Templating functions

Comp Sci 1570 Introduction to C++

Outline

1 Administrative notes

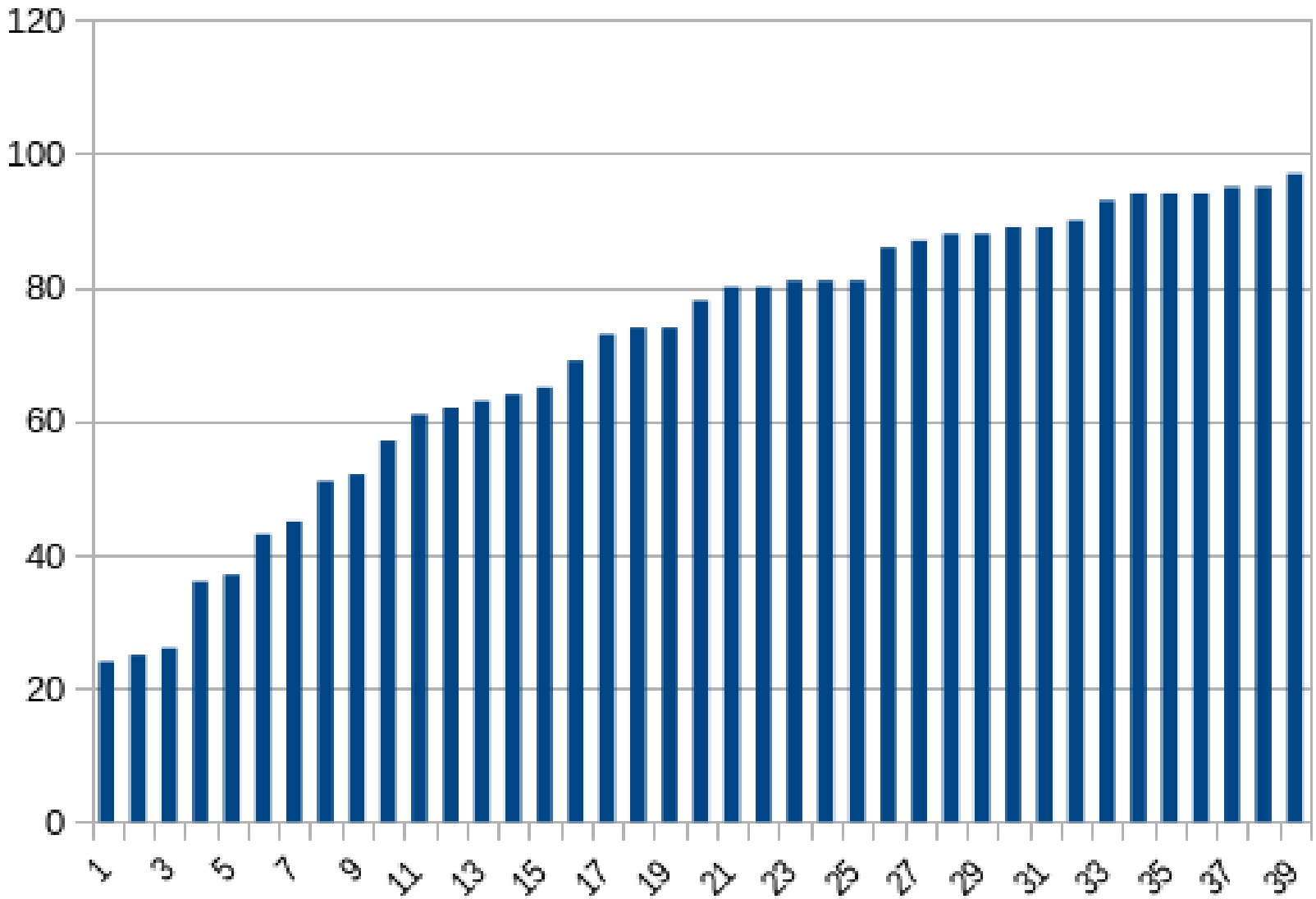
2 Review
- Functions
- Overloading

3 Templates
- Analogy
- Templates in C++
- Type placeholder
- Template parameter declaration
- Multiple parameters

4 Compiler processing

Test 1 grade distribution

grade on y, each student on x, sorted by y



- 50 or below should talk to me.

Outline

Outline

Functions help efficiency and code re-use

We have learned how to write functions that help make programs easier to write, safer, and more maintainable.

```
int max(int x, int y)
{
  return (x > y) ? x : y;
}
```

Outline

1 Administrative notes

2 Review
Functions
Overloading

3 Templates
Analogy
Templates in C++
Type placeholder
Template parameter declaration
Multiple parameters

4 Compiler processing

Overloading simplifies applying functions to types


```
int max(int x, int y)
{
  return (x > y) ? x : y;
}

double max(double x, double y)
{
  return (x > y) ? x : y;
}
```

- To re-use function names, we can overload functions.
- The body of the double version of max() is exactly the same as for the int version of max()!
- Implementation would work for all sorts of different types: chars, ints, doubles, and event strings.
- However, because C++ requires you to make your variables specific types, you're stuck writing one function for each type you wish to use.

