CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 4.2: Advanced selection and examples

Toronto Metropolitan University

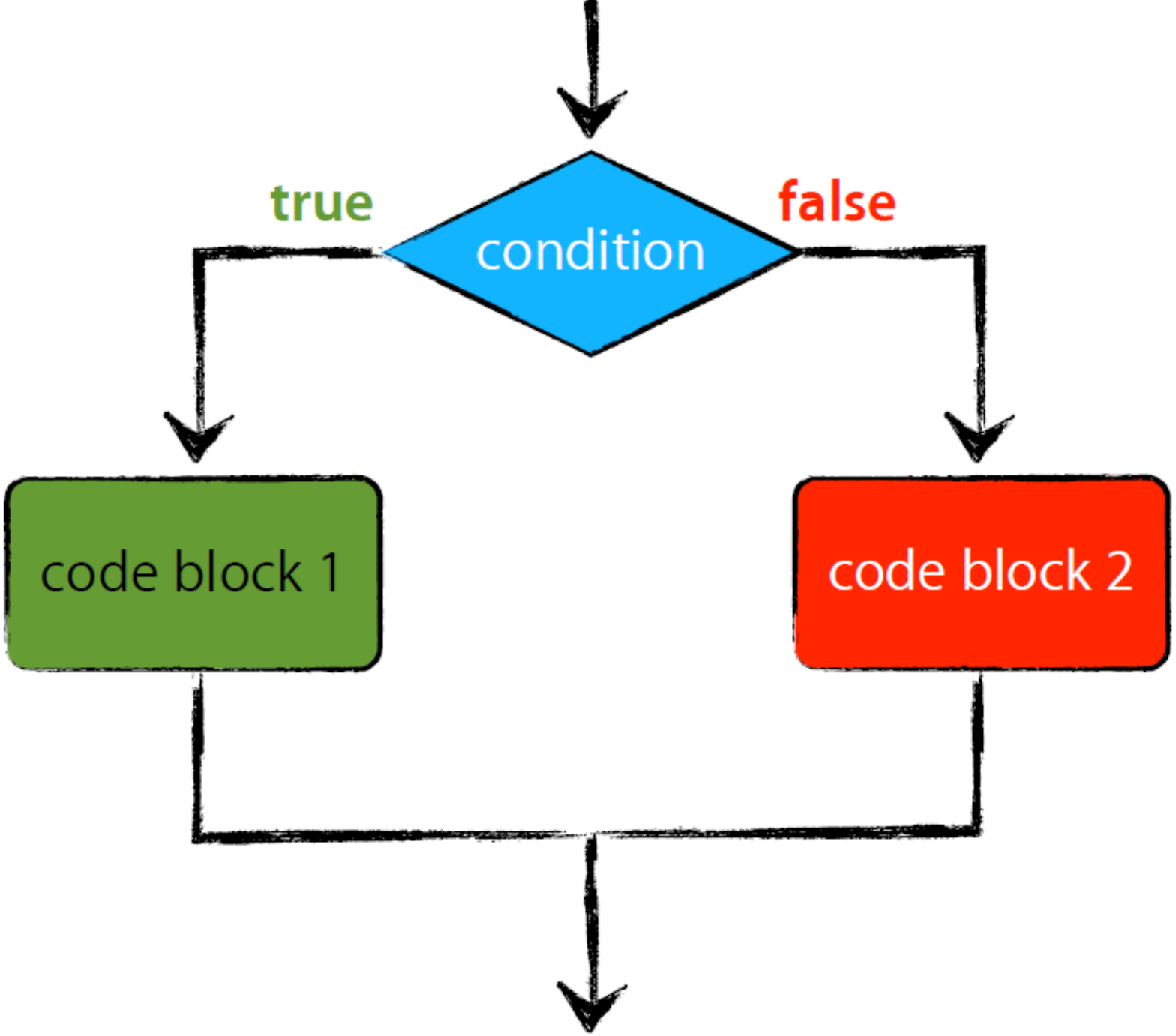
Notice!

Obligatory copyright notice in the age of digital delivery and online classrooms:

Last Class:

Control Structures

Determine the sequence of execution of a set of instructions.



