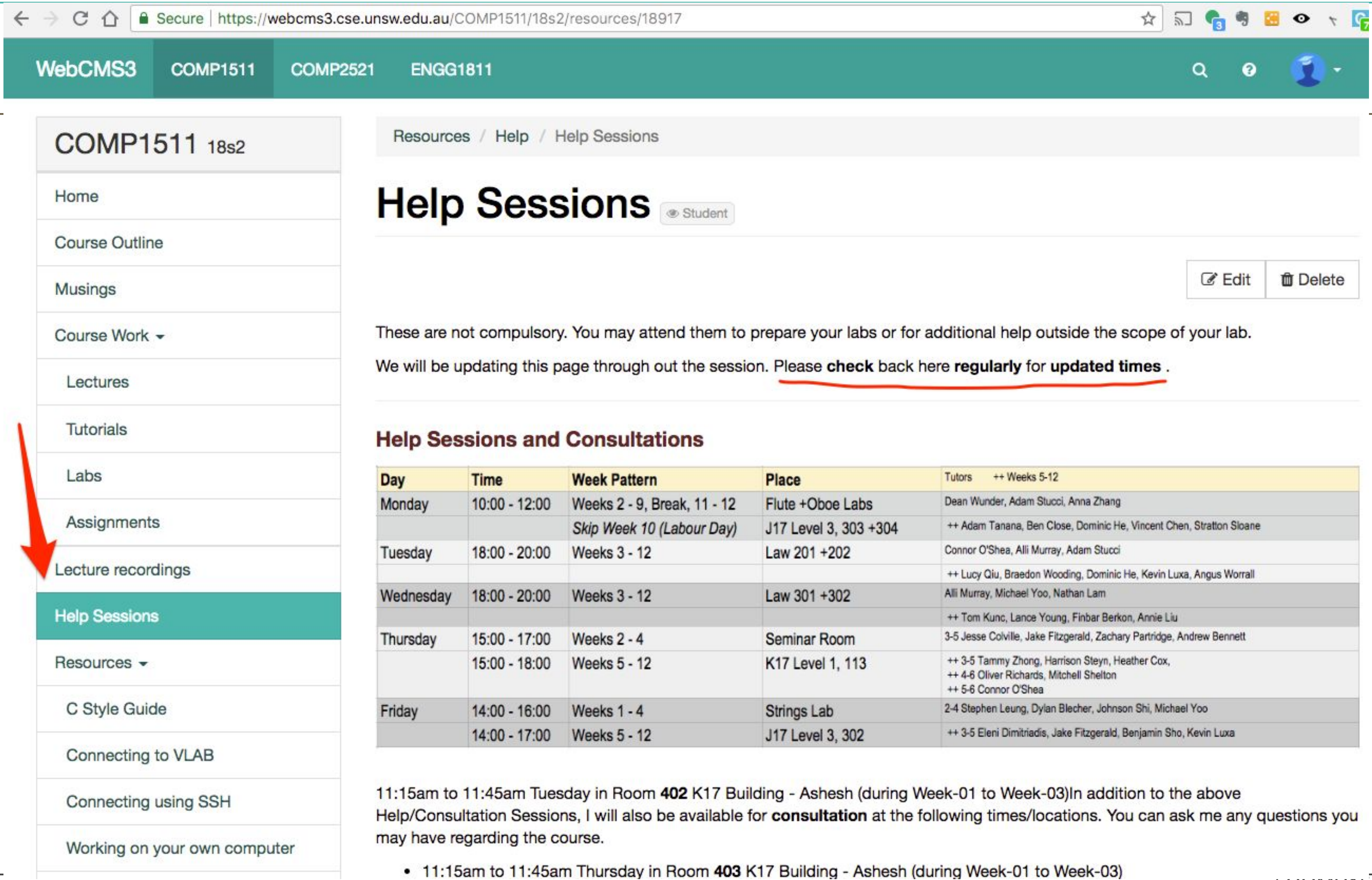
COMP1511: Selection, Function

Session 2, 2018

Help Sessions - available now!

- You can attend **Help Sessions** to get course related help (for tutorials, labs, assignments, etc.)
- Very **useful** and one of the **best ways** to learn and clarify any doubts.
- You can ask questions regarding your tutorials, labs, assignments or seek additional help to understand any course material.

- See "**Help Sessions**" **in the left panel** on the class webpage.



Help Sessions

These are not compulsory. You may attend them to prepare your labs or for additional help outside the scope of your lab.

We will be updating this page through out the session. Please **check** back here **regularly** for **updated times** .

Help Sessions and Consultations

Day	Time	Week Pattern	Place	Tutors ++ Weeks 5-12
Monday	10:00 - 12:00	Weeks 2 - 9, Break, 11 - 12	Flute +Oboe Labs	Dean Wunder, Adam Stucci, Anna Zhang
		Skip Week 10 (Labour Day)	J17 Level 3, 303 +304	++ Adam Tanana, Ben Close, Dominic He, Vincent Chen, Stratton Sloane
Tuesday	18:00 - 20:00	Weeks 3 - 12	Law 201 +202	Connor O'Shea, Alli Murray, Adam Stucci
				++ Lucy Qiu, Braedon Wooding, Dominic He, Kevin Luxa, Angus Worrall
Wednesday	18:00 - 20:00	Weeks 3 - 12	Law 301 +302	Alli Murray, Michael Yoo, Nathan Lam
				++ Tom Kunc, Lance Young, Finbar Berkon, Annie Liu
Thursday	15:00 - 17:00	Weeks 2 - 4	Seminar Room	3-5 Jesse Colville, Jake Fitzgerald, Zachary Partridge, Andrew Bennett
	15:00 - 18:00	Weeks 5 - 12	K17 Level 1, 113	++ 3-5 Tammy Zhong, Harrison Steyn, Heather Cox, ++ 4-6 Oliver Richards, Mitchell Shelton ++ 5-6 Connor O'Shea
Friday	14:00 - 16:00	Weeks 1 - 4	Strings Lab	2-4 Stephen Leung, Dylan Blecher, Johnson Shi, Michael Yoo
	14:00 - 17:00	Weeks 5 - 12	J17 Level 3, 302	++ 3-5 Eleni Dimitriadis, Jake Fitzgerald, Benjamin Sho, Kevin Luxa

11:15am to 11:45am Tuesday in Room **402** K17 Building - Ashesh (during Week-01 to Week-03)In addition to the above Help/Consultation Sessions, I will also be available for **consultation** at the following times/locations. You can ask me any questions you may have regarding the course.

- 11:15am to 11:45am Thursday in Room **403** K17 Building - Ashesh (during Week-01 to Week-03)

Recap: Variables and Constants

- Variable **Types**,
  - `int` (for integer values, e.g.: 42, -10),
  - `float` (for decimal numbers - single *precision*),
  - `double` (for decimal numbers - double *precision*),
  - `char` (for a character like ‘a’, ‘2’, ‘+’, ‘#’, etc.)
- **Variable names** can made up of letters, digits and underscores, they are case sensitive, use a lower case letter to start your variable names

- **Constant** values,
  - **#define** SPEED_OF_LIGHT 299792458.0

```
// Declare and Initialise
int answer = 42;

// Use
printf("Value is %d", num);
```

Recap: `scanf` and `printf`

- **scanf** uses a format string like **printf**. Notice **&** before the variable name.
- scanf **returns number** of items successfully read

Use **%d** to read an **int** (integer) value

```
int answer;
printf("Enter the answer: ");
scanf("%d", &answer);
```

