COMP1511 17s2

— Lecture 2 —

An Iffy Question

Andrew Bennett

<andrew.bennett@unsw.edu.au>

review: variables
decisions and conditions
constants with `#define`
Boolean logic

While you wait...

Go to the course website, and answer the polls!

`webcms3.cse.unsw.edu.au/COMP1511/17s2`

Admin

Don’t panic!

swapping tute-lab times

lecture recordings are on WebCMS 3

make sure you have **home computing** set up
VLAB

the style guide

Tutorials and Labs

Weekly activities

warmup

lab pair

challenge

Pair programming

two people, one computer

Code reviews

starting week 3

In Review: Variables

declare
the first time a variable is mentioned,
we need to specify its type.

initialise
before using a variable we need to assign it a value.

assign
to give a variable a value.

```
int num; // Declare
num = 5; // Initialise (also Assign)
...
num = 27; // Assign
```

5

making decisions

different behaviour in different situations

Driving, Take 1

Write a program which asks the user to enter their age.

If they are at least 16 years old,
then, display “You can drive.”

Then, whether or not they can drive,
display “Have a nice day.”

Driving, Take 1

Write a program which asks the user to enter their age.

If they are at least 16 years old,
then, display “You can drive.”

Then, whether or not they can drive,
display “Have a nice day.”

Driving, Take 1: Step by Step

… Print “How old are you?”
… Read in their age.
… If their age is ≥ 16: print “You can drive”.
… Print “Have a nice day.”

```c
// Can a user drive?
// Andrew Bennett <andrew.bennett@unsw.edu.au>
// 2017-07-31

#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
    printf ("How old are you? ");
    int age = 0;
    if (age >= 16) {
        printf ("You can drive.\n");
    }

    printf ("Have a nice day.\n");

    return EXIT_SUCCESS;
}
```

Detour: Defining Constant Values

Using the same value numerous times in a program
becomes high maintenance if the value changes…
and needs to be changed in many places.
(You may miss one!)

Other developers may not know (or you may forget!)
what this magical number means.

```
#define MIN_DRIVING_AGE 16
```

note
there is no semicolon at the end of this line

```c
// Can a user drive?
// Andrew Bennett <andrew.bennett@unsw.edu.au>
// 2017-07-31

#include <stdio.h>
#include <stdlib.h>

#define MIN_DRIVING_AGE 16

int main (int argc, char *argv[]) {
    printf ("How old are you? ");
    int age = 0;
    if (age >= MIN_DRIVING_AGE) {
        printf ("You can drive.\n");
    }

    printf ("Have a nice day.\n");

    return EXIT_SUCCESS;
}
```

Driving, Take 2

Write a program which asks the user to enter their age.

If they are at least 16 years old,
then display “You can drive.”
Otherwise, display “You cannot drive.”

Then, whether or not they can drive,
display “Have a nice day.”

Driving, Take 2

Write a program which asks the user to enter their age.

If they are at least 16 years old,
then display “You can drive.”
Otherwise, display “You cannot drive.”

Then, whether or not they can drive,
display “Have a nice day.”

Driving, Take 2: Step by Step

… Print “How old are you?”
… Read in their age.
… If their age is ≥ 16: print “You can drive”.
… Otherwise: print “You cannot drive”.
… Print “Have a nice day.”

```c
// Can a user drive?
// Andrew Bennett <andrew.bennett@unsw.edu.au>
// 2017-07-31

#include <stdio.h>
#include <stdlib.h>

#define MIN_DRIVING_AGE 16

int main (int argc, char *argv[]) {
    printf ("How old are you? ");
    int age = 0;
    if (age >= MIN_DRIVING_AGE) {
        printf ("You can drive.\n");
    } else {
        printf ("You cannot drive.\n");
    }

    printf ("Have a nice day.\n");

    return EXIT_SUCCESS;
}
```

More Conditions!

Sometimes, we want to consider
more than two options for paths.

In the case of the driving scenario,
we want to make sure the age is ≥ 0 and ≤ 120…

Driving, Take 3

```
printf ("How old are you? ");
int age = 0;
if (age < 0) {
    printf ("Invalid input.\n");
} else if (age < MIN_DRIVING_AGE) {
    printf ("You cannot drive.\n");
} else if (age <= MAX_DRIVING_AGE) {
    printf ("You can drive.\n");
} else {
    printf ("Invalid input.\n");
}

printf ("Have a nice day.\n");
```

Conditions in C

less than
in maths, $<$; in C, <

less than or equal to
in maths, $\leq$; in C, <=

greater than
in maths, $>$; in C, >

greater than or equal to
in maths, $\geq$; in C, >=

equal to
in maths, $=$; in C, ==

not equal to
in maths, $\neq$; in C, !=

Nested Conditions

```
if (age >= MIN_DRIVING_AGE) {
    if (age <= MAX_DRIVING_AGE) {
        printf ("You can drive.\n");
    }
}
```

Logical Operators in C

useful when we want to check
multiple conditions in a single if statement.
C uses **Boolean logic**:

AND

in maths, $\land$; in C, &&
both expressions must be true

OR

in maths, $\lor$; in C, ||
either or both expressions must be true

NOT

in maths, $\neg$; in C, !
the expression must be false

Nested Conditions, Redux

```
if (age >= MIN_DRIVING_AGE) {
    if (age <= MAX_DRIVING_AGE) {
        printf ("You can drive.\n");
    }
}
```

is the same as

```
if (age >= MIN_DRIVING_AGE && age <= MAX_DRIVING_AGE) {
    printf ("You can drive.\n");
}
```

An Iffy Answer

```
if (condition 1) {
    // Do stuff
} else if (condition 2) {
    // Do something else
} else if (condition 3) {
    // Do something completely different
} else {
    // In all other cases, do this.
}
```

Indentation

```
if (condition 1) {
// Do stuff
} else if (condition 2) {
// Do something else
} else if (condition 3) {
// Do something completely different
} else {
// In all other cases, do this.
}
```

Indentation

```
if (condition 1) {
    // Do stuff
} else if (condition 2) {
    // Do something else
} else if (condition 3) {
    // Do something completely different
} else {
    // In all other cases, do this.
}
```

Indentation

```
if (condition 1) {
if (condition 2) {
if (condition 3) {
// Do stuff
} else if (condition 4) {
// Do something else
}
} else if (condition 5) {
if (condition 6) {
// Do something completely different
}
} else {
// In all other cases, do this.
}
}
```

Indentation

```
if (condition 1) {
    if (condition 2) {
        if (condition 3) {
            // Do stuff
        } else if (condition 4) {
            // Do something else
        }
    } else if (condition 5) {
        if (condition 6) {
            // Do something completely different
        }
    } else {
        // In all other cases, do this.
    }
}
```

Application: Leap Years

nearly every four years

keeping the calendar in line with the real world

Leap Years

Every year that is exactly divisible by four is a leap year,
except for years that are exactly divisible by 100,
but these centurial years are leap years
if they are exactly divisible by 400.
For example, the years 1700, 1800, and 1900
were not leap years,
but the years 1600 and 2000 were.

Leap Years

Tutorial/lab exercise for this week

Error handling

Driving revisited:
What if somebody types in an invalid age?