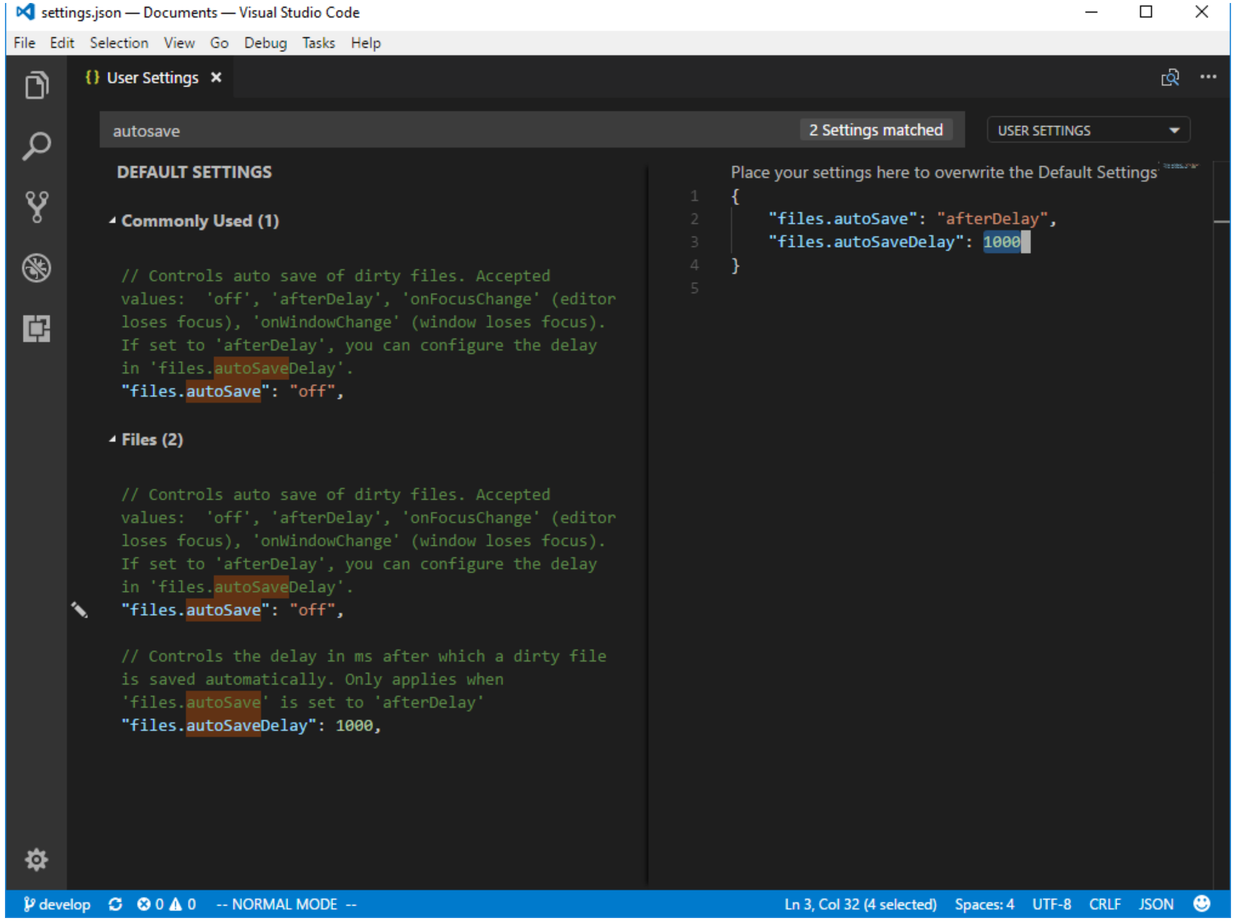
TDDE18 & 726G77

Functions

Labs update

- No more one time password. We will note who have demonstrated during the lab and register this in webreg.
- Use the terminal to send in your lab! Dont use Visual studio code!