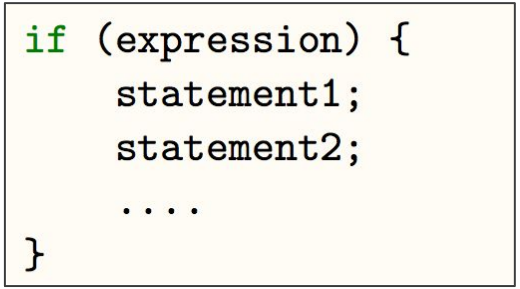
Use **%lf** to read a **double** value

```
double weight;
printf("Enter weight: ");
scanf("%lf", &weight);
```

Reading **multiple** variables (space separated):

```
int num1 = 0;
int num2 = 0;

printf("Enter two numbers, separated by a space: ");
scanf("%d %d", &num1, &num2);
printf("num1 = %d and num2 = %d\n", num1, num2);
```

Recap: Selection - Conditional Execution



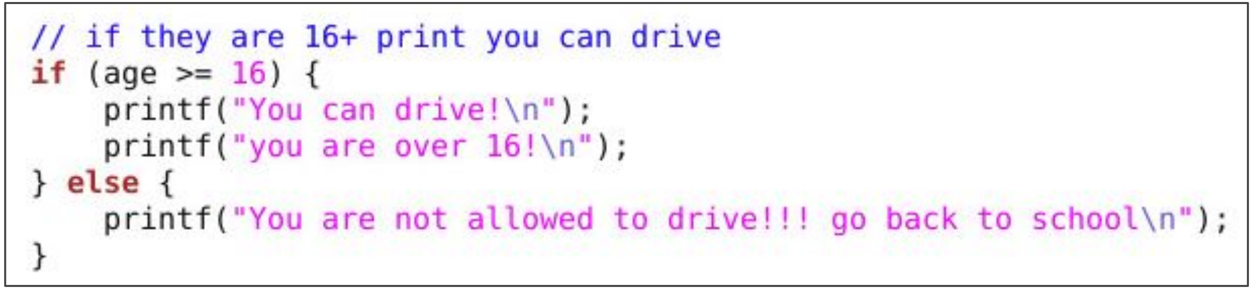
```
if (expression) {
     statement1;
     statement2;
     ....
}
```

Logical operators **&&**, **||** and **!**, e.g.:

```
If (mark > 0 && mark < 100) {

     statement1 ;

     statement2 ;

     …

}
```

By De Morgan’s Law, the following are equivalent in C:

```
! (x > 50 || Y > 50)

!(x > 50) && !(y > 50)

(x <= 50) && (y <= 50)
```



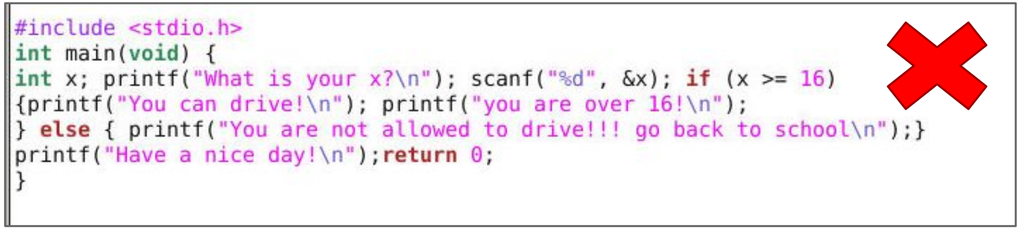
```
// if they are 16+ print you can drive
if (age >= 16) {
    printf("You can drive!\n");
    printf("you are over 16!\n");
} else {
    printf("You are not allowed to drive!!! go back to school\n");
}
```

Recap: Programming Style

- Use the following command to **configure gedit** in your account for COMP1511, the setup **offers features** like line numbers, auto indentation, etc.

```
% 1511 setup-gedit
```

- **Remember** to,
  - write **comments**
  - use **meaningful** variable names
  - use proper **indentations**



```
#include <stdio.h>
int main(void) {
int x; printf("What is your x?\n"); scanf("%d", &x); if (x >= 16)
{printf("You can drive!\n"); printf("you are over 16!\n");
} else { printf("You are not allowed to drive!!! go back to school\n");}
printf("Have a nice day!\n");return 0;
}
```

- **Style Guide** is available on the class webpage,

  see the left panel on the class webpage.

Correction:

- The lecture slide-21 of “Introduction to C” is now corrected!
- The above slide is now replaced by two new slides (slide-21 and 22) in “Introduction to C”, also presented here in the following two slides.

Common mistakes

In English, you say “print Hello if **x is not equal to 6, 7 or 8**”,

which one of the following is correct? A or B ?

A

```
if( (x != 6) || (x != 7) || (x != 8) ){
     printf(“Hello …  \n”);
}
```

B

```
if( (x != 6) && (x != 7) && (x != 8) ){
     printf(“Hello … \n”);
}
```

- Use **different test cases** (here, different values of x) **to check** your logical expression, and make sure that it behaves as per your requirements
- For the answer, see the following slide …

Tautology and Contradiction

- **Avoid *Tautology*:** a ***tautology*** is a logical expression that is ***always true***.

A

```
if( (x != 6) || (x != 7) || (x != 8) ){
     printf(“Hello … \n”);
}
```

- buggy implementation of “**x is not equal to 6, 7 or 8**”.

The above logical expression becomes *false* only if, at the same time,
x == 6 and x == 7 and x == 8, which is not possible! So the condition is always true!

- **Avoid *Contradiction*:** a ***contradiction*** is a logical expression that is ***always false***.

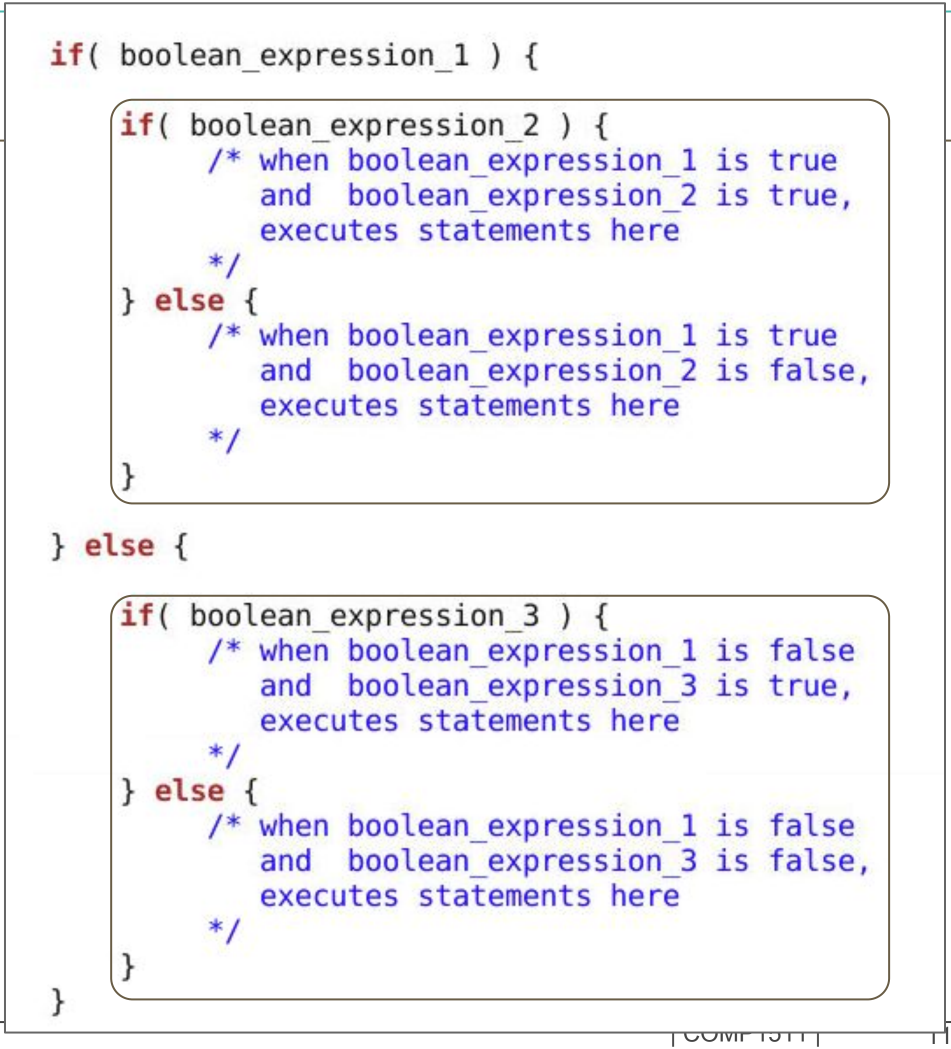
```
if( (x < 0) && (x > 100) ){
     printf(“Error … \n”);
}
```

- buggy implementation of “**print Error if x is outside the range 0 .. 100**”

The condition is always false!

