therefore the program will print “Passed without factor.”
The fix

How can we make the `else` belong to the first `if`?

Using braces:

```c
if (grade < 55) {
    if (grade > 52)
        grade = 55;
}
else
    printf(“Passed without factor.”);
```

• Now the program will print “Passed without factor.” only when `grade`>=55.
• The following code comes from a program that controls a light aircraft.
• If the altitude is above 10,000 ft. and speed is less than 100 mph, then there is danger and a warning should be printed.
• **Can you find a simpler way to write the code?**

```c
if (altitude > 10000) {
    if (airspeed < 100) {
        printf("Warning! Crash danger! ");
    }
}
```
Another riddle

- In the same program, the visibility variable indicates the degree of visibility in the skies
- The value is always nonnegative (0 or positive)
- If the value is 0, then the red light must be turned on
- **How can this be rewritten?**

```cpp
if (visibility == 0)
{
    turn_on_red_light();
}
```
Riddle: the impact of a short-circuit on an if statement

int x;
scanf("%d", &x);
if (x != 0) {
    x += 2;
    if (x || ++x) {
        x++;
    }
}
printf("%d\n", x);

- What will the next program print...
- If the user provided 7 for x?
- If the user provided 0 for x?
- If the user provided -2 for x?
Example: Calculator
We’ll now try to program a pocket calculator

The user will enter

– A number (x)
– Type of calculation (+ or − or * or /)
– An additional number (y)

The program will print the result of the calculation on x and y.
Input stage

• First, we ask the user to enter the values.

double x, y, result;
char op;

printf("Operation: ");
scanf("%lf%c%lf", &x, &op, &y);
Calculation stage

• If the calculation was an addition, we execute an addition.
• Else, if the calculation was a subtraction, we execute a subtraction.
• Else, if the calculation was a multiplication, we execute a multiplication.
• Else, if the calculation was a division, we execute a division.

• And now, in C...
if (op == ‘+’) 
    result = x + y;
else 
    if (op == ‘-’) 
        result = x - y;
else 
    if (op == ‘*’) 
        result = x * y;
else 
    if (op == ‘/’) 
        result = x / y;
else {
    printf("Error, unknown operator.");
    return 1;
}
Calculation stage (more readable)

```c
if (op == ' + ')
    result = x + y;
else if (op == ' - ')
    result = x - y;
else if (op == ' * ')
    result = x * y;
else if (op == ' / ')
    result = x / y;
else
{
    printf("Error, unknown operator.");
    return 1;
}
```

What did we change? Only the spaces. From the compiler’s perspective, we didn’t change anything. From a **readability** perspective, there’s a world of difference.
Printing the result

• And now that we’ve calculated the result, we need to print it.

```c
printf("\%lf %c \%lf = \%lf\n", x, op, y, result);
return 0;
```
And the result

Operation: 19.3 / 4.55
19.300000 / 4.550000 = 4.241758
switch statements
If..else if...else if...else if...

• In the calculator program we encountered something that’s very common

• A series of value checks on one variable
  – In this case, it was the variable `op`, which was of type `char`

• C has a special structure to make dealing with cases like these easier: `switch statements`. 
Switch statements

• The basic structure

```java
switch (expression) {
    case v₁:
        op₁₁;
        op₁₁b;
        :
        break;
    case v₂:
        op₂₁;
        op₂₁b;
        :
        break;
    :
    :
}
```

1. The expression must be a whole number (int or long or char).
2. The value of case must be a constant – no variables.
3. The value of case cannot appear more than once.
The switch statement works as follows:

A. Compute the value of the expression *(value)*.

B. Compare the *value* to the value in the first case *(v₁)*.
   - If they are equal: execute instructions *op₁a, op₁b*, etc. until the **break**.
   - Then move past the **switch** statement.

C. Compare the *value* to the value in the second case *(v₂)*.
   - If they are equal: execute instructions *op₂a, op₂b*, etc. until the **break**.
   - Then move past the **switch** statement.

Etc.

There is no limit on how many “cases” there are in a switch statement.
Example: Using switch in the calculator program

```c
switch (op) {
    case '+':
        result = x + y; break;
    case '-':
        result = x - y; break;
    case '*':
        result = x * y; break;
    case '/':
        result = x / y; break;
}
```
Default option

• And what if none of the options work?
• In the if-else-if structure, we could end with an else statement without a condition.
  – This meant “for all other cases”
• In switch statements, you can add a “default” section after the last case statement.
  – This will work “for all other cases”
Example: Default option

```c
switch (op) {
    : 
    : 
    case '*':
        result = x * y;
        break;
    case '/':
        result = x / y;
        break;
    default:
        printf("Error, unknown operator.");
        return 1;
}
```
Two options, same result

• In a switch statement, we might have a case where we have two (or more) options but we want the same result.

• Two (or multiple) case lines, one after the other
  – After the first one, do not put a break.
  – After the second (or last) one, continue as normal.

• With a regular condition it looks like the following:

```c
... 
else if (op == ‘*’ || op == ‘x’) 
  result = x * y;
else if ...
```
Example – two identical options

switch (op) {
    case ‘+’:
        result = x + y;
        break;
    case ‘-’:
        result = x - y;
        break;
    case ‘x’:
    case ‘*’:
        result = x * y;
        break;
    case ‘/’:
        :
        :
}

• In this example, we allow a lowercase “x” to be used as a substitute for the multiplication symbol “*” in our calculator
switch vs. if

- Remember that switch and if are logically equivalent statements.
  - We saw that every switch statement can be rewritten as if – else – if – else ....
  - But every if statement can also be rewritten as a switch using the following trick:

  ```
  switch ((expression) != 0)
  {
    case 1:
    
    statement_1
    
    break;
    
    default:
    
    statement_2
  }
  ```
How to choose between if and switch?

• If you want to write code that differentiates between two states, namely "if the condition is true - do X, else do Y" - **if** is better.

• If you want to write code that differentiates between many states at once - we prefer to use switch.

• For example, identifying the operation character stored in the variable (op).
  • If ‘+’ execute an addition
  • If ‘-’ execute a subtraction
  • If ‘*’ execute a multiplication
  • If ‘/’ execute a division
Example: Competition between restaurants

We want to have a survey on restaurants in the Technion.
3 restaurants are participating:
1) Greg – G/g
2) Chemistry – C/c
3) Student’s union – S/s

Write a program that counts the votes of students.
Every vote is entered as the first letter of the name of the restaurant (uppercase or lowercase). The survey is completed when someone enters a “q”.
Additional example – Competition between Technion restaurants

```c
#include <stdio.h>
int main(void) {
    int Gscore=0, Sscore=0, Cscore=0;
    char c;
    printf("Enter your favorite restaurant in the Technion:\n");
    printf("Greg Coffee (near Taub) press - G\n");
    printf("Student house press - S\n");
    printf("Chemical Engineering - C\n");
    printf("if you want to end the voting please press q\n");
```
while( (c=getchar()) != 'q') {
    switch (c) {
        case 'g':
            case 'G': Gscore++; break;
        case 's':
            case 'S': Sscore++; break;
        case 'c':
            case 'C': Cscore++; break;
        default: printf("No inventing new restaurants!
");
    } /* end of switch */
} /* end of while */

while causes the code to continue running as long as the condition is met.
printf("\nThe scores of the restaurants are as follows:\n");
printf("Greg Coffee: %d\n", Gscore);
printf("Student house: %d\n", Sscore);
printf("Chemical Engineering: %d\n", Cscore);

if (Sscore>Gscore && Sscore>Cscore)
    printf("Impossible... Logical error!!! \n");
} /* end of main */