The `if` Statement

```
/* single branch */

if (condition)
    condition is true, execute statement;
```

```
int temp;

printf("What is the temperature? ");
scanf("%d", &temp);

if (temp >= 100)
{
  printf("The water is boiling!\n");
}
```

Input:	Output:
107	The water is boiling!
24	

Two Branches

```
if (condition)
  condition is true, execute statement;
else
  condition is false, execute statement;
```

```
int temp;

printf("What is the temperature? ");
scanf("%d", &temp);

if (temp >= 20) {
   printf("It is warm outside \n"); }
else {
   printf("It is cool outside \n"); }
```

Input:	Output:
17	It is cool outside
24	It is warm outside

Quincy 2005 - [test *]

File Edit View Project Debug Tools Window Help

```c
#include <stdio.h>
int main (void)
{
    int temp;

    printf("What is the temperature? ");
    scanf("%d", &temp);

    if (temp >= 20)
    {
        printf("It is warm outside \n");
    }
    else
    {
        printf("It is cool outside \n");
    }
}
```

Press F1 for help | Ln 6, Col 38 | NUM

Quincy 2005

```
What is the temperature? 17
It is cool outside

Press Enter to return to Quincy...
```

Quincy 2005

```
What is the temperature? 24
It is warm outside

Press Enter to return to Quincy...
```

Moving On…

Multiple Branches

Nested `if` Statements

A nested `if` statement is an `if` statement inside an `if` statement

```
if (condition1)
  condition1 is true, execute statement;
else
  if (condition2)
    condition2 is true, execute statement;
```

```
int temp;

printf("What is the temperature? ");
scanf("%d", &temp);

if (temp >= 100)
    printf("Water is boiling \n");
else
    if (temp <= 0)
        printf("Water is frozen \n");
```

What about the **one-statement** rule?

Compound Statements

```
if (temp >= 100)
    printf("Water is boiling \n");
else
    if (temp <= 0)
        printf("Water is frozen \n");
```

An `if` condition and its accompanying statement(s) are considered a ***compound statement***, and thus obey the one-statement rule.

Apply Pro Tip

```
if (temp >= 100)
{
    printf("Water is boiling \n");
}
else                          /* Indents are used
{                             for readability, but
    if (temp <= 0)            are not required */
    {
        printf("Water is frozen \n");
    }
}
```

```
int noise = 100;

if (noise <= 50)
{
    printf("Quiet");
}
else
{
    if (noise <= 70)
    {
        printf("Intrusive");
    }
    else
    {
        if (noise <= 90)
        {
            printf("Deafening");
        }
        else
        {
            printf("Dangerous");
        }
    }
}
```

- Braces make this code much more clear
- It gets very messy when more `if/else` branches are added.
- We can leverage the one-statement rule to make this look cleaner.

```c
int noise = 100;

if (noise <= 50)
{
    printf("Quiet");
}
else
{
    if (noise <= 70)
    {
        printf("Intrusive");
    }
    else
    {
        if (noise <= 90)
        {
            printf("Deafening");
        }
        else
        {
            printf("Dangerous");
        }
    }
}
```

```c
int noise = 100;

if (noise <= 50) {
    printf("Quiet");
}
else if (noise <= 70) {
    printf("Intrusive");
}
else if (noise <= 90) {
    printf("Deafening");
}
else {
    printf("Dangerous");
}
```

- We are ***only*** rearranging here.
- No code has changed.
- This new form is more readable, intuitive, and expandable.

```
int noise = 40;

if (noise <= 50)
{
    printf("Quiet");
}
else if (noise <= 70)
{
    printf("Intrusive");
}
else if (noise <= 90)
{
    printf("Deafening");
}
else
{
    printf("Dangerous");
}
```

The order of the conditions is **VERY IMPORTANT!**

As soon as a true condition is found, the remaining `if/else` statements are skipped.

Notice:
No condition with the `else`. `else` catches everything that isn't covered by the other conditions.

```
int noise = 40;

if (noise <= 50)
{
    printf("Quiet");
}
else if (noise <= 70)
{
    printf("Intrusive");
}
else if (noise <= 90)
{
    printf("Deafening");
}
else
{
    printf("Dangerous");
}
```

This is **TRUE**

Execute this statement

None of this code executes!

```
int noise = 75;

if (noise <= 50)
{
    printf("Quiet");
}
else if (noise <= 70)
{
    printf("Intrusive");
}
else if (noise <= 90)
{
    printf("Deafening");
}
else
{
    printf("Dangerous");
}
```

Change value of **noise**

This is **FALSE**

Does **not** execute!

