Module 19: Templates

Learning Objectives

1. Write code that uses function templates.
2. Write code that uses class templates.
3. Explain the advantages of using templates.

Process Skills

1. Critical thinking. Use prior knowledge to interpret and explain related concepts.
2. Problem solving. Use prior knowledge to solve programming problems that require templates.

Please fill in the roles for each member of your team. Take a look at the description of each role to see its responsibilities. If there are only three people in the group, please assign the same person to the **Presenter** and **Reflector** role. It is a good idea to select roles that you have not recently taken.

Team name: ______________________________________ Date: _______________

Role	**Team Member Name**
Manager. Keeps track of time and makes sure everyone contributes appropriately.	
Presenter. Talks to the facilitator and other teams.	
Reflector. Considers how the team could work and learn more effectively.	
Recorder. Records all answers and questions and makes the necessary submission.	

For virtual activities: Once you select your roles, change your Zoom name using the format and example below.
Format: Group X: First name, Last name initial / Role
Example: Group 1: Paul I / Presenter

Model 1. Function overloading (9 min) Start time: ______

```
// food.h
#include <iostream>

class Food {
 public:
  Food(const std::string &name, int days_until_expired)
      : name_(name), days_until_expired_(days_until_expired) { }

  void Display() const {
    std::cout << name_ << " (expires in " << days_until_expired_
              <<  " days)\n";
  }

  std::string Name() const { return name_; }
 private:
  std::string name_;
  int days_until_expired_;
};
```

```
// nonfooditem.h
#include <iostream>

class NonFoodItem {
 public:
  NonFoodItem(const std::string &name, const std::string &brand)
      : name_(name), brand_(brand) { }

  std::string Name() const { return name_; }

  void Display() const {
    std::cout << brand_ << " " << name_ << "\n";
  }
 private:
  std::string name_;
  std::string brand_;
};
```

```
// feature.h
#include "food.h"
#include "nonfooditem.h"

void FeatureProduct(const Food &product) {
 for (int i = 0; i < product.Name().length() + 20; i++) {
   std::cout << "=";
 }
 std::cout << "\n";
 std::cout << " Featured Product! " << product.Name() << "\n";
 for (int i = 0; i < product.Name().length() + 20; i++) {
   std::cout << "=";
 }
 std::cout << "\nIn stock: \n  ";
 product.Display();
}

void FeatureProduct(const NonFoodItem &product) {
 for (int i = 0; i < product.Name().length() + 20; i++) {
   std::cout << "=";
 }
 std::cout << "\n";
 std::cout << " Featured Product! " << product.Name() << "\n";
 for (int i = 0; i < product.Name().length() + 20; i++) {
   std::cout << "=";
 }
 std::cout << "\nIn stock: \n  ";
 product.Display();
}
```

```
// main.cc
#include <iostream>
#include "feature.h"

int main() {
 Food orange("orange", 3);
 FeatureProduct(orange);
 std::cout << "\n";

 NonFoodItem toothpaste("toothpaste", "Colgate");
 FeatureProduct(toothpaste);
 std::cout << "\n";

 return 0;
}
```

Screen Output

```
===========================
 Featured Product! orange
===========================
In stock:
  orange (expires in 3 days)

===============================
 Featured Product! toothpaste
===============================
In stock:
  Colgate toothpaste
```

1. Which FeatureProduct function definition was called when the orange object was passed as an argument in the main function (`FeatureProduct(orange)`)? Take note that the choices only show the function declarations for brevity. Place a check (✓) beside your answer.

   a. FeatureProduct(const Food &product); ✓
   b. FeatureProduct(const NonFoodItem &product);
   c. FeatureProduct(const std::string &product);

2. Which FeatureProduct function definition was called when the toothpaste object was passed as an argument in the main function (`FeatureProduct(toothpaste)`)? Take note that the choices only show the function declarations for brevity. Place a check (✓) beside your answer.

   a. FeatureProduct(const Food &product);
   b. FeatureProduct(const NonFoodItem &product); ✓
   c. FeatureProduct(const std::string &product);

3. Will the function call below work? Why or why not? Place a check (✓) beside your answer.

   ```
   FeatureProduct(2.5);
   ```

   a. No, because there is no FeatureProduct overload that accepts a double. ✓
   b. No, because doubles do not contain a Display member function.
   c. Yes, because double values can be displayed on screen

4. How should we modify the FeatureProduct function so we can pass it a new class that we plan to write (e.g., LimitedTimeProduct)? Place a check (✓) beside your answer.

   a. We need to create a new FeatureProduct function overload that takes in the new class as a parameter. ✓
   b. We can use the FeatureProduct functions that we already created.
   c. We need to create a FeatureProduct member function inside our new class.