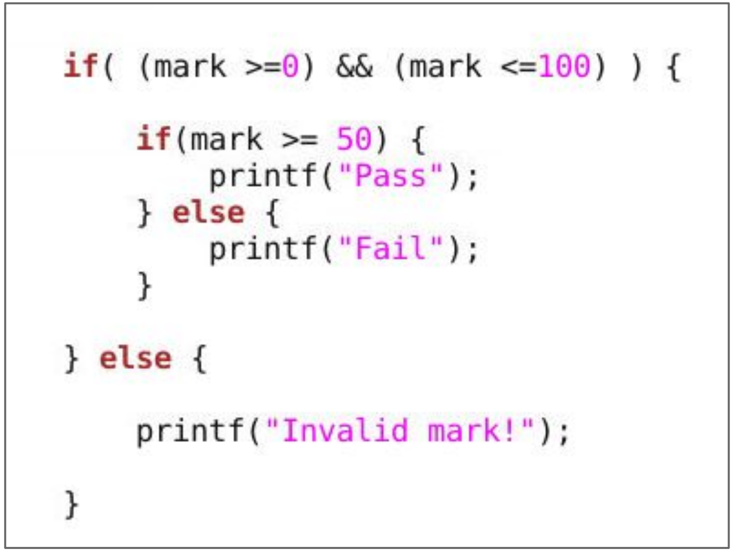
Selection: Nested `If`

You can put an `if/else` inside another `if/else`.

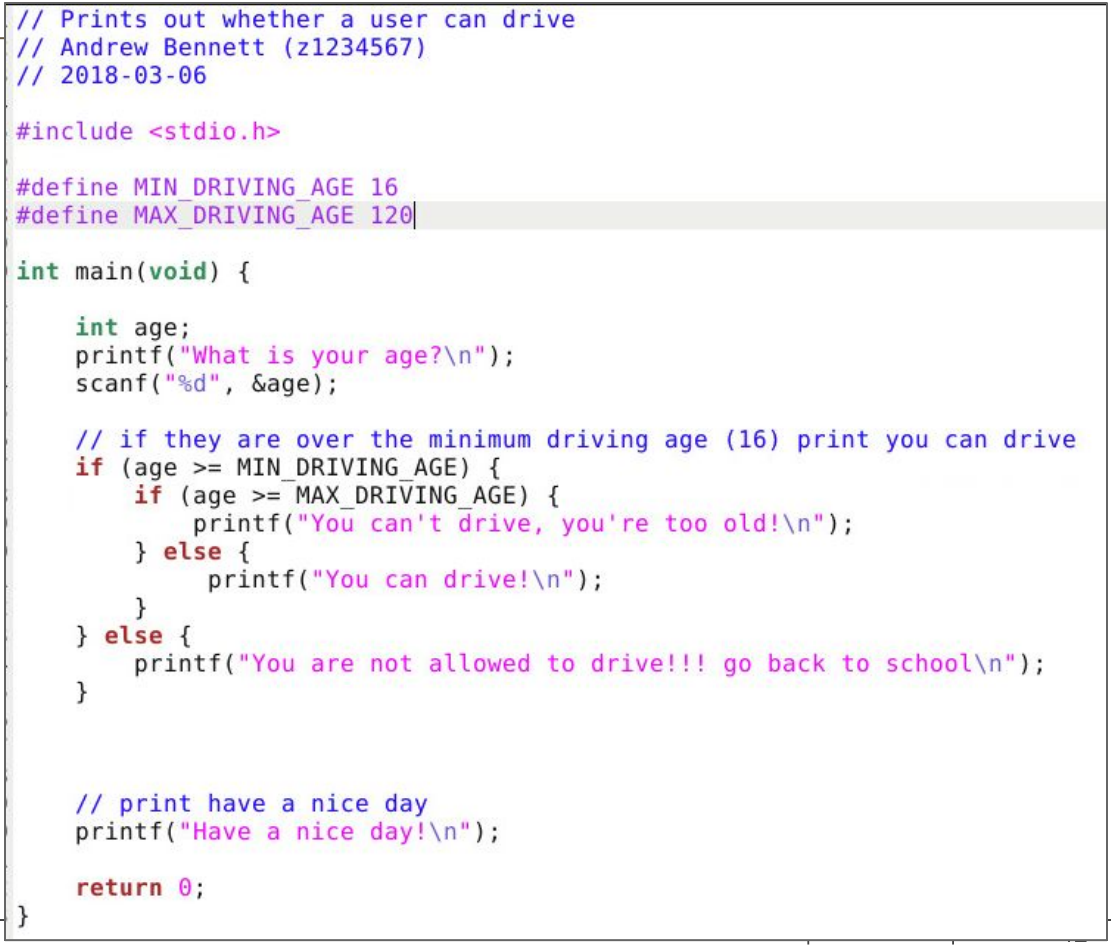


```
if( boolean_expression_1 ) {

    if( boolean_expression_2 ) {
        /* when boolean_expression_1 is true
           and  boolean_expression_2 is true,
           executes statements here
        */
    } else {
        /* when boolean_expression_1 is true
           and  boolean_expression_2 is false,
           executes statements here
        */
    }

} else {

    if( boolean_expression_3 ) {
        /* when boolean_expression_1 is false
           and  boolean_expression_3 is true,
           executes statements here
        */
    } else {
        /* when boolean_expression_1 is false
           and  boolean_expression_3 is false,
           executes statements here
        */
    }
}
```

Selection: Nested `If` – examples

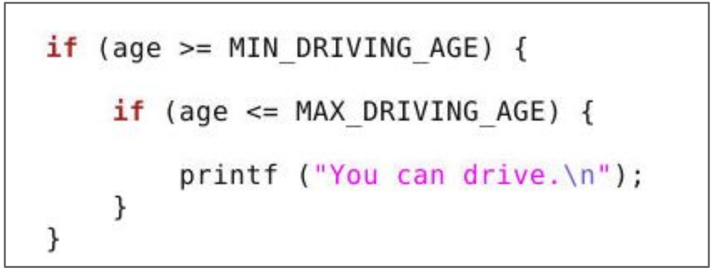


```c
if( (mark >=0) && (mark <=100) ) {

    if(mark >= 50) {
        printf("Pass");
    } else {
        printf("Fail");
    }

} else {

    printf("Invalid mark!");

}
```