Tooltip of the week – Preferences & Auto save



DEFAULT SETTINGS

Commonly Used (1)

```
// Controls auto save of dirty files. Accepted
values:  'off', 'afterDelay', 'onFocusChange' (editor
loses focus), 'onWindowChange' (window loses focus).
If set to 'afterDelay', you can configure the delay
in 'files.autoSaveDelay'.
"files.autoSave": "off",
```

Files (2)

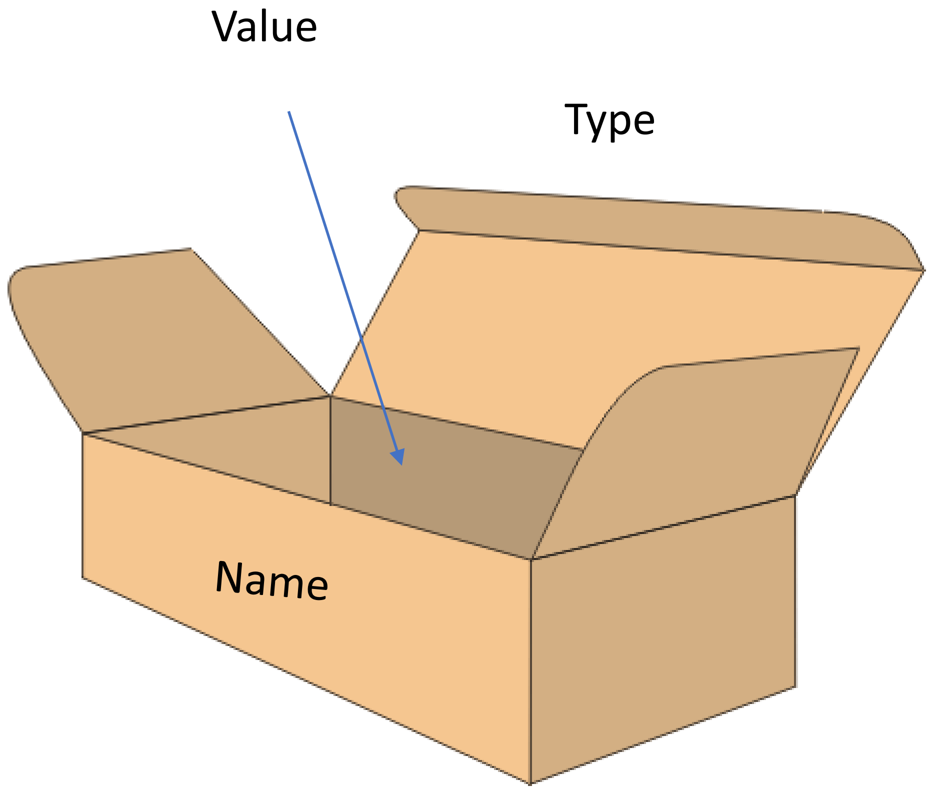
```
// Controls auto save of dirty files. Accepted
values:  'off', 'afterDelay', 'onFocusChange' (editor
loses focus), 'onWindowChange' (window loses focus).
If set to 'afterDelay', you can configure the delay
in 'files.autoSaveDelay'.
"files.autoSave": "off",

// Controls the delay in ms after which a dirty file
is saved automatically. Only applies when
'files.autoSave' is set to 'afterDelay'
"files.autoSaveDelay": 1000,
```

Place your settings here to overwrite the Default Settings

```
{
    "files.autoSave": "afterDelay",
    "files.autoSaveDelay": 1000
}
```

