CS135; User-Defined Functions Cont

Gael Zarco

April 5, 2025

1 Void Functions

`void` functions do not have a `functionType` and do not have any *formal parameters*.

Listing 1: Function Definition

```
1   void functionName(formal parameter list) {
2     statements
3   }
```

User-defined void functions can be placed either before or after the function `main`.

- If placed after `main` → *function prototype* required.
- Does not have a *return type*.
  - `return` statement without any value is typically used to exit the function early.

Coding Standard: Functions are protoyped before `main` and defined after `main`. Non-recursive functions should have **ONE** `return` statement.

1.1 Syntax

Listing 2: Formal Parameter List Syntax

```
1   dataType& variable, dataType& variable, ...
```

`&` in the listing above means *some* parameters will have `&` and some will not.

Listing 3: Function Call Syntax

```
1   functionName(actual parameter list);
```

Listing 4: Actual Parameter List Syntax

```
1   expression or variable, expression or variable, ...
```

1.2 Parameter Types

- **Value Parameters**: Formal parameters that receive a copy of the content of the corresponding actual parameters.
- **Reference Parameters**: Formal parameters that receive a copy of the location (memory address) of the corresponding actual parameter.

When attaching `&` after the `dataType` in the formal parameter list of a function, the variable after that `dataType` becomes a reference parameter.

2 Value Parameters

When function is called, for a value param, the actual value of the param is copied into the corresponding formal param. Meaning, there is no connection between the two as they are now separate entities. Therefore, the formal param is manipulated with its own copy of the data.

3 Reference Variables as Parameters

Reference params receive the address (memory location) of the actual param and enable direct manipulation of the actual param.
Useful in three situations, when:

- actual param needs to be altered.
- you want to return more than one value from a func (`return` statements can return only **ONE** value).
- passing the address would save memory space and time relative to copying a large amount of data.

Remember to attach `&` after the `dataType` in the formal param list of a function to convert the following `variable` into a reference param.

4 Value and Reference Parameters and Memory Allocation

Local Variables are vars declared in the body of the function and are allocated in the function data area. During data manip, the address stored in the formal param directs the computer to manip the data of the *memory cell* of that address (reference param). Value params possess a copy of the actual param. Therefore, reference params can permanently change the value of the actual param while value params cannot.

5 Reference Params and Value-Returning Functions

You can use reference params in value-returning functions, although it is not recommended. Try converting it to a void function instead. If a function needs to return more than one value, also convert it to a void function and use appropriate reference params.

6 Scope of An Identifier

Scope of an identifier refers to where in the program an identifier is accessible (identifier is the name of something in C++).

- **Local Identifier**: Identifiers declared within a func/block → not accessible outside of the func/block.
- **Global Identifier**: Identifiers delcared outside of every func definition.

C++ Does not allow nesting of functions.

Rules when an identifier is accessed:

1. Global identifierss accessible by func/block if:
    - The identifier is declared before the func definition.
    - The func name is different than the identifier.
    - All func params have names different names than the identifier.
    - All local identifiers (such as local vars) have names different than the name of the identifier.
2. **Nested Block** An identifier within a block is accessible:
    - Only within the block and not at any point before or after.
    - By blocks that are nested within that block, as long as the identifier does not share a name with the root block.

Notes about global variables:

1. Some compilers initialize global vars to default values.
2. **Scope Resolution Operator** in C++ is : :
3. By using scope resolution operator:
    - A global var declared before the func/block definition can be accessed by the func/block even if the func/block has an identifier with the same name as the globar var.

4. To access a global var declared after the func definition, the func must not contain any identifier with the same name.
    - Reserved word `extern` indicates that a global var has been declared elsewhere.

7 Global Variables, Named Constants, and Side Effects

A func that uses global variables is not independent.

If more than one func uses the same global var:

- It can make it difficult to debug code.
- Problems caused in one area of the program can appear in a separate form in another.

Global named constants have no side effects.

Coding Standard: Global variables are prohibited at all times. Global constants are acceptable.

8 Static and Automatic Variables

- **Automatic Variable**: Memory is allocated at block entry and deallocated upon block exit.
    - By default, variables declared within a block are automatic variables.
- **Static Variable**: Memory remains allocated as long as the program executes.
    - Global variables declared outside of any block are `static` variables.

Use reserved word `static` to delcare a `static` var.

Listing 5: `static` Variable Declaration

```
1    static dataType identifier;
```

9 Using Drivers and Stubs

A **Driver** is a separate program used to test a function.

When results calculated by one func are needed in another, use a **Function Stub**.

- A func stub is a function that is not fully coded.

10 Function Overloading

In a C++ program, several functions can have the same name. **Function Overloading** occurs when creating several functions with the same name.

Two funcs are said to have **different formal param lists** if both funcs have either:

- A different number of params.
- Conflicting `dataType` of formal params in atleast one instance.

Overloaded funcs must have different formal param lists. The **signature** is the name and formal param list of the func. The param list supplied in a call to an overloaded func determines which one is executed.

11 Functions with Default Parameters

In a func call, the number of actual and formal params must be the same.

- C++ relaxed this condition for funcs with default params.
- Can specify the value of a default param in the *function prototype*.
- If a value for a default param is not specified in a func call, the default value is used.

All default params must be the rightmost parameters of the func.

- If default vaulue is not specified $\rightarrow$ You must omit all of the arguments to its right.

Default values can be constants, global vars, or func calls. You cannot, however, assign a constant value as a default value to a reference param.