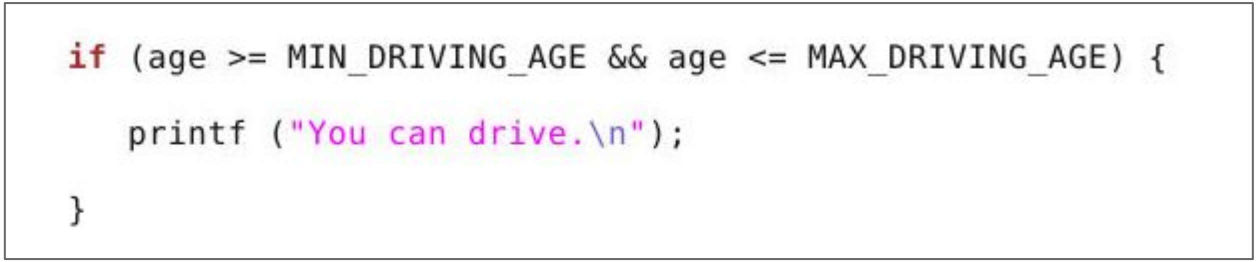
```c
// Prints out whether a user can drive
// Andrew Bennett (z1234567)
// 2018-03-06

#include <stdio.h>

#define MIN_DRIVING_AGE 16
#define MAX_DRIVING_AGE 120

int main(void) {

    int age;
    printf("What is your age?\n");
    scanf("%d", &age);

    // if they are over the minimum driving age (16) print you can drive
    if (age >= MIN_DRIVING_AGE) {
        if (age >= MAX_DRIVING_AGE) {
            printf("You can't drive, you're too old!\n");
        } else {
             printf("You can drive!\n");
        }
    } else {
        printf("You are not allowed to drive!!! go back to school\n");
    }



    // print have a nice day
    printf("Have a nice day!\n");

    return 0;
}
```

Selection: Nested `If`



```
if (age >= MIN_DRIVING_AGE) {

    if (age <= MAX_DRIVING_AGE) {

        printf ("You can drive.\n");
    }
}
```

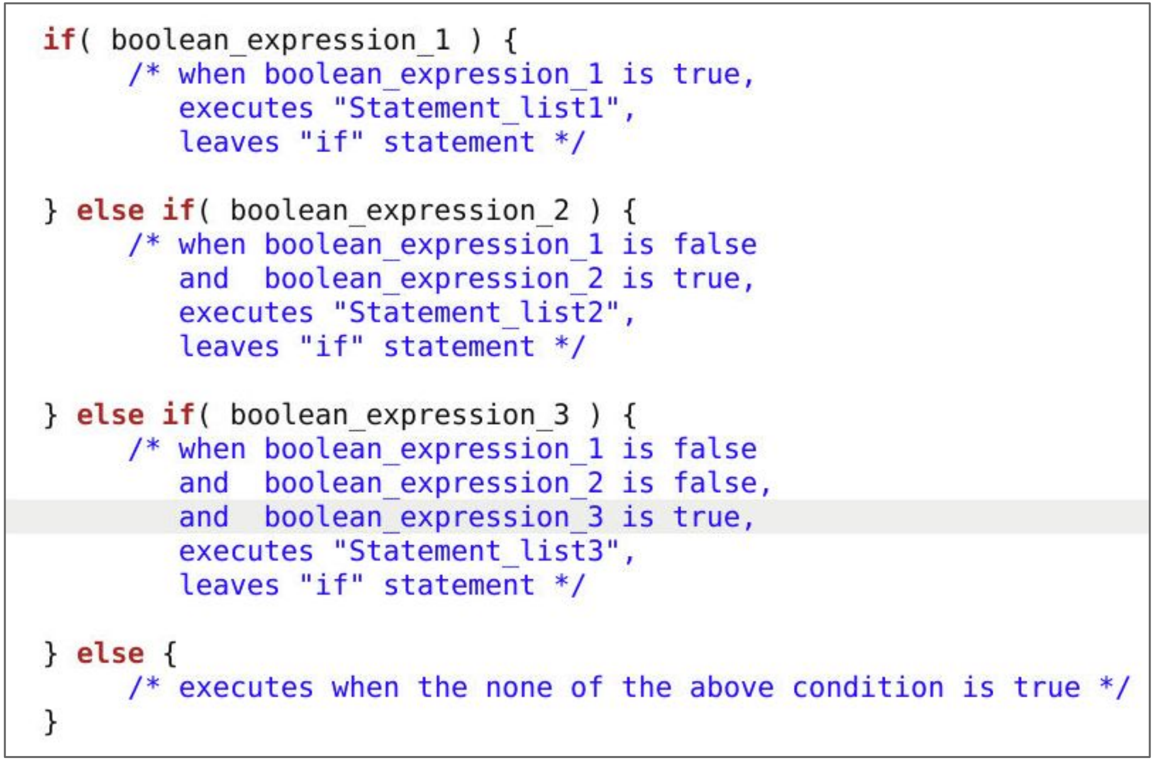
The above is same as the following …



```
if (age >= MIN_DRIVING_AGE && age <= MAX_DRIVING_AGE) {

    printf ("You can drive.\n");

}
```

Selection: **If...else if...else** Statement

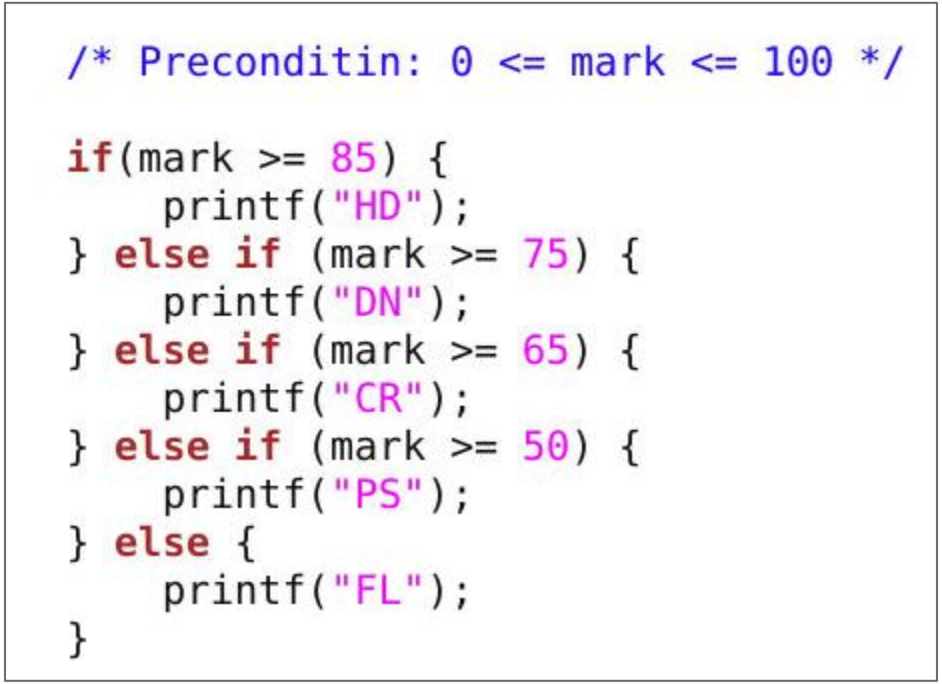
An **if** statement can be followed by an optional **else if...else** statement,



```
if( boolean_expression_1 ) {
     /* when boolean_expression_1 is true,
        executes "Statement_list1",
        leaves "if" statement */

} else if( boolean_expression_2 ) {
     /* when boolean_expression_1 is false
        and  boolean_expression_2 is true,
        executes "Statement_list2",
        leaves "if" statement */

} else if( boolean_expression_3 ) {
     /* when boolean_expression_1 is false
        and  boolean_expression_2 is false,
        and  boolean_expression_3 is true,
        executes "Statement_list3",
        leaves "if" statement */

} else {
     /* executes when the none of the above condition is true */
}
```

