DPST1091 / CPTG1391

Introduction to Programming

Week 2 – Lecture 1

Lecturer and Course Convener:

Dr Pantea Aria

Control Flow

unswcollege.edu.au

Agenda

- **Last week**
  - Welcome
  - Vlab and hello world
  - Memory, Variables, Constants
  - Arithmetic Operators

- **Today**
  - Control Flow
  - Logical Operators

Input/Output Recap

printf()

What is printf?

printf is a function from the **<stdio.h>** library used to **print output to the screen.**

```c
#include <stdio.h>

int main(void) {

    int number = 10;
    double price = 4.5;

    // Print an integer
    printf("Number: %d\n", number);

    // Print a double with 2 decimal places
    printf("Price: %.2f\n", price);

    // Print text and values together
    printf("Total items: %d, Total cost: %.2f\n", number, price);

    return 0;
}
```

Input/Output Recap

scanf()

What is scanf?

scanf is a function from **<stdio.h>** used to **read input from the keyboard**.

```
1  #include <stdio.h>
2
3  int main(void) {
4      int age;
5      double height;
6
7      // Ask the user for their age
8      printf("Enter your age: ");
9      scanf("%d", &age);
10
11     // Ask the user for their height
12     printf("Enter your height in meters: ");
13     scanf("%lf", &height);
14
15     // Display the input values
16     printf("Age: %d\n", age);
17     printf("Height: %.2f m\n", height);
18
19     return 0;
20 }
```

Control Flow

many problems require executing statements only in **some circumstances**

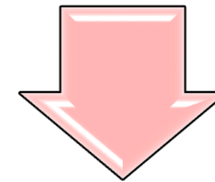
- e.g read two integers and print largest one

sometimes called **control flow**, **branching** or **conditional execution**

In C if Statement can do this.

if

Determines the result of a **Boolean** (**true/false**) question



if **true, do something**

eg: if an int x is even, print it out

Understanding true and false in C

There is **no** “**boolean**” type in **C**.

anything **non-zero** is regarded as “**TRUE**”

0 is regarded as “**FALSE**”

if statement syntax

```
if (expression) {
        statement1;
        statement2;
        ...
}
```

`statement1`, `statement2`, ... are executed if `expression` is **non-zero**.

`statement1`, `statement2`, ... are **NOT executed** if `expression` is **zero**.

Example

```c
1  int main(void) {
2      int number = 10;
3  
4      // Check if the number is positive
5      if (number > 0) {
6          printf("The number is positive.\n");
7      }
8  
9      return 0;
10 }
```

The else keyword

```
if (expression) {
        statement1;
        statement2;
        ...
} else {
        statement3;
        statement4;
        ...
}
```

`statement1`, `statement2`, ... are executed if `expression` is **non-zero**.

`statement3`, `statement4`, ... are **executed** if `expression` is **zero**.

Example

```
 1 int main(void) {
 2 
 3     int number = -3;
 4 
 5     // Check if the number is positive
 6     if (number > 0) {
 7         printf("The number is positive.\n");
 8     }
 9     // If the number is not positive, execute this block
10     else {
11         printf("The number is zero or negative.\n");
12     }
13 
14     return 0;
15 }
```

Multiple if statements can be chained together:

```
int a, b;
printf("Please enter two numbers, a and b: ");
scanf("%d %d", &a, &b);

if (a > b) {
    printf("a is greater than b\n");
} else if (a < b) {
    printf("a is less than b\n");
} else {
    printf("a is equal to b\n");
}
```

Relational Operators

C provides the usual **comparison operators** for numbers:

Operator	Meaning	C Example	Result
>	Greater than	5 > 3	1 (true)
>=	Greater than or equal to	5 >= 5	1 (true)
<	Less than	2 < 4	1 (true)
<=	Less than or equal to	4 <= 3	0 (false)
!=	Not equal to	10 != 8	1 (true)
==	Equal to	6 == 6	1 (true)

Important notes:

⟶ Be cautious when comparing **doubles** using == or !=

⟶ Doubles represent **approximations**, so exact equality may not hold as expected.

⟶ **All evaluate to either true (1) or false (0)**

Logical Operators

Logical operators are used to **combine or invert conditions**.

```
&&   and operator
||   or operator
!    not operator
```

Operator	Meaning	Example	Evaluation	Result
&&	True if **both** operands are true	2 > 0 && 2 < 10	1 && 1	1 (true)
||	True if **either** operand is true	24 > 42 || 2 < 10	0 || 1	1 (true)
!	True if operand is **false**	!(24 > 42)	!0	1 (true)

logical operators return:

⟶ the **int 0** for **false**

⟶ the **int 1** for **true**

Logical Operators - Conditional evaluation

→ **The C operator && || have a useful property.**

They **always** evaluate their **left-hand side first**.