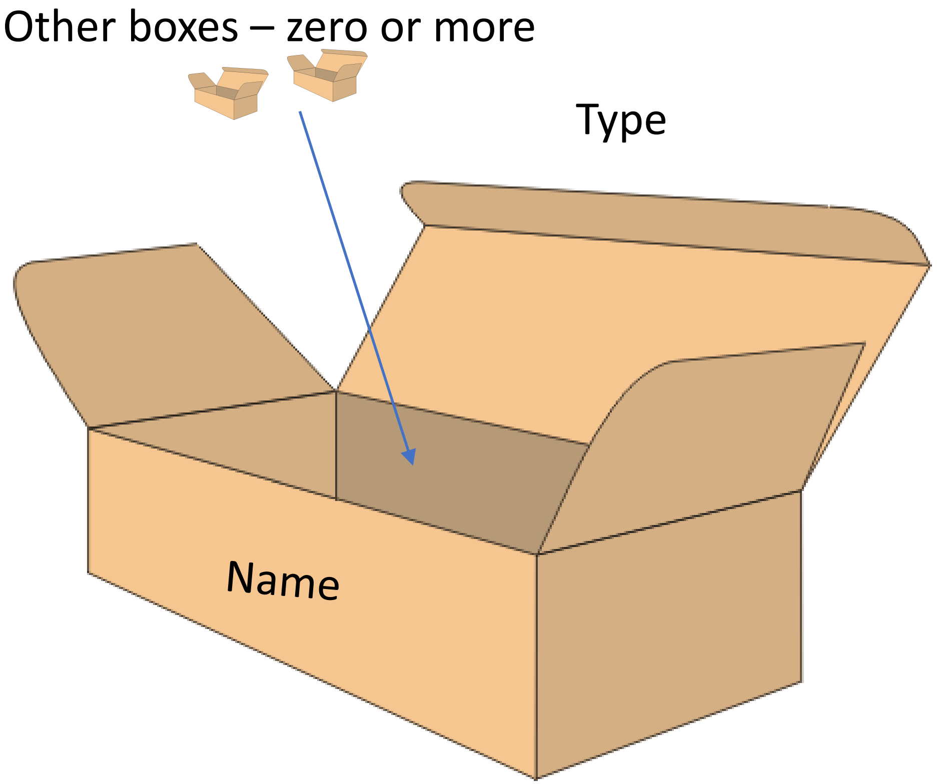
Variable



Variable

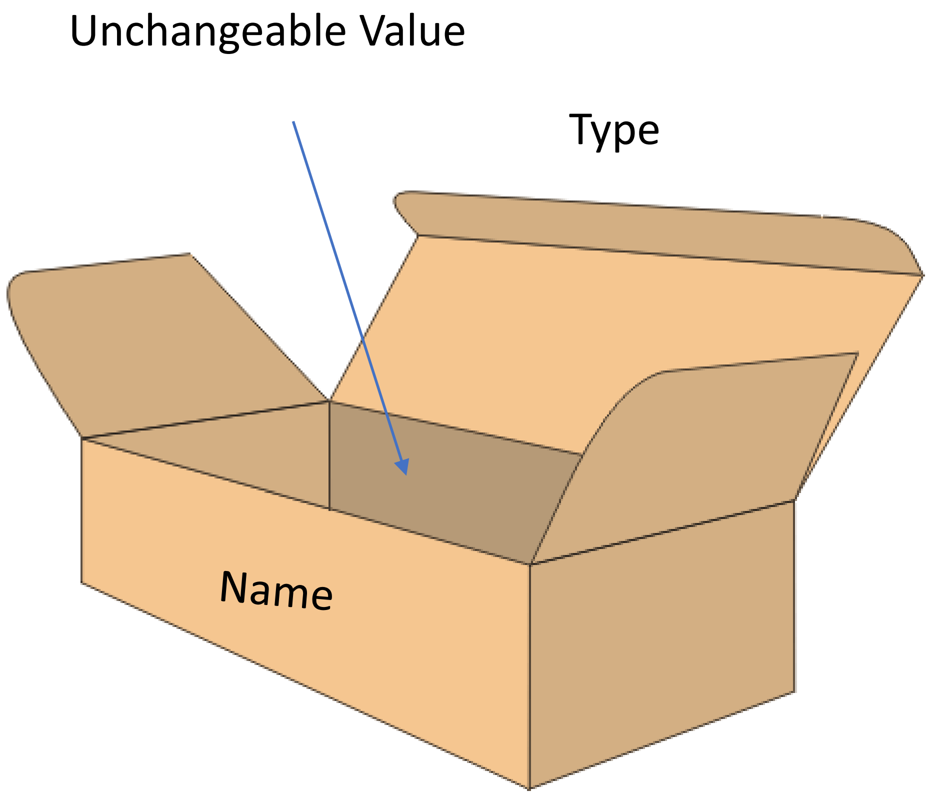
- Fundamental (also called built-in types)
  - Stores a value of a fundamental type, nothing more
- Object
  - Stores values tied to an derived type (struct, class)
  - Operations associated to the type are provided
  - More about classes later in the course
- Pointer
  - Stores the address of some other variable
  - More about pointers in the course

