Parameters and communication

- Need some way to communicate between caller and callee (registers, memory, parameters)
- Call stack typical approach to support generalized recursion
- Wide variety of parameter passing mechanisms (by value, by reference, by value-return, by result, by name) and styles (positional vs keyword, optional parameters, variadic, etc)
- Overloading functions: using a single function name to represent different functions, identified by unique parameter lists
- Generic functions: subset of parameter types are not specified in function skeleton/template, are filled in by compiler based on actual parameter lists

Passing mechanisms

- By value (make copy, on stack or in heap with reference)
- By reference (implications for speed, memory, side effects)
- By value-return (copy to, copy back)
- By result (passing a variable to be “filled in”, either using reference or copying to stack then to target variable)
- By name (pass a name to callee, callee uses whichever variable is in most local scope with that name)

Parameter options

- Optional parameters (default values?), e.g. C++ or lisp
- Keyword vs positional parameters, e.g. lisp
- Variadic functions: arbitrary number of parameters, we've seen in lisp, can be done with macros in C or templates in C++ (will cover in the slides/video for lab 9)

Overloading functions

- Using the same name for a variety of functions, differentiated by parameter lists
- Compiler needs to be able to tell, from the actual parameters passed, which specific callee you wanted invoked
- Should this be checked at compile time, or run time?
- Implicit type conversion rules and optional parameters add complexity

Generic (e.g. templated) functions

- Subset of formal parameters are not given a type, function definition treated as a skeleton or template
- Compiler/interpretter examines actual calls to function to identify type used, compiler/intepretter generates full definition of callee based on those types
- Potential issues with seperate compilation (object file with function implementation may be compiled without knowledge of the calls from object files to be linked later)