They **only** evaluate their **right-hand** side **if needed**.

&& will **not** evaluate right-hand side if left-hand side is false (zero).

|| will **not** evaluate right-hand side if left-hand side is true (non-zero).

For example, we can write `x != 0 && y/x > 2` **without risking division by zero**.

Example:

```c
#include <stdio.h>

int main(void) {
    int a = 5;
    int b = -3;

    // AND: true only if both conditions are true
    if (a > 0 && b > 0) {
        printf("Both a and b are positive.\n");
    } else {
        printf("At least one of a or b is not positive.\n");
    }

    // OR: true if at least one condition is true
    if (a > 0 || b > 0) {
        printf("At least one of a or b is positive.\n");
    }

    // NOT: invert a condition
    if (!(b > 0)) {
        printf("b is not positive.\n");
    }

    return 0;
}
```

Demo

- →if_int.c
- →if_decimal.c
- →if_char.c
- →if_nested.c

Live lecture code is written for teaching, not perfection.
It may include extra comments and may not always follow ideal coding style

UNSW College

IT'S BREAK TIME!

10 MINUTES BREAK!

```
#include <stdio.h>
#define ON_BREAK 1
int main(){
    // Time for a 10 minutes break! Switch to PARTY_MODE
    #define PARTY_MODE ON_BREAK
    if {PARTY_MODE == ON_BREAK) ;
        print("Program will resume in 10 minutes...");
        sleep(600.);    // Take a break
        exit(0);
}
```

BRB

CHIPS

Relax... We'll be back soon!

loops

Sometimes we need to **execute code multiple times**.

if statements allow us to **execute code only once or not at all** — in other words, **0 or 1** time.

while loops allow us to **execute code repeatedly**, **0 or more** times.

Both ***if*** statements and ***while*** loops use a **condition** to decide whether to run the code.

For ***while*** loops, the **body of the loop keeps executing** as long as the condition remains **true**.

while syntax

```
while (CONDITION) {
      statement1;
      statement2;
      ...
      statementn;
}
```

⟶ C has other looping constructs - but ***while*** is all you need

⟶ ***for*** loops can be a little more concise/convenient. We'll see them later - for now use ***while***

⟶ Often use a **loop counter** variable to count loop repetitions

⟶ Can then have a ***while*** loop execute **n** times.

Example:

```
// read an integer n
// print n asterisks
int loop_counter, n;

printf("How many asterisks? ");
scanf("%d", &n);

loop_counter = 0;
while (loop_counter < n) {
    printf("*");
    loop_counter = loop_counter + 1;
}
printf("\n");
```

while Loop

Loop Counter Pattern

Here is the programming pattern for a while that executes **n** times:

```
int i = 0;
while (i < n) {
    //
    // statements the loop needs to perform
    //

    i = i + 1;
}
```

While Statements

Termination

You can control termination (stopping) of ***while* loops** in many ways.

Easy to write ***while*** loop that **do not terminate**.

Often a **sentinel variable** is used to stop a while loop when a condition occurs in the body of the loop

```
// read numbers printing whether even or odd
// stop if zero read
int stop_loop, numbers;

stop_loop = 0;
while (stop_loop != 1) {
    scanf("%d", &number);
    if (number == 0) {
        stop_loop = 1;
    } else if (number % 2 == 1) {
        printf("%d is odd.\n", number);
    } else {
        printf("%d is even.\n", number);
    }
}
```

while Loop – Sentinel Variable Pattern

Here is the programming pattern for a ***while* loop** that executes until the **sentinel variable** is changed.

```
stop_loop = 0;
while (stop_loop != 1) {
    //
    // statements the loop needs to perform
    //
    if (.......) {
        stop_loop = 1;
    }
    //
    // perhaps more statements
    //
}
```

Demo

- →while_count.c
- →while_count_odd.c
- →while_count_sum.c
- →while_sentinel.c
- →while_sentinel_average.c

Live lecture code is written for teaching, not perfection.
It may include extra comments and may not always follow ideal coding style

UNSW College

Voice of the Student

Anonymous ongoing feedback
Anything you wanted to share with me

26T1 Voice of the Student – Fill out form

See you soon …