TEMPLATE FUNCTIONS

Creating functions operating on general data types

Delroy A. Brinkerhoff

GENERALIZED DATA TYPES

```
//template <class T>
template <typename T>
void swap(T& x, T& y)
{
        T temp = x;
        x = y;
        y = temp;
}
```

```
double  data[1000];
        . . .
swap(data[i], data[j]);



Person  people = new Person[100];
        . . .
swap(people[x], people[y]);
```

PROCESSING TEMPLATE FUNCTIONS

- Library programmers put template functions in header files
- Application programs `#include` the header
- The preprocessor copies the header to the application
- The compiler deduces the arguments' type from their definition in the function
- The compiler replaces the template variable with the deduced type, making one instance of the function for each type replacement
- The compiler translates the function to machine code after the replacement

STANDARD TEMPLATE LIBRARY (STL) <algorithm>

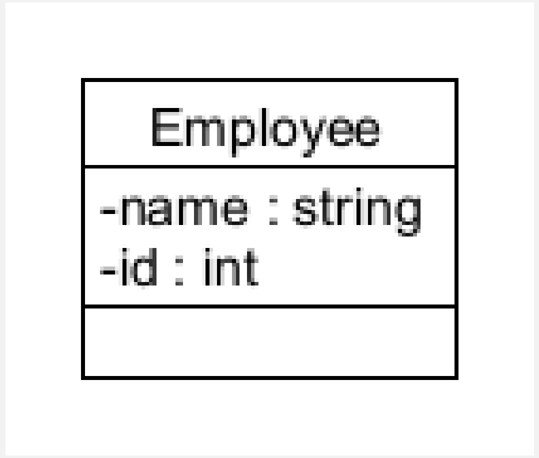
```
template <class T>
const T& min(const T& a, const T& b)
{
    return (a < b) ? a : b;
}
```

```
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    int x = 20;
    int y = 10;
    cout << min(x, y) << endl;

    return 0;
}
```

ESTABLISHING A "NATURAL" ORDERING



```
friend bool operator<(const Employee& e1, const Employee& e2)
{
    return e1.name < e2.name;
}


Employee e1("Dilbert", 123);
Employee e2("Alice", 987);
Employee e3 = min(e1, e2);
```

ORDERING WITH A COMPARATOR

```
template <class T, class Compare>
const T& min(const T& a, const T& b, Compare comp)
{
    return comp(a, b) ? a : b;
}
```

- A comparator implements a programmer-specified ordering
- Comparators can be passed as function pointers
- Comparators can be passed as objects overloading `operator<`

COMPARATOR FUNCTION POINTERS

```
friend bool comp1(const Employee& e1, const Employee& e2)
       { return e1.name < e2.name; }
friend bool comp2(const Employee& e1, const Employee& e2)
       { return e1.id < e2.id; }


Employee e1("Dilbert", 123);
Employee e2("Alice", 987);
Employee e3 = min(e1, e2, comp1);
Employee e4 = min(e1, e2, comp2);
```

MULTIPLE, FLEXIBLE ORDERINGS: COMPARATOR OBJECTS

```
class CompareName
{
    public:
        bool operator() (const Employee& e1, const Employee& e2)
            { return e1.getName() < e2.getName(); }
};


CompareName     c1;
Employee        e1("Dilbert", 123);
Employee        e2("Alice", 987);
Employee        e3 = min(e1, e2, c1);
```