

Notes #8

Functions

Revisited!

Chapter 9

CMPT 130

© Dr. B. Fraser

Topics

- 1) Does **modifying a parameter** inside a function affect the calling code?
- 2) **Where** can we put functions in our code?

Pass-by-value

Pass by value

- Only the **value** of an **argument** is passed into the function's parameter.

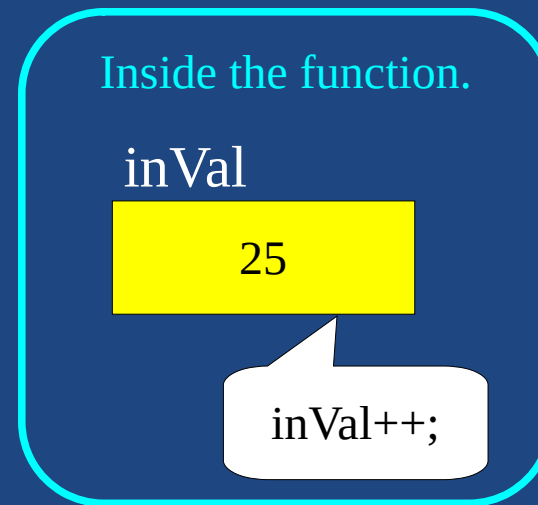
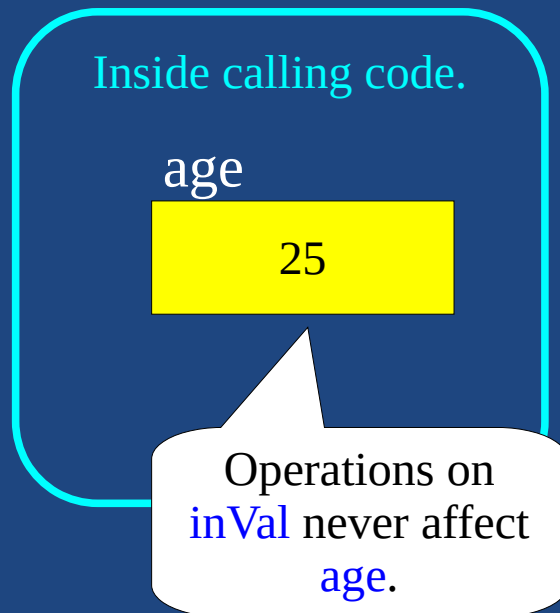
```
void growOlder(int inVal) {  
    inVal++;  
}
```

```
int main () {  
    int age = 25;  
    growOlder(age);  
    cout << "Age is: " << age << endl;  
    return 0;  
}
```

- Changing the parameter's value in a function...

Explaining pass by value

- **Pass by value:**
function's parameter is set to a **copy** of argument.
 - Changing the copy does not affect the original.



Prototypes

Prototypes

- Must know some things about a function to call it.
 - Function prototypes eliminates the need to put..

myProgram.cpp

```
void doStuff() {  
    ...  
}
```

```
int main () {  
    doStuff();  
    return 0;  
}
```

myProgram.cpp

```
int main () {  
    doStuff();  
    return 0;  
}
```

```
void doStuff() {  
    ...  
}
```

This is the
prototype.

We can now call
`doStuff()` above
where the function
is defined.

Needed information to call

- To call a function we need to know:
 -
 - number, type, and order of parameters,
 - return type of function.
- Function prototype idea:
 - Rather than defining the whole function at the top, tell the compile at the top of the .cpp file..

Using prototypes

- Function prototype is similar to a function definition except:
 - (place a ';' instead of {...})

// Prototype

```
void printSum(int x, int y);
```

```
int main ()
```

```
{
```

```
    printSum(1,2);
```

```
    return 0;
```

```
}
```

// Display the sum of the two values.

```
void printSum(int x, int y)
```

```
{
```

```
    cout << x << " + " << y << " = " << (x + y) << endl;
```

```
}
```

