Templates and generic programming

Improving on the first week’s assignment

Quiz: How did you calculate the machine precision?

1. Did you just have a main() function

2. Did you have three functions with different names?
   1. epsilon_float()
   2. epsilon_double()
   3. epsilon_long_double()

3. Did you have three functions with the same name?
   1. epsilon(float x)
   2. epsilon(double x)
   3. epsilon(long double x)

4. Or did you have just one function that could be used for any type?
   1. epsilon()
Generic algorithms versus concrete implementations

- Algorithms are usually very generic:
  for min() all that is required is an order relation “<”
  \[
  \min(x,y) = \begin{cases} 
  x & \text{if } x < y \\ 
  y & \text{otherwise}
  \end{cases}
  \]

- Most programming languages require concrete types for the function definition
  - C:
    ```
    int min_int(int a, int b) { return a<b ? a : b;}
    float min_float (float a, float b) { return a<b ? a : b;}
    double min_double (double a, double b) { return a<b ? a : b;}
    ```
  
  - Fortran:
    ```
    MIN(), AMIN(), DMIN(), ...
    ```

Function overloading in C++

- Solves one problem immediately: we can use the same name
  ```
  int min(int a, int b) { return a<b ? a : b;}
  float min (float a, float b) { return a<b ? a : b;}
  double min (double a, double b) { return a<b ? a : b;}
  ```

- Compiler chooses which one to use
  ```
  min(1,3); // calls min(int, int)
  min(1.,3.); // calls min(double, double)
  ```

- However be careful:
  ```
  min(1,3.1415927); // Problem! which one?
  min(1.,3.1415927); // OK
  min(1,int(3.1415927)); // OK but does not make sense
  or define new function double min(int,double);
  ```
How can several functions have the same name?

1. Why should it be a problem?
2. I don’t know
3. The compiler uses magic
4. It is a problem, but I know how it can be solved

C++ versus C linkage

- How can three different functions have the same name?
  - Look at what the compiler does
    ```
    c++ -c -save-temps -O3 min.cpp
    ```
  - Look at the assembly language file min.s and also at min.o
    ```
    nm min.o
    ```
  - The functions actually have different names!
    - Types of arguments appended to function name
  - C and Fortran functions just use the function name
    - Can declare a function to have C-style name by using `extern “C”`
      ```
      extern “C” { short min(short x, short y); }
      ```
Using macros (is dangerous)

- We still need many functions (albeit with the same name)

- In C we could use preprocessor macros:
  - `#define min(A,B) (A < B ? A : B)

- However there are serious problems:
  - No type safety
  - Clumsy for longer functions
  - Unexpected side effects:
    - `min(x++,y++)`; // will increment the smaller number twice!!!
    - // since this is: `(x++ < y++ ? x++ : y++)`

- Look at it:
  - `c++ -E minmacro.cpp`

---

Generic algorithms using templates in C++

- C++ templates allow a generic implementation:

  ```cpp
template <class T>
inline T min (T x, T y)
{
    return (x < y ? x : y);
}
```

- Using templates we get functions that
  - work for many types `T`
  - are optimal and efficient since they can be inlined
  - are as generic and abstract as the formal definition
  - are one-to-one translations of the abstract algorithm

Programming techniques for scientific simulations
**Usage Causes Instantiation**

```cpp
template <class T>
T min(T x, T y)
{
    return x < y ? x : y;
}

int x = min(3, 5);
int y = min(x, 100);
float z = min(3.14159f, 2.7182f);
```

```cpp
// T is int
int min<int>(int x, int y)
{
    return x < y ? x : y;
}

// T is float
float min<float>(float x, float y)
{
    return x < y ? x : y;
}
```

**Polymorphism**

- **Definition**: Using many different types through the same interface

- **What are the advantages?**
Generic programming process

- Identify useful and efficient algorithms
- Find their generic representation
  - Categorize functionality of some of these algorithms
  - What do they need to have in order to work in principle
- Derive a set of (minimal) requirements that allow these algorithms to run (efficiently)
  - Now categorize these algorithms and their requirements
  - Are there overlaps, similarities?
- Construct a framework based on classifications and requirements
- Now realize this as a software library

Generic Programming Process: Example

- (Simple) Family of Algorithms: min, max
- Generic Representation

\[
\begin{align*}
\text{min}(x, y) &= \begin{cases} 
x & \text{if } x < y \\
y & \text{otherwise}
\end{cases} \\
\text{max}(x, y) &= \begin{cases} 
x & \text{if } x > y \\
y & \text{otherwise}
\end{cases}
\end{align*}
\]

- Minimal Requirements?
- Find Framework: Overlaps, Similarities?
(Simple) Family of Algorithms: min, max

**Generic Representation**

\[
\begin{align*}
\text{min}(x,y) &= \begin{cases} 
x & \text{if } x < y \\
y & \text{otherwise}
\end{cases} \\
\text{max}(x,y) &= \begin{cases} 
x & \text{if } y < x \\
y & \text{otherwise}
\end{cases}
\end{align*}
\]

Minimal Requirements yet?
Find Framework: Overlaps, Similarities?

**Possible Implementation**

```cpp
template <class T>
T min(T x, T y)
{
    return x < y ? x : y;
}
```

What are the Requirements on T?
operator < , result convertible to bool
**Generic Programming Process: Example**

- Possible Implementation

  ```
  template <class T>
  T min(T x, T y)
  {
    return x < y ? x : y;
  }
  ```

- What are the Requirements on `T`?
  - `operator <`, result convertible to `bool`
  - Copy construction: need to copy the result!

---

**Generic Programming Process: Example**

- Possible Implementation

  ```
  template <class T>
  T const& min(T const& x, T const& y)
  {
    return x < y ? x : y;
  }
  ```

- What are the Requirements on `T`?
  - `operator <`, result convertible to `bool`
  - that’s all!
The problem of different types: manual solution

- What if we want to call min(1,3.141)?

```cpp
template <class R, class U, class T>
R const& min(U const& x, T const& y)
{
    return (x < y ? static_cast<R>(x) : static_cast<R>(y));
}
```

- Now we need to specify the first argument since it cannot be deduced.
  
  ```cpp```
  min<double>(1,3.141);
  min<int>(3,4);
  ```cpp```