Overloaded Functions

- Overloaded functions make programming convenient – only one function name for set of functions that perform similar operations.
- Each of the overloaded functions must be written individually.

```cpp
int square(int number) {
    return number*number;
}

double square(double number) {
    return number*number;
}
```

Different return type

Different parameter type

Same code
Overloaded Functions

- Overloaded functions make programming convenient – only one function name for set of functions that perform similar operations.
- Each of the overloaded functions must be written individually.

```c
int square(int number)
{
    return number*number;
}

double square(double number)
{
    return number*number;
}
```

What if I want to do this for 20 data types? And later another 20?

How can I solve this?
Function Templates

Function template:

- A *generic function* that can work with *any* data type
- Programmer writes the function specification, but *substitutes parameters for data types*
- When compiler encounters a call to the function, it *generates code* to handle the specific data type(s) used in the call, called a *template function*
Function Templates

Template prefix with keyword `template`

Generic data type starting with keyword `class`

Replace parameter type with generic type T

Replace return value type with generic type T

template <class T>
T square(T number)  
{  
  return number*number;  
}

int square(int number)  
{  
  return number*number;  
}

double square(double number)  
{  
  return number*number;  
}
### Function Templates

#### Template

```cpp
template <class T>
T square(T number)  
{
    return number*number;
}
```

#### Compiler generates

```cpp
int y, x=4;
y=square(x);
```

```cpp
double z,a=4.5;
z=square(a);
```

```cpp
int square(int number)  
{
    return number*number;
}
```

```cpp
double square(double number)  
{
    return number*number;
}
```
Function Templates

Function template:
• Merely specification of a function
• By itself it does not cause memory to be used
• Actual instance of function is created in memory when compiler encounters a call to template function
Using operators in function templates:

- Our square example uses the * operator
- If we want to use our own classes with function templates, then we must ensure that:
  
  Every operator used in function template has been defined for the object type.
  
  If passing a user-defined class object to the function template, all operators used in the function template must be defined for that class => class must overload those operators
Function Templates

Function templates with multiple types:
• Can use more than one generic type in a function template
• Each type must have their own parameter

```cpp
template <class T1, class T2>
int largest(const T1 &var1, const T2 &var2)
{
    if (sizeof(var1) > sizeof(var2))
        return sizeof(var1);
    else
        return sizeof(var2);
}
```
Function Templates

How to start with Function templates

• May be easier to first write a normal function for one data type, e.g. int
• Debug this function properly until it works correctly
• Then add template prefix (template<class T>)
• Then replace all references to int with T