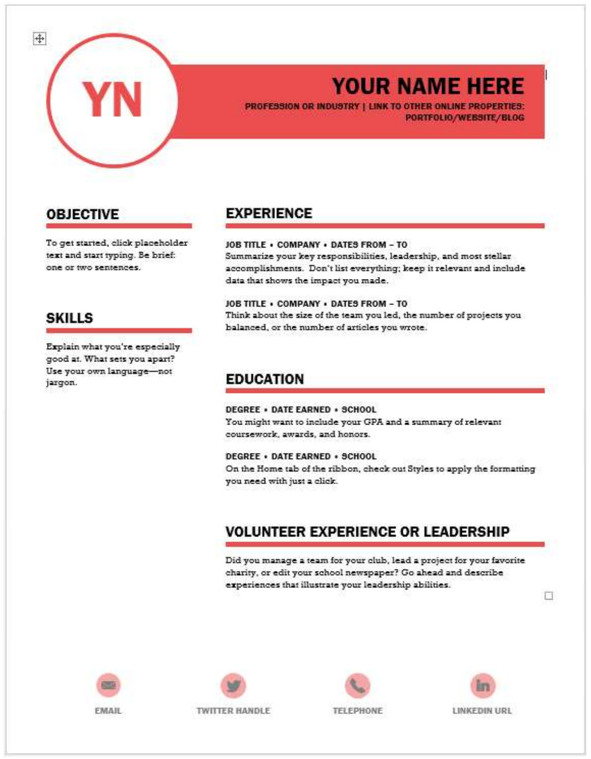
C++ Templates

An Introduction to **Generic Programming**

Prep: go to https://codeboard.io/signup and create an account
(Web IDE we use for tutorial exercises)

Templates outside of Programming



YN

YOUR NAME HERE

PROFESSION OR INDUSTRY | LINK TO OTHER ONLINE PROPERTIES: PORTFOLIO/WEBSITE/BLOG

OBJECTIVE

To get started, click placeholder text and start typing. Be brief: one or two sentences.

SKILLS

Explain what you're especially good at. What sets you apart? Use your own language—not jargon.

EXPERIENCE

JOB TITLE • COMPANY • DATES FROM – TO
Summarize your key responsibilities, leadership, and most stellar accomplishments. Don't list everything; keep it relevant and include data that shows the impact you made.

JOB TITLE • COMPANY • DATES FROM – TO
Think about the size of the team you led, the number of projects you balanced, or the number of articles you wrote.

EDUCATION

DEGREE • DATE EARNED • SCHOOL
You might want to include your GPA and a summary of relevant coursework, awards, and honors.

DEGREE • DATE EARNED • SCHOOL
On the Home tab of the ribbon, check out Styles to apply the formatting you need with just a click.

VOLUNTEER EXPERIENCE OR LEADERSHIP

Did you manage a team for your club, lead a project for your favorite charity, or edit your school newspaper? Go ahead and describe experiences that illustrate your leadership abilities.

EMAIL TWITTER HANDLE TELEPHONE LINKEDIN URL

- Microsoft Word Resume Template
- Fill in your
  - Name
  - Experience
  - Education
  - Et cetera…

- Can reuse **one** template for multiple **people**

Today’s Questions

- What are C++ templates?
- How can I write a function template?
- How can I write a class template?
- When should I write a C++ function/class template?

Templates in C++

- Generic Programming: a way to write code **once** that works for multiple **types**
  - Types: float, int, char, std::string, custom classes, etc.