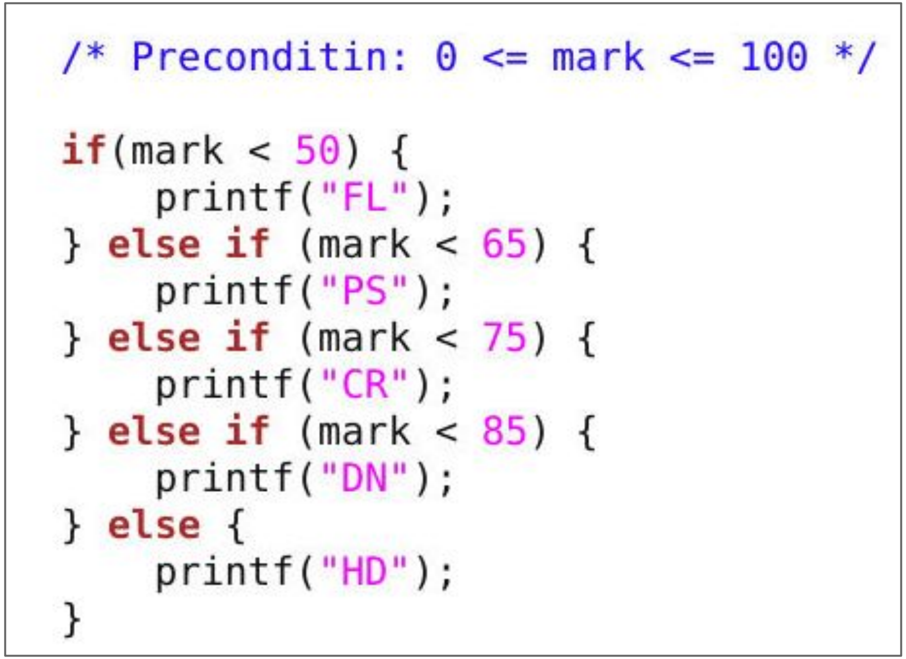
Selection: If...else if...else - examples

Alternative-1



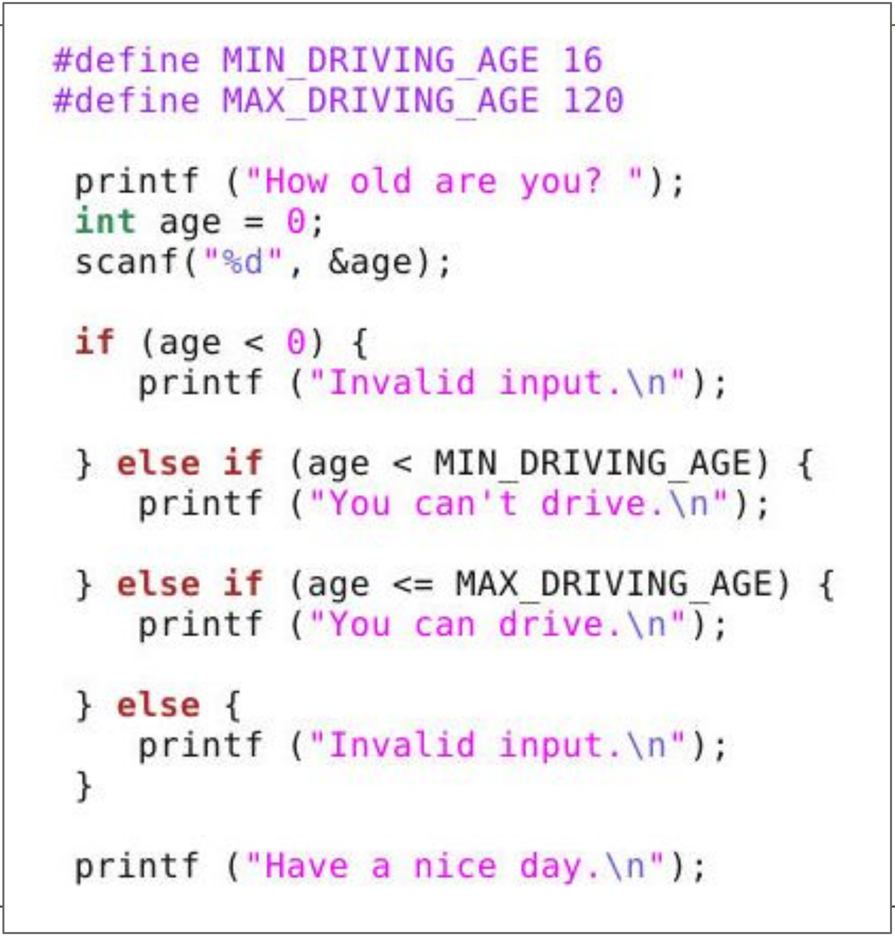
```
/* Preconditin: 0 <= mark <= 100 */

if(mark >= 85) {
    printf("HD");
} else if (mark >= 75) {
    printf("DN");
} else if (mark >= 65) {
    printf("CR");
} else if (mark >= 50) {
    printf("PS");
} else {
    printf("FL");
}
```

Alternative-2



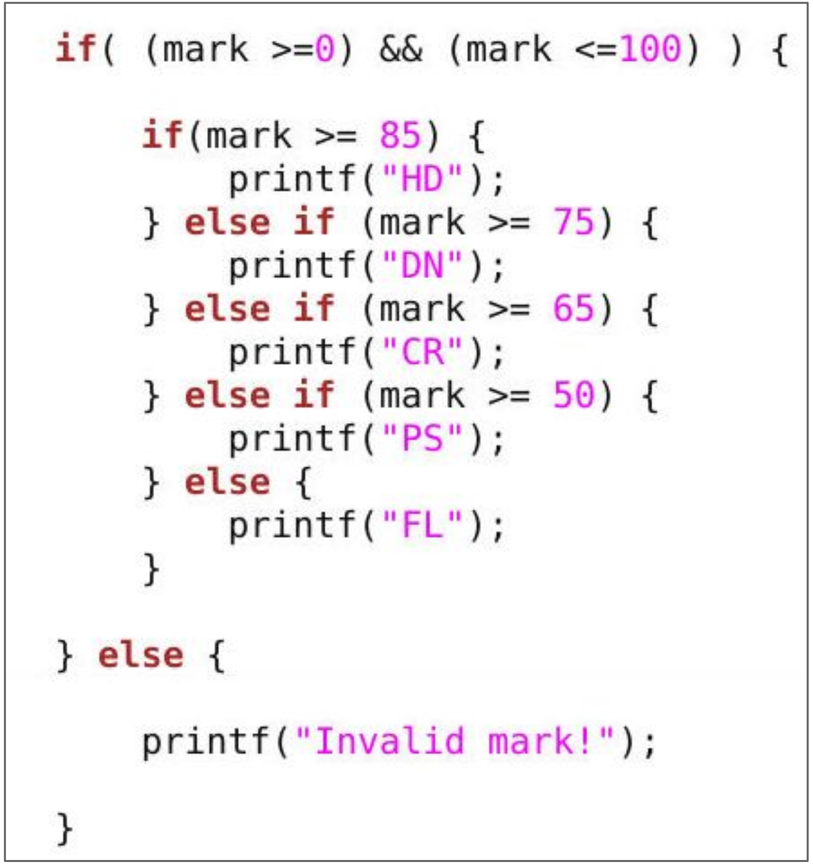
```
/* Preconditin: 0 <= mark <= 100 */

if(mark < 50) {
    printf("FL");
} else if (mark < 65) {
    printf("PS");
} else if (mark < 75) {
    printf("CR");
} else if (mark < 85) {
    printf("DN");
} else {
    printf("HD");
}
```

