

# Templates

## Function Templates

- ◆ Describes functions in terms of a data-type parameters
- ◆ Allows a function to be reused independently of the data type
- ◆ You choose a representation for the data type
  - Typically T if only one type
    - ◆ `template <class T>`
- ◆ Templates can more than one data-type
  - `template <class T1, class T2>`

2

## Function Templates

```
template <class T>
int
search ( const T array[], const T key, int size ) {
    for ( int i = 0; i < size; i ++ ) {
        if key == array[i];
        return i
    }
    return -1
}
```

**T can represent ints, doubles, floats, user defined types, etc**

3

## Function Templates

- ◆ If you are using a user defined type, you may need to overload the `==` operator

```
class employee {
    string name
    Date hireDate

    int operator==(const employee&) const;
}
```

4

## Class Templates

- ◆ Templates can be used to create generic ADTs
  - Example - you wouldn't have to rewrite the list class each time you want to have a list of some new item. For programming assignment 1, you could have used 1 templated list class

5

## Class Templates

- ◆ Define the actual data type for each class when you declare an instance of the class
- ◆ Class templates are not compiled separately from the client program. Compiler must first see the actual data type
  - Typically class code is included in the .h file

6

## Class Templates

```
File – stuff.h
template <class T>
class Stuff {
    T blah;
    void set(T);
};

template <class T>
void Stuff<T>::set(T temp) {
    blah = temp
}

File – main.cc
int
main ()
    Stuff<int> s1;
    Stuff<double> s2;
    Stuff<Node> s3;
```

7

## Templated Pointer-based Linked List

```
template <class T>
class List {
private:
    Node<T>* head;
    Node<T>* last;
public:
    // Assume constructor – head = null, last = null
    void pushFront ( T );
    void pushback ( T );

    // removes the item at rank specified
    void remove ( int );
};

template <class T>
class Node {
public:
    Node<T>* next;
    T item;
};
```

8

## Templatized Pointer-based Linked List

```
template <class T>
void
List<T>::pushFront ( T a ) {
    Node<T>* ptr = new Node<T>;
    ptr->item = a
    ptr->next = head
    head = ptr
    if ( tail == NULL )
        tail = ptr
}
```

9

## Templatized Pointer-based Linked List

- ◆ In class exercise - Write the push back function templated

10