- What can be templated in C++?
  - Functions
  - Classes
  - Some other stuff too (not covered)

- Reference: Problem Solving in C++, Chapter 17 (Walter Savitch)

Function Templates

Swapping Values: A Common Function

```cpp
void swap(int& a, int& b) {
  int temp = a;
  a = b;
  b = temp;
}

void swap(double& a, double& b)
{
  double temp = a;
  a = b;
  b = temp;
}
```

- Two functions
  - Both are named **swap**
  - Both do the same thing
  - Difference: int versus double
- Q: Will this compile? Why or why not? (Poll)

motivating_function_templates.cpp

Swapping Values: A Common Function

```
void swap(int& a, int& b) {
  int temp = a;
  a = b;
  b = temp;
}

void swap(double& a, double& b)
{
  double temp = a;
  a = b;
  b = temp;
}
```

motivating_function_templates.cpp

- Two functions
  - Both are named **swap**
  - Both do the same thing
  - Difference: int versus double
- Q: Will this compile?
  - Yes
  - The compiler calls based on parameter types
  - Function Overloading

Implementing swap for all types

Option 1: Write a function for each type

- Positives
  - Only support the types you want
- Negatives
  - Write new swap function for each new type (e.g. new class)
  - Error prone: make mistakes while repeating code

Option 2: Write a template where the type can change

- Positives
  - Supports all types that will compile
  - Code reuse avoids copy-paste
- Negatives
  - Longer compile times

Writing a Function Template

Tell the compiler we want to use templates

Specify the name for what will be *replaced* in the template

```
template<typename Type>
void swap(Type &a, Type& b) {
  Type temp = a;
  a = b;
  b = temp;
}
```

Use **Type** throughout the function. Compiler replaces **Type** with the actual datatype (e.g. int).

The swap function will work for **one Type** that has a proper (deep copy) **operator=**

templated_swap.cpp

Using a Function Template

```cpp
int main() {
  int i = 5, j = 9;
  std::cout << "Before Swap | ";
  std::cout << “i: " << i << " j: " << j << "\n";
  // compiler implicitly creates swap<int>
  swap(i, j); // implicit because i and j are int
  std::cout << "After Swap | ";
  std::cout << “i: " << i << " j: " << j << "\n";

  double x = 0.5, y = 4.6;
  std::cout << "Before Swap | ";
  std::cout << “x: " << x << " y: " << y << "\n";
  // explicitly create swap<double>
  swap<double>(x, y); // explicit because of <double>
  std::cout << "After Swap | ";
  std::cout << “x: " << x << " y: " << y << "\n";
}
```

templated_swap.cpp

```
i: 5 j: 9
i: 9 j: 5
x: 0.5 y: 4.6
x: 4.6 y: 0.5
```

Which of these will compile with no errors?

```
template<typename Type>
void swap(Type &a, Type& b)
```

```
int i = 5, j = 9;
double x = 0.5, y = 4.6;
char m = 'm', n = 'n';

swap(m, n); // Case A
swap(i, n); // Case B
swap<double>(i, j); // Case C
swap<int>(i, j); // Case D
swap<char>(x, y); // Case E
```

Poll

Case	Compiles?
A	
B	
C	
D	
E	

templated_swap.cpp

Which of these will compile with no errors?

```
template<typename Type>
void swap(Type &a, Type& b)
```

```
int i = 5, j = 9;
double x = 0.5, y = 4.6;
char m = 'm', n = 'n';

swap(m, n); // Case A
swap(i, n); // Case B
swap<double>(i, j); // Case C
swap<int>(i, j); // Case D
swap<char>(x, y); // Case E
```

Case	Compiles?
A	**Yes**, m and n are both of type char
B	**No**, an int cannot be swapped with a char
C	**No**, explicit request for double but given int
D	**Yes**, explicit request for int and given int
E	**No**, explicit request for char but given double

templated_swap.cpp

Implement a Templated minimum Function

```
template<typename Type>
Returns? minimum(Arguments?) {
    Implement function here
}
```

templated_minimum.cpp

Implement a Templated minimum Function

```
template<typename Type>
Returns? minimum(Type a, Type b) {

             Implement function here

}
```

templated_minimum.cpp

Implement a Templated minimum Function

```
template<typename Type>
Type minimum(Type a, Type b) {
```

Implement function here

```
}
```

templated_minimum.cpp

Procedure

- Will split you into breakout rooms.
  - Groups of 2.
- Go to the following url: https://codeboard.io/projects/206059
  - One of you can share screen with the other.
- Complete the code under "exercise1.h"
- Once you complete your code, hit "compile" and "run"
  - Top of the window.
  - Check if print statements are correct.
- Will be back in 4 minutes.
- This exercise is not graded.

templated_minimum.cpp

Implement a Templated minimum Function

```
template<typename Type>
Type minimum(Type a, Type b) {
  if(a < b) {
    return a;
  } else {
    return b;
  }
}
```

templated_minimum.cpp

When will templated minimum compile?