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Model 2. Function template (9 min) Start time: ______

```
// feature.h
// Template function definitions can only be written in
// header files (not in .cc files)

template <typename T> // template parameter T
void FeatureProduct(const T &product) {
 for (int i = 0; i < product.Name().length() + 20; i++) {
   std::cout << "=";
 }
 std::cout << "\n";
 std::cout << " Featured Product! " << product.Name() << "\n";
 for (int i = 0; i < product.Name().length() + 20; i++) {
   std::cout << "=";
 }
 std::cout << "\nIn stock: \n  ";
 product.Display();
}
```

```
// No changes in food.h, nonfooditem.h, and main.cc
```

Screen Output

```
==========================
 Featured Product! orange
==========================
In stock:
  orange (expires in 3 days)

==============================
 Featured Product! toothpaste
==============================
In stock:
  Colgate toothpaste
```

5. The template parameter creates a placeholder for the data type of the object passed to the template function call. What is the syntax for creating a template parameter? Place a check (✓) beside your answer.

   a. template <typename identifier> ✓
   b. class <template identifier>
   c. template <identifier>

   * identifier refers to the name of the template parameter

6. When we pass orange to the FeatureProduct template function call, the compiler will internally set the template parameter's data type T to Food. As a result, it would seem that we have a FeatureProduct function taking in a Food parameter.

   template <typename T>
   void FeatureProduct(const T &product) → void FeatureProduct(const Food &product)

   If we pass toothpaste to the FeatureProduct template function, what data type will be assigned to the template parameter T? Place a check (✓) beside your answer.

   a. NonFoodItem ✓
   b. Food
   c. Product
   d. T

7. Review the FeatureProduct template function. When we pass orange to the FeatureProduct template function call, whose Name member function is called (`product.Name()`)? Place a check (✓) beside your answer.

   a. Food ✓
   b. NonFoodItem
   c. main

8. Review the FeatureProduct template function. When we pass toothpaste to the FeatureProduct template function call, whose Display member function is called (`product.Display()`)? Place a check (✓) beside your answer.

   a. Food
   b. NonFoodItem ✓
   c. main

9. Write an AnnounceDonation template function that accepts a single template parameter and does not return a value. You can use any name for the template parameter. We assume that arguments passed to the function provide a Display member function. Calling the AnnounceDonation function displays the following screen output. Take note that the underlined text is produced when calling the parameter's Display function.

   Thank you for donating <u>$10.00</u>
   Several people will benefit from your kindness.

```
// TODO: Provide member function definition
template <typename T>
void AnnounceDonation(const T& product) {
    std::cout << "Thank you for donating ";
    // TODO: call parameter's Display member function.
    product.Display();
    std::cout << "Several people will benefit from your kindness";
}
```

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Model 3. Class template (15 min) Start time: _______

```
// inventory.h
#include <iostream>
#include <vector>

template <typename T> // template parameter T
class Inventory {
 public:
  Inventory(const std::string &name)
      : name_(name) { }

  void Add(const T& item) {
    contents_.push_back(item);
  }

  void ListContents() {
    std::cout << name_ << " list:\n";
    for (T content: contents_) {
      content.Display();
    }
  }

 private:
  std::string name_;
  std::vector<T> contents_;
};
```

```
#include <iostream>
#include "food.h"
#include "nonfooditem.h"
#include "inventory.h"

int main() {
 Inventory<Food> titan_food("fruits");
 Food apple("apple", 7);
 Food orange("orange", 3);

 titan_food.Add(apple);
 titan_food.Add(orange);
 titan_food.Add(Food("banana", 5));
 titan_food.ListContents();
 std::cout << "\n";

 Inventory<NonFoodItem> titan_non_food("non-perishables");
 NonFoodItem toothpaste("toothpaste", "Colgate");
 NonFoodItem toilet_paper("toilet paper", "Charmie");

 titan_non_food.Add(toothpaste);
 titan_non_food.Add(toilet_paper);
 titan_non_food.Add(NonFoodItem("soap", "Dove"));
 titan_non_food.ListContents();
 return 0;
}
```

10. Much like template parameters for functions, we can also use them in class templates. What is the syntax for creating a class' template parameter? Place a check (✓) beside your answer.

    a. template <typename identifier> ✓
    b. class <template identifier>
    c. template <identifier>
    * identifier refers to the name of the template parameter

11. We can create objects from template classes by writing the class name followed by the template parameter's data type enclosed in < >. Where have we seen this notation before? Place a check (✓) beside your answer.

    a. Instantiating std::vector. ✓
    b. Comparing two boolean values/variables.
    c. Calling a function/member function.