Selection: `If...else if...else` - example



```
#define MIN_DRIVING_AGE 16
#define MAX_DRIVING_AGE 120

 printf ("How old are you? ");
 int age = 0;
 scanf("%d", &age);

 if (age < 0) {
    printf ("Invalid input.\n");

 } else if (age < MIN_DRIVING_AGE) {
    printf ("You can't drive.\n");

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

 } else {
    printf ("Invalid input.\n");
 }

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

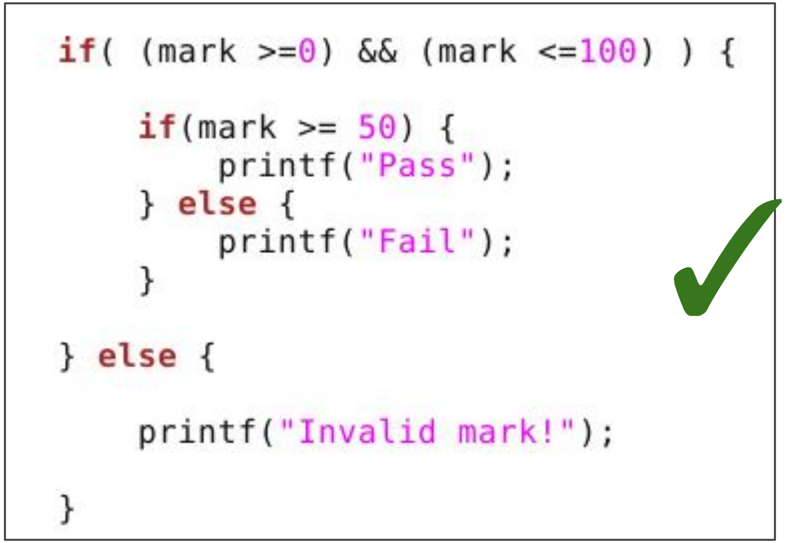
Selection: If...else if...else - example



```c
if( (mark >=0) && (mark <=100) ) {

    if(mark >= 85) {
        printf("HD");
    } else if (mark >= 75) {
        printf("DN");
    } else if (mark >= 65) {
        printf("CR");
    } else if (mark >= 50) {
        printf("PS");
    } else {
        printf("FL");
    }

} else {

    printf("Invalid mark!");

}
```

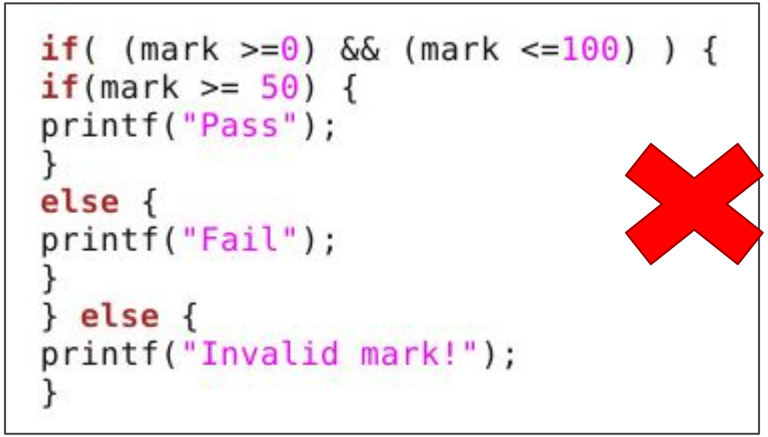
Program Layout: Indentation

- Always use proper indentations and layout



```
if( (mark >=0) && (mark <=100) ) {

    if(mark >= 50) {
        printf("Pass");
    } else {
        printf("Fail");
    }

} else {

    printf("Invalid mark!");

}
```

✓



```
if( (mark >=0) && (mark <=100) ) {
if(mark >= 50) {
printf("Pass");
}
else {
printf("Fail");
}
} else {
printf("Invalid mark!");
}
```

✗

Abstraction via Functions

Functions allow you to:

- separate out “**encapsulate**” a piece of code serving a single purpose
- **hide complexity** of an implementation from a user of the function.

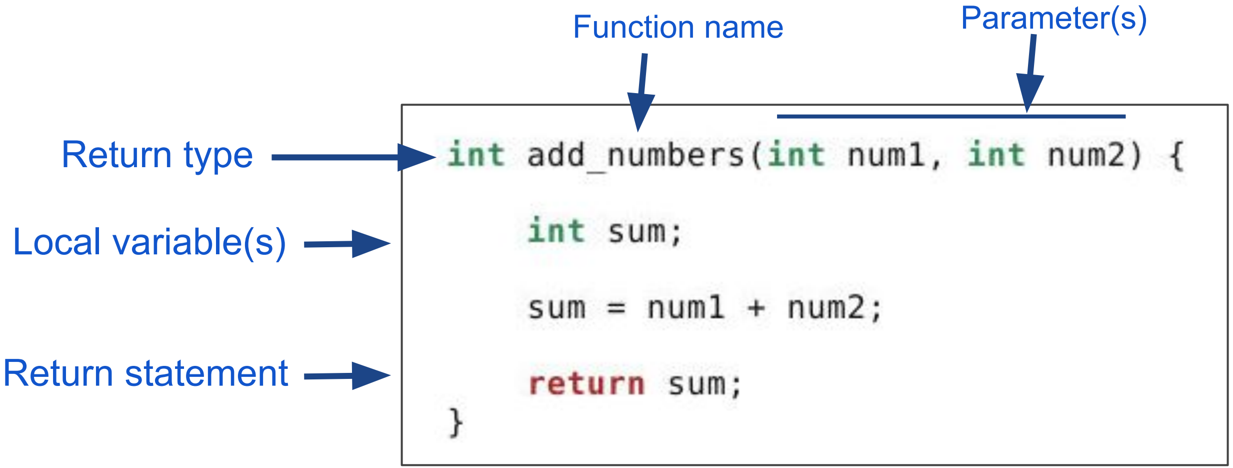
  For example: we use **`printf`** and **`scanf`** functions, but don’t need to know how they are implemented. We only need to know how they behave!
- **easily test** and **verify** a piece of code
- **easily reuse** the code (function) - no copy/paste!
- **shorten code** resulting in easier modification and debugging
- offer **flexibility** - **`printf`** is implemented differently on different operating systems

Functions we already use:

- `printf() , scanf()`

Structure of a Function

1. Return type
2. Function name
3. Parameters (inside brackets, comma separated)
4. Local Variables (only accessible within function)
5. Return statement



Function name

Parameter(s)

Return type

Local variable(s)

Return statement

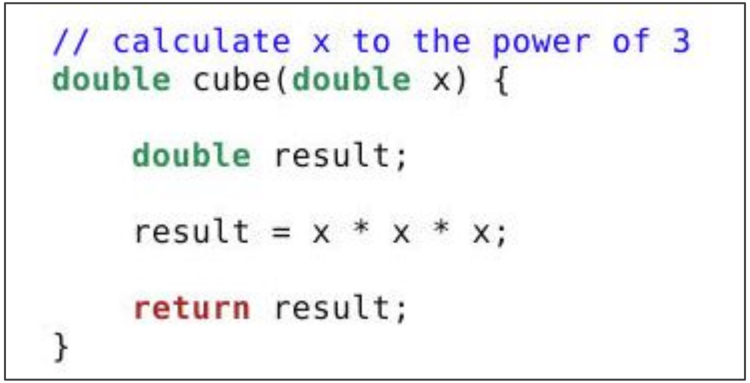
```c
int add_numbers(int num1, int num2) {
    int sum;
    sum = num1 + num2;
    return sum;
}
```

The return Statement

- when a `return` statement is executed, the function terminates.
- the returned expression will be evaluated and, if necessary, converted to the type expected by the calling function.
- *Important:* **all local variables and parameters will be thrown away** when the function terminates.
- the calling function is free to use the returned value, or to ignore it.
- functions can be declared as returning `void`, which means that nothing is returned. A `return` without a value statement can still be used to terminate such a function.

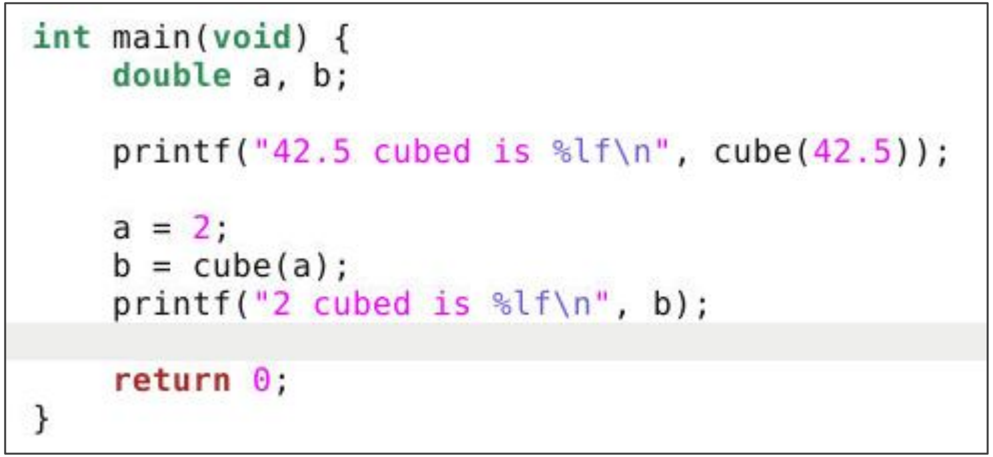
Function - Example

Defining a function