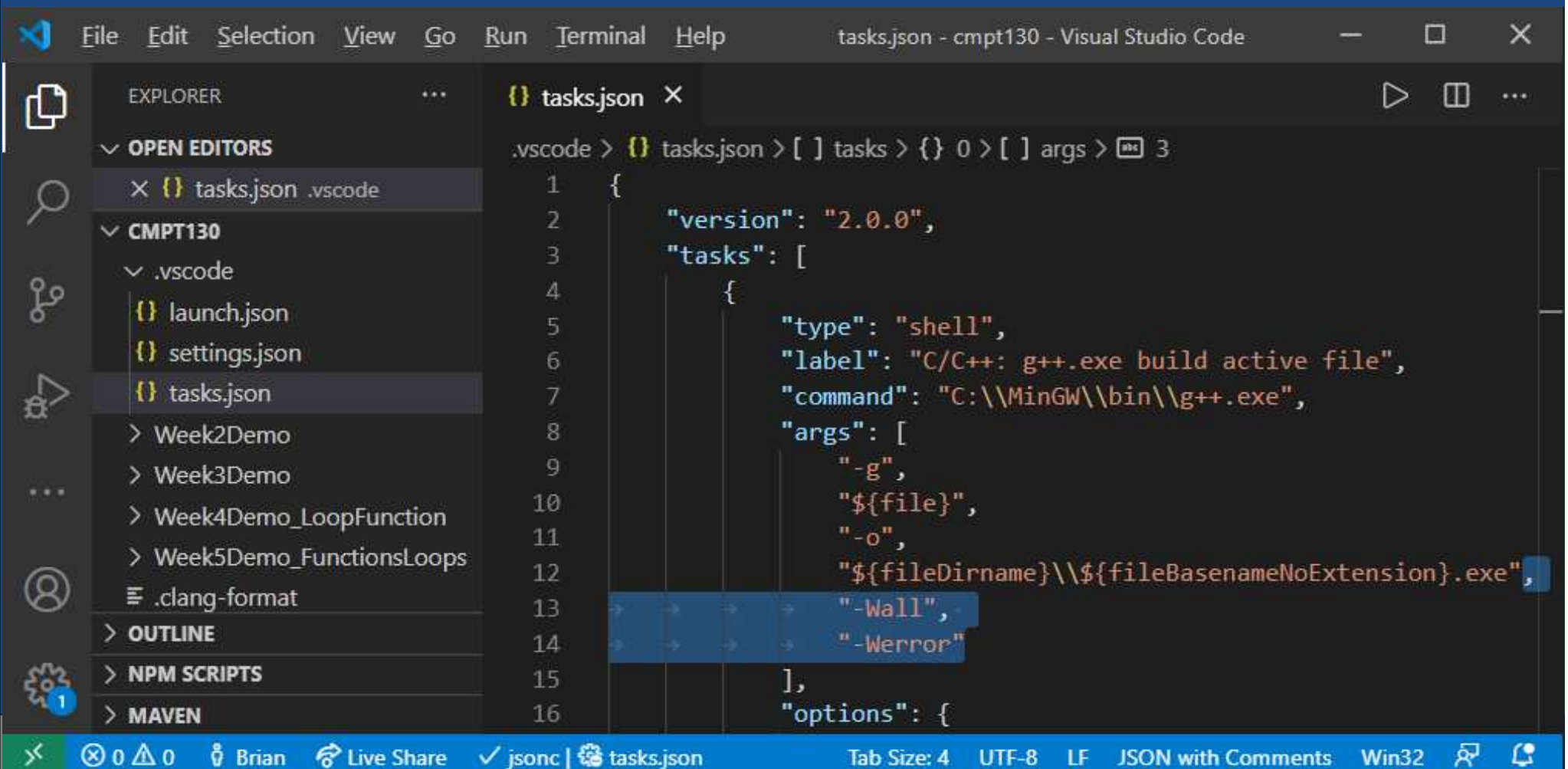
Compiler Warnings

- The compiler generates:
 - Errors if it cannot create a program.
 - Warnings if it finds a likely mistake.
Example: forgetting () on a function call.
- Warnings should not be ignored.
 - They can help you find logic errors!
 - Turn on.. : -Wall
 - Make warnings.. : -Werror

Good so that you can't just run the program and miss critical warnings to help fix your code.

Warnings in VS Code

- In task.json add to compiler's "args":
 , "-Wall", "-Werror"



The screenshot shows the Visual Studio Code interface with the `tasks.json` file open in the editor. The Explorer sidebar on the left shows the project structure for `CMPT130`, including `.vscode` and `tasks.json`. The editor displays the following JSON content:

```
.vscode > {} tasks.json > [ ] tasks > {} 0 > [ ] args > 3
1  {
2      "version": "2.0.0",
3      "tasks": [
4          {
5              "type": "shell",
6              "label": "C/C++: g++.exe build active file",
7              "command": "C:\\MinGW\\bin\\g++.exe",
8              "args": [
9                  "-g",
10                 "${file}",
11                 "-o",
12                 "${fileDirname}\\${fileBasenameNoExtension}.exe",
13                 "-Wall",
14                 "-Werror"
15             ],
16             "options": {
```

The status bar at the bottom indicates the file is `tasks.json` with a tab size of 4, UTF-8 encoding, LF line endings, and JSON with Comments format.

Summary

- **Pass-by-value:** pass in just a copy.
- Use **prototypes** to define function below a call to it.
- Heed the compiler's warnings!