Move onto this code

```
int noise = 75;

if (noise <= 50)
{
    printf("Quiet");
}
else if (noise <= 70)
{
    printf("Intrusive");
}
else if (noise <= 90)
{
    printf("Deafening");
}
else
{
    printf("Dangerous");
}
```

This is **FALSE**

Does **not** execute!

Move onto this code

```
int noise = 75;

if (noise <= 50)
{
    printf("Quiet");
}
else if (noise <= 70)
{
    printf("Intrusive");
}
else if (noise <= 90)
{
    printf("Deafening");
}
else
{
    printf("Dangerous");
}
```

This is **TRUE**

Execute this statement

None of this code executes!

```c
int noise;
scanf("%d", &noise);
if (noise <= 50)
{
    printf("Quiet");
}
else if (noise <= 70)
{
    printf("Intrusive");
}
else if (noise <= 90)
{
    printf("Deafening");
}
else
{
    printf("Dangerous");
}
```

Input:	Output:
105	Dangerous
12	Quiet
71	Deafening

```c
int noise;
scanf("%d", &noise);
if (noise <= 50)
{
    printf("Quiet");
}
else if (noise <= 70)
{
    printf("Intrusive");
}
else if (noise <= 90)
{
    printf("Deafening");
}
else
{
    printf("Dangerous");
}
```

Are these the same?

NO!!!

- **Condition order matters!**
- **Quiet and intrusive cannot be reached!**
- **<=50? Also <=90**

```c
int noise;
scanf("%d", &noise);
if (noise <= 90)
{
    printf("Deafening");
}
else if (noise <= 70)
{
    printf("Intrusive");
}
else if (noise <= 50)
{
    printf("Quiet");
}
else
{
    printf("Dangerous");
}
```

`switch` Statements

int or char

```
switch(control_value)
{
        case value1:
                /* statement(s) */
                break;
        case value2:
                /* statement(s) */
                break;
        case value3:
                /* statement(s) */
                break;
        default:
                /* statement(s) if no case value
                   matches the control value */
}
```

Notice:
These are colons!
Not semi-colons.

```
int control = 0;
switch(control)
{
      case 0:
            /* statement(s) */
            break;
      case 1:
            /* statement(s) */
            break;
      case 2:
            /* statement(s) */
            break;
      default:
            /* statement(s) if no case value
               matches the control value */
}
```

If **control** == 0 ,
Execute the following statements:

The **break** keyword exits the **switch** control structure

Does not execute!

`if/else` equivalent

```
switch(control)
{
    case value1:
        /* statement(s) */
        break;
    case value2:
        /* statement(s) */
        break;
    case value3:
        /* statement(s) */
        break;
    default:
        /* statement(s) if no
        value matches control */
}
```

```
if (control == value1)
{
        /* statement(s) */
}
else if (control == value2)
{
        /* statement(s) */
}
else if (control == value3)
{
        /* statement(s) */
}
else
{
        /* statement(s) if no
        value matches control */
}
```

We can have multiple cases per outcome.

```
char colour = 'G';

switch(colour)
{
    case 'R':
    case 'r':
        printf("Stop!");
        break;
    case 'Y':
    case 'y':
        printf("Caution!");
        break;
    case 'G':
    case 'g':
        printf("Go!");
        break;
    default:
        printf("Invalid colour");
}
```

No match!

No match!

Match

Does not execute!

Output?

Go !

```
char value = 'A';

switch(value)
{
   case 'A':
      printf("A ");
   case 'B':
      printf("B ");
   case 'C':
      printf("C ");
   case 'D':
      printf("D ");
   case 'E':
      printf("E ");
   default:
      printf("Default");
}
```

Something's missing ...

Output?

A B C D E Default

We forgot the **break**s! Once we enter a case, <u>everything</u> below gets executed **<u>UNLESS</u>** we use a **break** to exit the switch.

Examples!