```
// calculate x to the power of 3
double cube(double x) {

    double result;

    result = x * x * x;

    return result;
}
```

Using a function



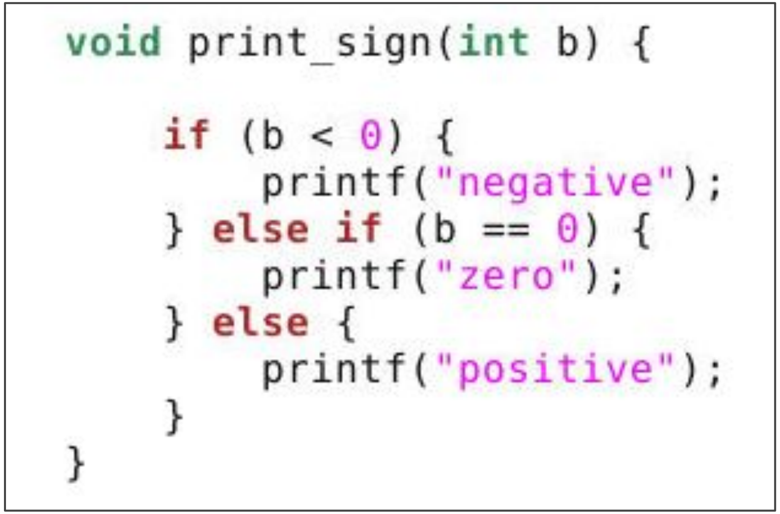
```
int main(void) {
    double a, b;

    printf("42.5 cubed is %lf\n", cube(42.5));

    a = 2;
    b = cube(a);
    printf("2 cubed is %lf\n", b);

    return 0;
}
```

Function Properties

- **function** have a **type** - the type of the value they return
- type **void** for functions that return no value
- **void** also used to indicate function has no parameters
- function can not return arrays (we will discuss this later)
- function have their own **local variables** created when function called and **destroyed** when function returns
- function’s **local** variables are **not accessible outside** the function
- **return** statement stops execution of a function
- **return** statement specifies value to return unless function is of type **void**
- **run-time error** if end of non-void function reached without return

Functions with No Return Value

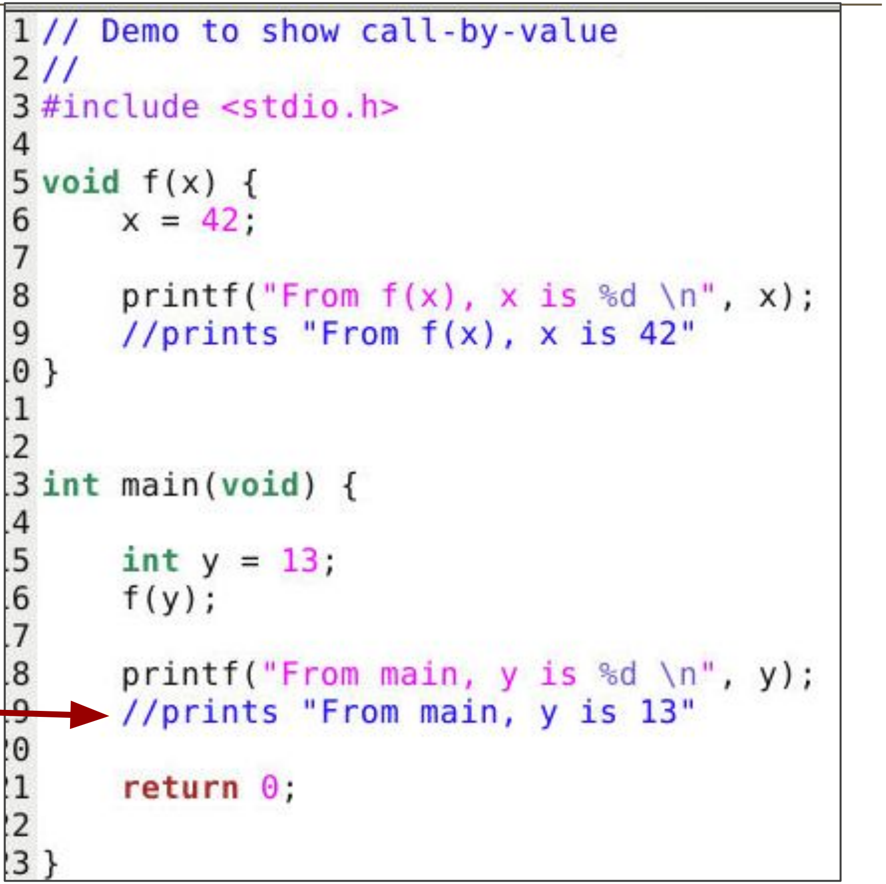
- Some functions do not to compute a value.
- They are useful for "**side-effects**" such as output.



```c
void print_sign(int b) {

    if (b < 0) {
        printf("negative");
    } else if (b == 0) {
        printf("zero");
    } else {
        printf("positive");
    }
}
```

Function Parameters : call-by-value

- functions take 0 or more parameters
- parameters are variables
  - **created** each time a function is called and
  - **destroyed** when a function returns
- C functions are ***call-by-value*** (but beware arrays)
- **parameters initialised** with the value supplied **by the caller**
- Important: if a **parameter** variable is **changed** in the function, it has **no effect outside the function**



```c
// Demo to show call-by-value
//
#include <stdio.h>

void f(x) {
    x = 42;

    printf("From f(x), x is %d \n", x);
    //prints "From f(x), x is 42"
}


int main(void) {

    int y = 13;
    f(y);

    printf("From main, y is %d \n", y);
    //prints "From main, y is 13"

    return 0;

