CO650 Advanced Programming
C++ Functions and Parameters

Functions

What is a function?

- **A named block of code that can optionally return a value.**
- A building block for modular designed software
- The name must conform to **identifier naming rules**
- If no value is returned the type is void.
- Invoke the function using the name of the function followed by the call operator **()**

Variables & Types

- Valid identifier's name consists of one or more digits, letters or the underscore character.
- Must begin with a letter or underscore.
- Not be a reserved word
- They are **case sensitive**
- A variable has an associated type

Return Value

- If the function is returning a value then its type must match the type to be returned
- The return keyword finishes the function call and returns the value to the right
- Any statements placed after the return keyword will not be executed

Parameters

What are parameters?

- Data in the form of parameters can be passed to a function.
- The parameters (formal parameters) are a comma delimited set of types and identifier names placed with the function's ()
- Parameters are treated as local variables within the function.

Parameters Passing

- There are four parameter passing mechanisms
    - By value
    - By reference
    - By constant reference
    - By pointer
- Considerations before deciding which mechanism to use

    - Should the function be able to change the value of the variable passed to the parameter. If so, use by reference or pointer.
    - If not, and the type of the formal parameter is primitive use by value or in the case of a user defined type use constant reference.

Passing By Value

- A copy of the argument is assigned to the formal parameter
- Should only be used when the overhead of making the copy is acceptable
- Typically used for parameters of primitive types

Passing By Value

- & indicates the formal parameter accepts a reference.
- the referent and the reference share the same value. Any changes to reference will also change the referent.
- Commonly used with the user defined types as it avoids the copy overhead

SideNote - Difference of copy overhead between primitive types and user defined types

- Size of Data
    - Primitive types have fixed sizes, usually small, and are directly managed by the underlying hardware. Because of their small size, copying them is typically just a matter of moving a few bytes around, which is very fast.
    - User defined types can be much larger because they may contain several data members.
- Copy Operation Complexity
    - Copying a primitive type is straightforward because it involves direct bit-by-bit copying without additional logic required
    - Copying a user-defined type might involve more than bit-by-bit copy. If the class has defined a copy constructor, this function will execute during the copying process which might involve operations that have their own overhead
    - User defined type that does not have any special constructor or destructor will otherwise have default bitwise copy behavior
- Memory Access
    - Contiguous Memory:
        - This term refers to a sequence of memory addresses that are sequential and uninterrupted. Each memory address is adjacent to the one before it
        - Contiguous memory allocation is critical for certain operations or data structures, because access patterns benefit from predictability of memory addresses. It allows for efficient data access because the next address is predictable and loading data into the cache can be done in chunks.
        - Allocating large blocks of contiguous memory can be difficult, especially in systems that have been running for a while (due to fragmentation), and it might not always be possible to find a large enough block of free contiguous memory
    - Non-Contiguous Memory

    - This term refers to memory segments that are not sequentially adjacent in physical or virtual memory space. The data may be scattered across different memory locations
    - Non-contiguous memory allocation is more flexible because it allows small chunks of free memory to be utilized effectively, helping prevent wastage due to fragmentation.
    - Accessing non-contiguous memory can be slower since data isn't linearly laid out. The system often cannot predict which memory addresses will be accessed next, potentially leading to more cache misses and therefore, slower performance.

Passing By Constant Reference

- & indicates the formal parameter accepts a reference.
- const indicates that the value pointed to b the reference can't be changed within the function

Passing By Pointer

- Mechanism used within the C code and inherited by C++
- The address of a variable is passed as an argument to the formal parameter of type pointer

Arrays As Parameters

- The array name is a pointer to the first element in the array
- This can be passed as an argument to a function
- If the size of the array is not passed and loop exists within the function to loop over the array as parameter, loop would not know how many times it should iterate.

Multi Dimensional Arrays As Parameters

- Tow or more dimensional arrays can't be passed as a simple pointer
- The parameter accepting the array should be declared in the same way the argument is

```
returnType FunctionName(ArrayType identifier[size1][size2]
```

The first size may optionally be omitted

Constant Parameters

- By preceding a parameter with the const keyword we can ensure that its value is not changed within the function
- In the case of a reference, the value the reference refers to, can't be changed. The reference is a const anyway.

Default Parameters

- The last parameters can be assigned default values
- If when invoking the function, the arguments are not passed, the default values will be assigned
- It is illegal to assign the first parameter a default and not the second

Function's Declaration

- Sometimes referred to as the prototype
- Shows the function's interface. Not the statements within the body.
- Consists of Function name, parameter types and return type.
- Parameter names are optional
- Often related function declarations are placed in a header file

Why Declare Functions?

- In C++ a function must have been declared or defined before it can be invoked up to that point within the code
- It is possible to overcome this by declaring the function above the invocation within the code and then later defining it.

Forward Declaration

- Sometimes referred to as the prototype
- Shows the function's interface. Not the statements within the body.
- Consists of Function name, parameter types and return type.
- Parameter names are optional
- Often related function declarations are placed in a header file