Outline

Templates

- What is the general definition of a template?

Outline

1 Administrative notes

2 Review
Functions
Overloading

3 Templates
Analogy
Templates in C++
Type placeholder
Template parameter declaration
Multiple parameters

4 Compiler processing

Templates

- A template is a model that serves as a pattern for creating similar objects
- One type of template that is very easy to understand is that of a stencil.
- A stencil is an object (e.g. a piece of cardboard) with a shape cut out of it (eg. the letter J).
- By placing the stencil on top of another object, then spraying paint through the hole, you can very quickly produce stenciled patterns in many different colors!
- Note that you only need to create a given stencil once.
- You can then use it as many times as you like, to create stenciled patterns in whatever color(s) you like.
- Even better, you don't have to decide the color of the stenciled pattern you want to create until you decide to actually use the stencil.

Outline

Templating functions in C++

- A template is a blueprint or formula for creating a generic class or a function.
- Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.
- We define the function using placeholder types, called template typename parameters. Once we have created a function using these placeholder types, we have effectively created a "function stencil".
- The general form of a template function definition is shown here:

```
template <typename type> ret-type func-name(parameter list)
{
   // body of function
}
```

Outline

1 Administrative notes

2 Review
- Functions
- Overloading

3 Templates
- Analogy
- Templates in C++
- Type placeholder
- Template parameter declaration
- Multiple parameters

4 Compiler processing

Type placeholder

- You can name your placeholder types almost anything you want, so long as it's not a reserved word.
- However, in C++, it's customary to name your template types the letter T (short for "Type").
- Here's our new function with a placeholder type:

```
T max(T x, T y)
{
  return (x > y) ? x : y;
}
```

This is a good start – however, it won't compile because the compiler doesn't know what "T" is!

Outline

1 Administrative notes

2 Review
Functions
Overloading

3 Templates
Analogy
Templates in C++
Type placeholder
Template parameter declaration
Multiple parameters

4 Compiler processing

Template parameter declaration

- In order to make this work, we need to tell the compiler two things:
  - First, that this is a template definition
  - Second, that T is a placeholder type.
- We can do both of those things in one line, using what is called a template parameter declaration:

```
// this is the template parameter declaration
template <typename T>
T max(T x, T y)
{
    return (x > y) ? x : y;
}
```

Outline

1 Administrative notes

2 Review
- Functions
- Overloading

3 Templates
- Analogy
- Templates in C++
- Type placeholder
- Template parameter declaration
- Multiple parameters

4 Compiler processing

Multiple parameters

If the template function uses multiple template type parameter, they can be separated by commas:

```
template <typename T1, typename T2>
// template function here
```

Outline

1 Administrative notes

2 Review
- Functions
- Overloading

3 Templates
- Analogy
- Templates in C++
- Type placeholder
- Template parameter declaration
- Multiple parameters

4 Compiler processing

Example


```
template<class TYPE>
void PrintTwice(TYPE data)
{
  cout<<"Twice: " << data * 2 << endl;
}
```

Compiler processing

The first line of code:

template<**class** TYPE>

tells the compiler that this is a function-template. The actual meaning of TYPE would be deduced by compiler depending on the argument passed to this function. Here, the name, TYPE is known as template type parameter.

Compiler processing

For instance, if we call the function as:

```
PrintTwice(124);
```

TYPE would be replaced by compiler as int, and compiler would instantiate this template-function as:

```
void PrintTwice(int data)
{
  cout<<"Twice: " << data * 2 << endl;
}
```

Compiler processing

And, if we call this function as:

```
PrintTwice(4.5547);
```

It would instantiate another function as:

```
void PrintTwice(double data)
{
    cout<<"Twice: " << data * 2 << endl;
}
```

Compiler processing

If you call PrintTwice function with int and double parameter types, two instances of this function would be generated by the compiler:

```
void PrintTwice(int data) { ... }
void PrintTwice(double data) { ... }
```

- The code is duplicated, but these two overloads are instantiated by the compiler and not by the programmer.
- Benefit is that you need not to do copy-pasting the same code, or to manually maintain the code for different data-types, or to write up a new overload for new data-type that arrives later.
- Code size (at binary/assembly level) would increase, since there are now two function definitions.
- Effectively, for N number of data-types actually called in main, N instances of same function (i.e. overloaded functions) would be created.

Operators are functions too

When templating a function, you must insure that any operator used in the body of the templated function is implemented (supported) for the types you may pass in.