- When exactly **one** type is used
  - Like **swap**, the function does not support two different types per call

- When **operator<** is defined
  - **swap** needed **operator=**, here we need **operator<**

- When is **operator<** not defined?
  - For example, arrays: int x[20], y[20];

Class Templates

Applying Templates to Classes

- What would you template inside a class?
  - Member functions
  - Member data

- Examples:
  - A templated array class
  - A templated linked list
  - A templated binary search tree

A Simple Example: Pairs of Data

- Useful to store two pieces of data together
  - key + value pairs
    - Recall: ECE244 binary search tree nodes store (key, value) pairs
    - Key and value may not be the same. E.g.:
      - Key: **std::string** name
      - Value: **int** studentNumber
- Let’s create a class that can store two arbitrary pieces of data
  - First (any FirstT)
  - Second (any SecondT)
- Pair example usage:

```
my_binary_tree.insert (pair_to_insert);  // key & value
```

Pair Example

- Pair is an association of two different types of variables
  - Two members: first, second
  - Members of different types
- Example: Two instantiations of the pair object

first (type std::string)	second (type int)
Apple	**4**

first (type double)	second (type std::string)
42.0	**Forty-two**

An Example pair Class

```
template<typename FirstT, typename SecondT> class pair {
public:
  pair(FirstT first_val, SecondT second_val);

  void set_first(FirstT first_val);
  void set_second(SecondT second_val);

  FirstT get_first() const;
  SecondT get_second() const;

private:
  FirstT first;
  SecondT second;
};
```

templated_pair.cpp

An Example pair Class

We can have more than one type templated

```
template<typename FirstT, typename SecondT> class pair {
public:
  pair(FirstT first_val, SecondT second_val);

  void set_first(FirstT first_val);
  void set_second(SecondT second_val);

  FirstT get_first() const;
  SecondT get_second() const;

private:
  FirstT first;
  SecondT second;
};
```

templated_pair.cpp

Defining Member Functions for pair

- Definitions **must** be in the header file
  - Compiler must be able to generate code whenever it sees a new templated pair
  - Cannot separate interface (*.h or *.hpp) from definition (*.cpp) when using templates

Implementing pair

```
template<typename FirstT, typename SecondT> class pair {
public:
  pair(FirstT first_val, SecondT second_val)
   : first(first_val), second(second_val) { }

  void set_first(FirstT first_val) {
    first = first_val;
  }

  // ... etc
```

templated_pair.hpp

Using pair

Compiler creates a **pair** that takes an int and a float

pair1
int first = 1
float second = 2.5f

```
pair<int, float> pair1(1, 2.5f);
std::cout << pair1.get_first() << " ";
std::cout << pair1.get_second() << "\n";

pair<std::string, double> exam("Mario", 48.5);
std::cout << exam.get_first() << " received ";
std::cout << exam.get_second() << "/100 on the exam.\n";

exam.set_second(49.0);
std::cout << exam.get_first() << "'s grade changed to ";
std::cout << exam.get_second() << " by a nice professor.\n";
```

exam
std::string first = “Mario”
double second = 48.5

Compiler creates **another** **pair** that takes a std::string and double

templated_pair.cpp

Demo 1

Template example: my_max

Tips for Writing Templates

- Don’t
  - Template errors can be very complex and overwhelming for beginners
  - Many templated functions and classes already exist
    - The **Standard Template Library** is your friend!
- If you do…
  1. Write the code for one type first
  2. Test it
  3. Convert it to a template
  4. Test again with multiple types