Example #1: Floating-Point Equality

Quincy 2005 - [test2.c]

File Edit View Project Debug Tools Window Help

```c
#include <stdio.h>

int main (void)
{
    double x = 0.1;
    double y = 0.3;
    double z = 0.1 + 0.1 + 0.1;

    printf("%.21f\n", x);
    printf("%.21f\n", y);
    printf("%.21f\n", z);

    printf("%d\n", y == z);

    return (0);
}
```

Press F1 for help

Recall:

quincy

```
0.100000000000000010000
0.299999999999999990000
0.300000000000000040000
0

Press Enter to return to Quincy...
```

Example #1: Floating-Point Equality

What to do…?

1) Avoid floating point values
   - Often impossible
2) Avoid testing floating point equality
   - Usually possible, but not always
3) Test for “close enough”
   - What is close enough? Depends on our problem.

Example #1: Floating-Point Equality

```
#include <stdio.h>
#include <math.h>

#define EPS 1e-12

int close_enough(double n1, double n2)
{
      double diff = abs(n1 - n2);
      return diff < EPS;
}
```

close_enough.c - C:\Users\aufke\Google Drive\Teaching\CPS 188\Code Samples - Geany

File Edit Search View Document Project Build Tools Help

Symbols

- Functions
  - close_enough [5]
  - main [11]
- Macros
  - EPS [3]

close_enough.c

```c
#include <stdio.h>
#include <math.h>
#define EPS 1e-12

int close_enough(double n1, double n2)
{
    double diff = abs(n1 - n2);
    return diff < EPS;
}

int main (void)
{
    double a = 0.3;
    double b = 0.1 + 0.1 + 0.1;

    printf("Are a and b equal? %d\n", a == b);
    printf("Are a and b close enough? %d\n", close_enough(a, b));

    return 0;
}
```

C:\WINDOWS\SYSTEM32\cmd.exe

```
Are a and b equal? 0
Are a and b close enough? 1


------------------
(program exited with code: 0)

Press any key to continue . . .
```

Example #2:

- Read in three numbers (integers) using scanf
- Print them in ascending order using as ***few comparisons as possible***.

```
int a, b, c;
scanf("%d", &a);
scanf("%d", &b);
scanf("%d", &c);
```

Six possible orderings:

a, b, c	a, c, b
b, a, c	b, c, a
c, a, b	c, b, a

A *straightforward* solution might check each possibility exhaustively.

```c
#include <stdio.h>

int main (void)
{
   int a, b, c;
   scanf("%d%d%d", &a, &b, &c);

   if (a <= b && b <= c)
      printf("%d, %d, %d\n", a, b, c);
   else if (a <= c && c <= b)
      printf("%d, %d, %d\n", a, c, b);
   else if (b <= a && a <= c)
      printf("%d, %d, %d\n", b, a, c);
   else if (b <= c && c <= a)
      printf("%d, %d, %d\n", b, c, a);
   else if (c <= a && a <= b)
      printf("%d, %d, %d\n", c, a, b);
   else if (c <= b && b <= a)
      printf("%d, %d, %d\n", c, b, a);

   return (0);
}
```

- We want ascending order.
- Why are we doing <=?

```c
#include <stdio.h>

int main (void)
{
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    if (a <= b && b <= c)
        printf("%d, %d, %d\n", a, b, c);
    else if (a <= c && c <= b)
        printf("%d, %d, %d\n", a, c, b);
    else if (b <= a && a <= c)
        printf("%d, %d, %d\n", b, a, c);
    else if (b <= c && c <= a)
        printf("%d, %d, %d\n", b, c, a);
    else if (c <= a && a <= b)
        printf("%d, %d, %d\n", c, a, b);
    else if (c <= b && b <= a)
        printf("%d, %d, %d\n", c, b, a);

    return (0);
}
```

- Notice we are checking the same condition multiple times.
- Is there any way we can organize our logic to avoid this?
- This is what separates programmers from computer scientists.
- Let's say our goal is to ***minimize*** comparisons.
- Here, we have a whopping **12** comparisons.
- Though to be fair, in the *best case*, only two are carried out.

```c
#include <stdio.h>

int main (void)
{
   int a, b, c;
   scanf("%d%d%d", &a, &b, &c);

   if (a <= b && b <= c)
      printf("%d, %d, %d\n", a, b, c);
   else if (a <= c && c <= b)
      printf("%d, %d, %d\n", a, c, b);
   else if (b <= a && a <= c)
      printf("%d, %d, %d\n", b, a, c);
   else if (b <= c && c <= a)
      printf("%d, %d, %d\n", b, c, a);
   else if (c <= a && a <= b)
      printf("%d, %d, %d\n", c, a, b);
   else if (c <= b && b <= a)
      printf("%d, %d, %d\n", c, b, a);

   return (0);
}
```

Redundant!