12. We can use the template parameter anywhere inside a template class. Much like template parameters for functions, template parameters will serve as data type placeholders that are replaced after providing the data type during object instantiation. When we instantiate the titan_food Inventory, we also provide the Food data type (i.e., Inventory<Food> titan_food). The compiler will internally set the template parameter's data type T to Food and replace all instances of T to Food. As a result, it would seem that we have an Inventory class that uses Food objects for its parameters and member variables.

```
template <typename T>
class Inventory {
 public:
  Inventory(const std::string &name)
      : name_(name) { }

  void Add(const T& item) {
    contents_.push_back(item);
  }

  void ListContents() {
    std::cout << name_ << " list:\n";
    for (T content: contents_) {
      content.Display();
    }
  }

 private:
  std::string name_;
  std::vector<T> contents_;
};
```

```

class Inventory {
 public:
  Inventory(const std::string &name)
      : name_(name) { }

  void Add(const Food& item) {
    contents_.push_back(item);
  }

  void ListContents() {
    std::cout << name_ << " list:\n";
    for (Food content: contents_) {
      content.Display();
    }
  }

 private:
  std::string name_;
  std::vector<Food> contents_;
};
```

What data type will be assigned to the placeholder T, if we use NonFoodItem when instantiating the Inventory (i.e., Inventory<NonFoodItem> titan_non_food)? Place a check (✓) beside your answer.

a. Food
b. NonFoodItem ✓
c. std::vector

13. If we instantiate an Inventory object with the Food data type (i.e., Inventory<Food> titan_food), what kind of object can we pass to its Add member function when we call it? Place a check (✓) beside your answer.

    a. Food ✓
    b. NonFoodItem
    c. std::string

14. Complete the DonationManager template class. It stores the name of the manager and a std::vector to store donations added by the manager. The data type of std::vector's contents will depend on the template parameter. Create a Donate member function that accepts a single parameter and returns no values. It will add the parameter to the std::vector of donations. The parameter's data type will depend on the template. Create a ListDonations member function that accepts no values and does not return a value. It contains a loop going through each item in the std::vector and calls each element's Display member function to show information on the screen.

```
// TODO: Create class' template parameter


template <typename T>
class DonationManager {
 public:
  DonationManager(const std::string &name)
      : name_(name) { }

  // TODO: Complete the Donate member function definition
  void Donate(const T& item)

  {
    contents_.push_back(item);
  }

  // TODO: Complete the ListDonations member function definition
  void ListDonations()
  {
    std::cout << "Donations received:\n";
  // TODO: Complete the ranged for loop to create a variable
  // that stores each element of contents_


    for (T content: contents_) {
      content.Display();
    }
  }

 private:
  std::string name_;
  // TODO: Provide contents_ 's data type

  std::vector<T> contents_;
};
```

15. Complete the main function to create a DonationManager object that uses a Cash template. Donate the two cash objects to the DonationManager. Create a DonationManager that uses a NonFoodIdem template. Donate the two NonFoodItem to the DonationManager.

```
#include <iostream>
#include "cash.h"
#include "nonfooditem.h"
#include "donationmanager.h"
#include "announce.h"

int main() {
  // TODO: Complete code to instantiate a DonationManager object
that
  // uses a Cash template

  DonationManager<Cash> cash_donations("Cash donations");

  Cash donation1(100.0);
  Cash donation2(60.0);

  // TODO: Write code to donate the two cash objects
  cash_donations.Donate(donation1);
  cash_donations.Donate(donation2);

  cash_donations.ListDonations();

  std::cout << "\n";

  // TODO: Complete code to instantiate a DonationManager object
that
  // uses a NonFoodItem template

  DonationManager<NonFoodItem> titan_non_food("non-perishables");

  NonFoodItem toothpaste("toothpaste", "Colgate");
  NonFoodItem toilet_paper("toilet paper", "Charmie");

  // TODO: Write code to donate the two NonFoodItem objects
  titan_non_food.Donate(toothpaste);
  titan_non_food.Donate(toilet_paper);



  titan_non_food.ListDonations();

  return 0;
}
```

Reflector questions

1. What was the most useful thing your team learned during this session?

2. List the concepts you previously learned and used in this worksheet.

3. What was your group's biggest challenge in creating the AnnounceDonation template function and DonationManager class.