Struct – Compound data type



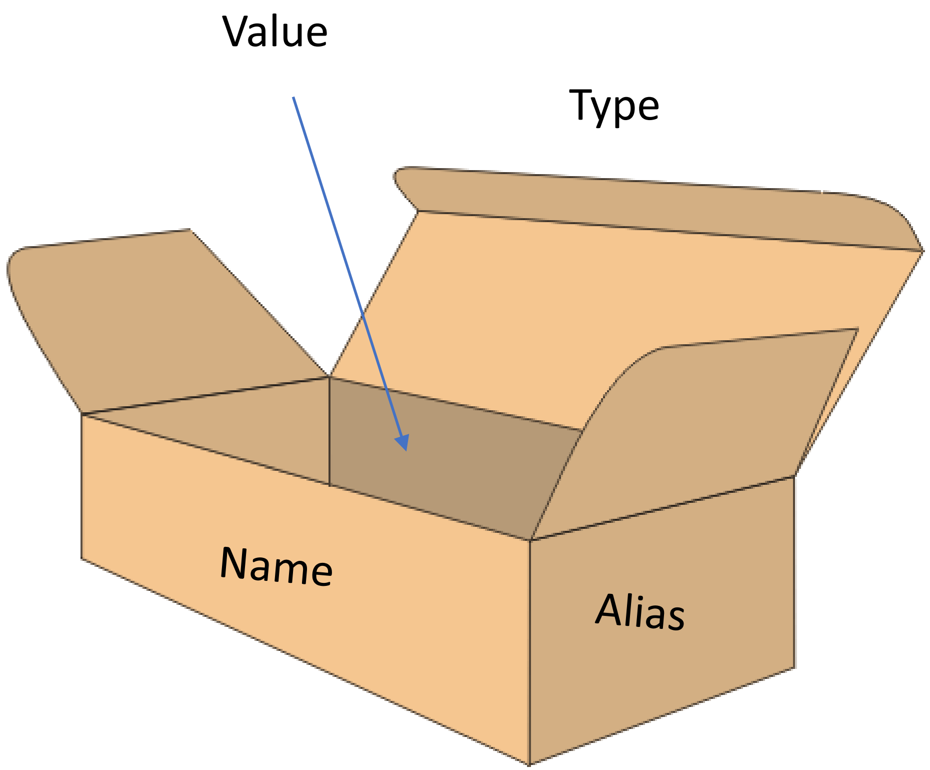
- With struct it is possible to combine variables into one derived type

Constants



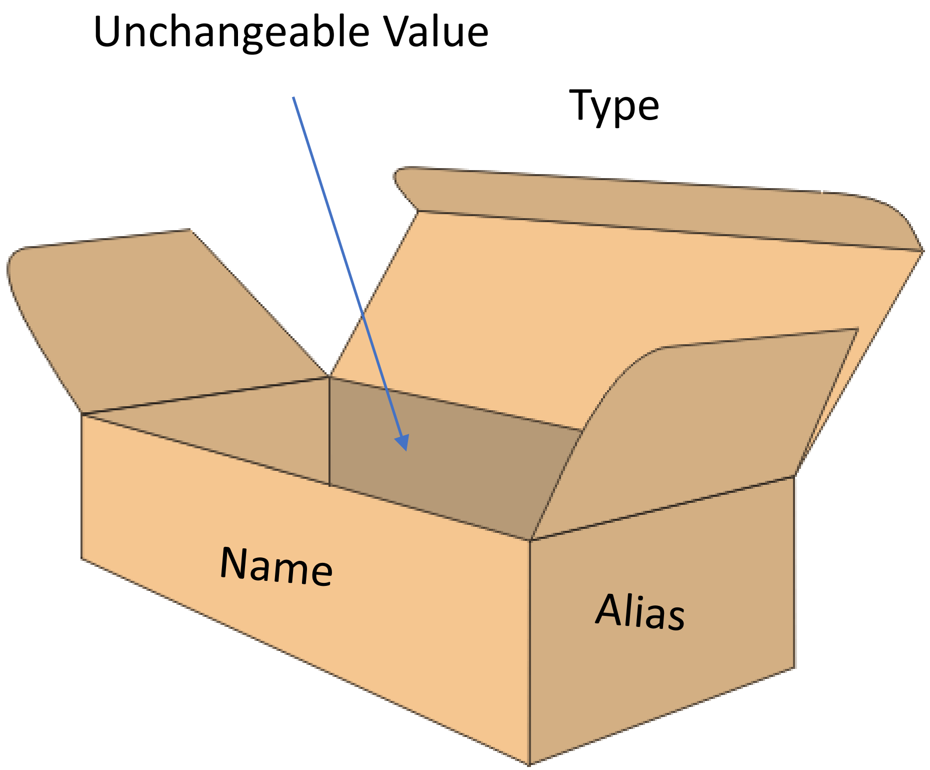
- A variable can be declared *const*
- Modification of a const variable will give compilation error.