- If it's none of the previous 5, it must be the 6th.
- No need to test the sixth condition

```c
#include <stdio.h>

int main (void)
{
   int a, b, c;
   scanf("%d%d%d", &a, &b, &c);

   if (a <= b && b <= c)
      printf("%d, %d, %d\n", a, b, c);
   else if (a <= c && c <= b)
      printf("%d, %d, %d\n", a, c, b);
   else if (b <= a && a <= c)
      printf("%d, %d, %d\n", b, a, c);
   else if (b <= c && c <= a)
      printf("%d, %d, %d\n", b, c, a);
   else if (c <= a && a <= b)
      printf("%d, %d, %d\n", c, a, b);
   else
      printf("%d, %d, %d\n", c, b, a);

   return (0);
}
```

- We're down to 10, from 12.
- However, we can do MUCH better.

```
int a, b, c, sm, mid, lg;
scanf("%d%d%d", &a, &b, &c);

sm = a;
lg = b;
if (sm > lg) {
    sm = b;
    lg = a;
}
if (c < sm)
    sm = c;
else if (c > lg)
    lg = c;
mid = (a + b + c) - (sm + lg);
printf("%d, %d, %d\n", sm, mid, lg);
```

Three comparisons!

- Down from 10.
- LOTS of extra assignments and arithmetic, however…

If we ***only*** care about printing…

```
if (a < b) {
   if (a < c) {
      if (b < c)
         printf("%d, %d, %d\n", a, b, c);
      else
         printf("%d, %d, %d\n", a, c, b);
   }
   else
      printf("%d, %d, %d\n", c, a, b);
}
else {
   if (a > c) {
      if (b > c)
         printf("%d, %d, %d\n", c, b, a);
      else
         printf("%d, %d, %d\n", b, c, a);
   }
   else
      printf("%d, %d, %d\n", b, a, c);
}
```

If we ***only*** care about printing?

- No extra variables
- No extra assignments
- No extra arithmetic
- At most three comparisons.


Spot the ERROR!
South Africa
SOUTH AFRICA 2010
Controversy at the World Cup
8:05
48°
PL$ LOAN STORE
Tonight
Partly Cloudy Low: 49°

```
#include <stdio.h>

int main (void)
{
    float a = 10;

    if (10 == 10 & (a == 10 | a == 1) )
        printf("a equals 1 or 10\n");

    return 1;
}
```

Should be && and ||

```
#include <stdio.h>

int main (void)
{
     float number;

     if (-100.1)
        printf("Enter a number: ");

     scanf("%f", &n

     if (number > 100)
        printf("The input is ");
        printf("> 100\n");
     else
        printf("The input is <= 100");

     return(0);
}
```

Two statements, missing curly braces!

```
#include <stdio.h>

int main(void)
{
   int a = 5, b = 7;

   if (a == b)
      printf("a equals b");
   else (b != a)
      printf("b equals a");

      return 0;
}
```

No condition with else!

```
#include <stdio.h>

int main(void)
{
    int a = 5, b = 7;

    if (a = 5)
        printf("a is 5");
    else
        printf("a is not 5");

    return 0;
}
```

- We are *assigning* 5 to **a**
- We want to *compare* 5 to **a**
- Use ==, not =, to compare

```
#include <stdio.h>

int main(void)
{
      char q = ‘A’, b = 7;

      if (q == ‘B’);
            printf(“Hello!”);
      else
            printf(“Goodbye!”);

      return 0;
}
```

Semi-colon kills the if structure

```
#include <stdio.h>

int main(void)
{
        int a = 5, b = 7;

        if (a < 5 && a > 7)
                printf("Hello!");
        else
                printf("Goodbye!");

        return 0;
}
```

This is ***impossible***, no matter what **a** is.

```
#include <stdio.h>

int main(void)
{
        int a = 5, b = 7;

        if (b > 5 || b < 7)
                printf("Hello!");
        else
                printf("Goodbye!");

        return 0;
}
```

This is ***always*** true, no matter what **b** is.

Questions?