}
```

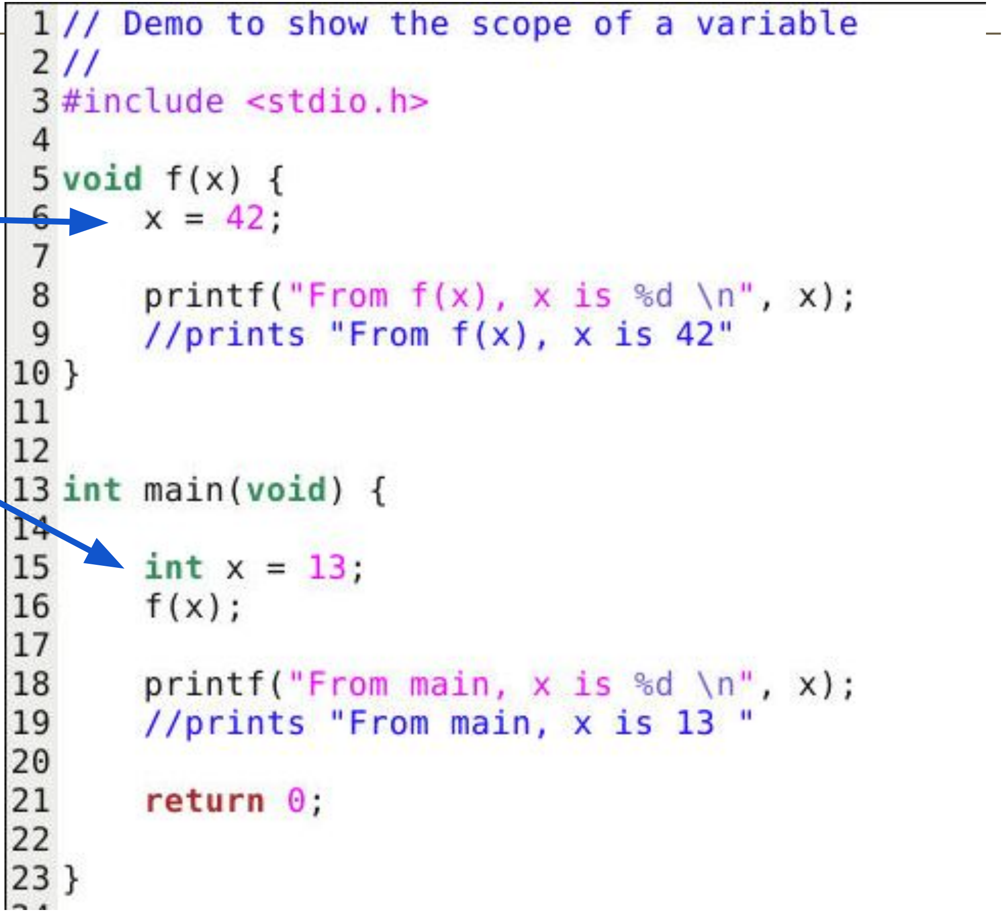
Function - Scope of a variable

Two different variables:

- **x** in function "**f**"
- **x** in function "**main**"

Variable **x** of the function "**f**" is

- **created** every time "**f**" is called and
- **destroyed** when the function "**f**" returns



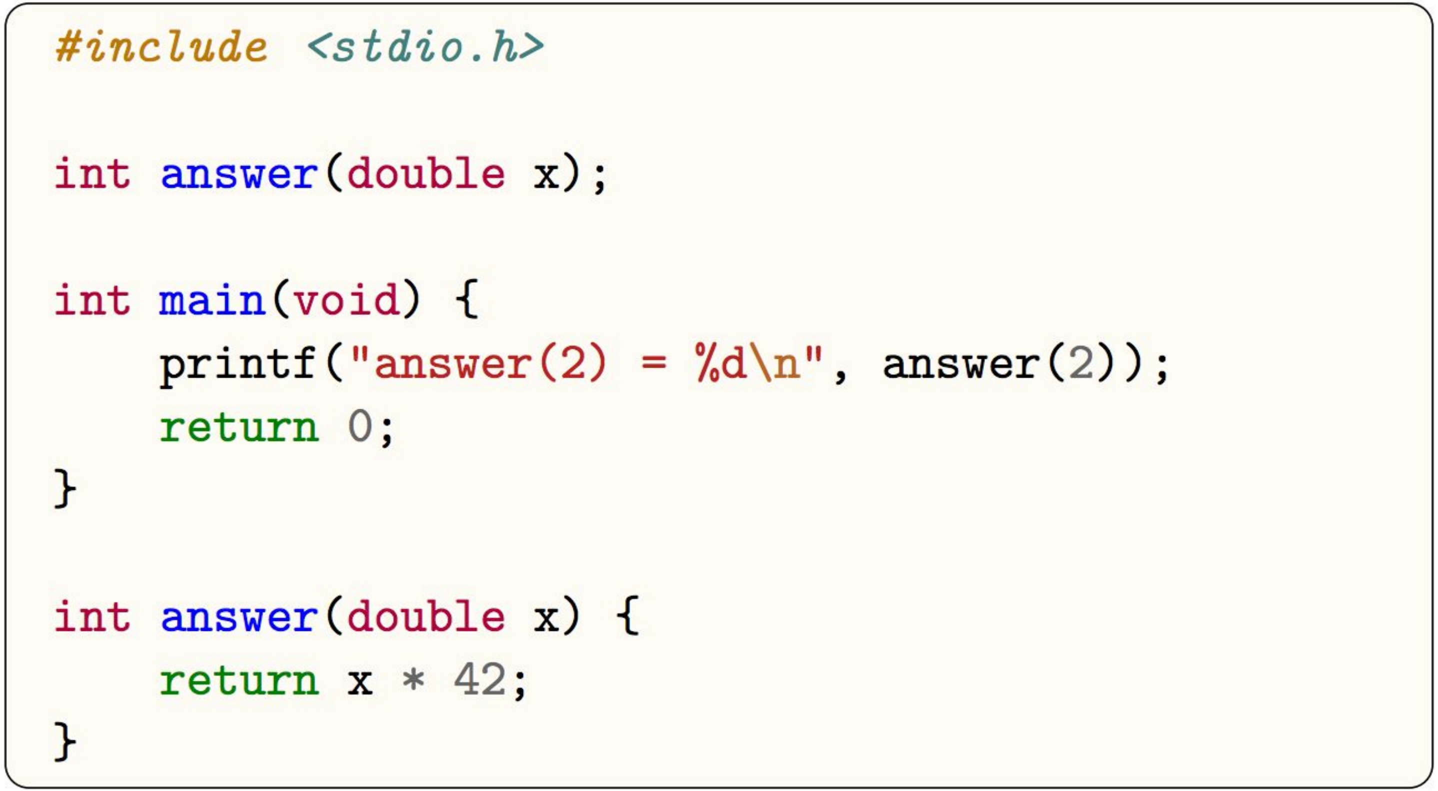
```c
// Demo to show the scope of a variable
//
#include <stdio.h>

void f(x) {
    x = 42;

    printf("From f(x), x is %d \n", x);
    //prints "From f(x), x is 42"
}


int main(void) {

    int x = 13;
    f(x);

    printf("From main, x is %d \n", x);
    //prints "From main, x is 13 "

    return 0;

}
```

Function Prototypes

- Function prototypes allow function to be called before it is defined.
- Specifies key information about the function:
  - function return type
  - function name
  - number and type of function parameters
- Allows top-down order of functions in file
  More readable!
- Allows us to have function definition in separate file.
  Crucial to share code and for larger programs
- Example prototypes:

```
double power(double x, int n);
void print_sign(int b);
```

Example: Prototype allowing Function use before Definition



```
#include <stdio.h>

int answer(double x);

int main(void) {
    printf("answer(2) = %d\n", answer(2));
    return 0;
}

int answer(double x) {
    return x * 42;
}
```

Library functions

- Over 700 function are defined in the C standard library.
- You'll need to use less than 20 of these in COMP1511
- The C compiler needs to see a prototype for these fucntion before you use them.
- You do this indirectly with **#include** line
- For example **stdio.h** contains prototypes for **printf** and **scanf** so:

```
#include <stdio.h>

int main(void) {
    printf("Andrew Rocks!\n");
}
```