Reference



- Alias to another already existing variable
- A reference cannot refer to another variable after definition

Const&



- The value could be change using the original variable and not the reference

Sequence and block

```
{       // Beginning of the block
        statement 1;
        statement 2;
        statement 3;
}       // End of the block
```

Any variable declared inside a block is only visible inside that block

Function

- A block that has been given a name
- Can be executed (called) by writing it’s name in other parts of the program

```
return-type function-name(parameter-list) {
        statement1;
        statement2;
        return expression;
}
```

Function types

- Global functions – Visible everywhere in you program after you declaration
- Member functions – A function that is a part of an object variable
- Lambda functions – A function created inline, or “on the fly”
- Function objects – An object possible to call as a function

Global function

- Also called a subroutine or procedure if there are no return value.
- Visible after declaration

```
... // Call foo here is a compilation error

void foo();

... // Ok to call foo
```

Function declaration and definition

- Declaration
  - Tells the compiler the function exists somewhere

```
void foo();
```

- Definition
  - Places function code in program

```
void foo() {
}
```

- Give the programmer a way to separate the program

Function result

```
return-type function-name(parameter-list) {
        statement1;
        statement2;
        return expression;
}
```

- **return-type** could be of any type that is declared in your program
- return **expression** must be of the return-type
- return statement exits the function

Function input parameters

```
return-type function-name(parameter-list) {
        statement1;
        statement2;
        return expression;
}
```

Zero or more specified in parameter list

Beware of automatic conversion if the compiler know a way to convert

Function - Best practice

- Always use const in case fundamental types
- Always use const& in case object types.

- Remove const only if you must

Function overload

- Different functions can have the same name
- Functions with same name must have different parameters
- Arguments given determine which function is actually called (closest match
- Compiler will select the “best match” among functions with the same name
- Return value is not considered even if different

Overloading example

```
int triangle_area(int base, int height);
int triangle_area(int side1, int side2, int side3);
int triangle_area(int side1, int side2, float angle);
int triangle_area(int side, float angle1, float angle2);

triangle_area(1, 1);
triangle_area(1, 1, 1);
triangle_area(1, 1, 1.0); // which is called?
triangle_area(1, 1.0, 1.0);
triangle_area(1, 1, 1.0f);
triangle_area(1, 1.0f, 1.0f);
```

Default values

- Parameters can be given default values
- Specified in declaration only, since definition may be unknown to compiler if program is in several files
- Default values can only be specified for last non-default parameter
- Can be omitted when calling the function

Default values

- Parameters can be given default values
- Specified in declaration only, since definition may be unknown to compiler if program is in several files
- Default values can only be specified for last non-default parameter
- Can be omitted when calling the function
- Combined with function overload then the declaration must be unambiguous!

Recursion

- A function can call itself. Helpful to solve complex problem where the solution space is exponential
- N-factorial example

```
int factorial(int n) {                 // 5 4 3 2 1
     if (n == 1)
          return 1;
     return n * factorial(n – 1);
}
```

File seperation

- Related functions can be gathered in one file to form a package.
- A package can be compiled separately, and do not need recompilation unless you change a package source file.
- Public declarations are place in a header file .h
- Definitions are placed in a implementation file .cc/.cpp
- Header and implementation files should have the same name, except for the extension

Header guard

- Header file must have a preprocessor guard to protect from multiple inclusion

```
#ifndef _FILE_NAME_H_
#define _FILE_NAME_H_
// public declarations
#endif
```

Compiling multiple files

- Never compile a header file. This will give you a cached version of that header file. This file have the file extension .gch. Remove this if you have it!

Lesson 2

- English in T11 – C building