物件導向程式設計

Template (1/3)

Joseph Chuang-Chieh Lin
Dept. CSIE, Tamkang University

Platform

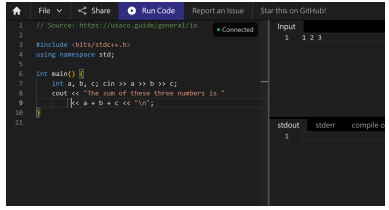
- Dev-C++

  Click here to download.

  Note: Please use this version otherwise you can't compile your programs/projects in Win10.

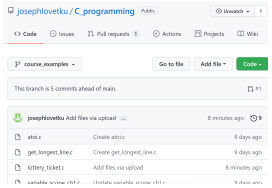
- OnlineGDB (https://www.onlinegdb.com/)

- Real-Time Collaborative Online IDE (https://ide.usaco.guide/)



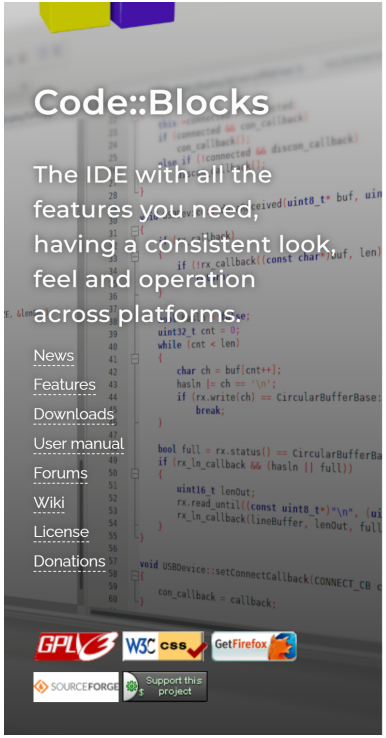
- Other resources:
  - MIT OpenCourseWare - Introduction to C++ [link].
  - Learning C++ Programming [Programiz].
  - GeeksforGeeks [link]

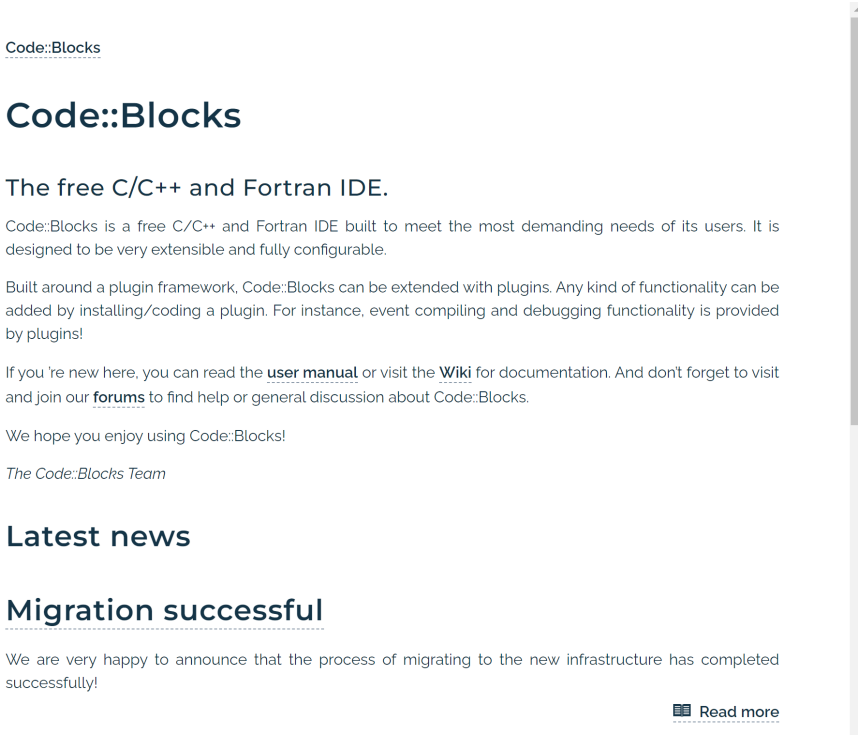
My GitHub page:
click the link here to visit.



Platform/IDE

- https://www.codeblocks.org/





Template

- **Goal**: Pass **data type** as a **parameter** so that we don't have to write the same code or the same function for different data types.
- For example, we can write one function `sort()` and pass data type as a parameter so that we can sort data of many kinds of types.
- When does template expand/kick in?

Template

- **Goal**: Pass **data type** as a **parameter** so that we don't have to write the same code or the same function for different data types.
- For example, we can write one function `sort()` and pass data type as a parameter so that we can sort data of many kinds of types.
- When does template expand/kick in?
  - Compile time.
  - The source code contains only the function or class, yet the compiled code contains multiple copies of it.

Function Template

- A generic function that can be used for different data types.

```
template <class T> T flex_max(T x, T y) {
    return (x > y) ? x : y;
}
```

```
int main() {
    cout << flex_max<int>(3, 7) << endl;
    cout << flex_max<double>(3.0, 7.0)
         << endl;
    cout << flex_max<char>('g', 'e')
         << endl;

    return 0;
}
```

Function Template

- A generic function that can be used for different data types.

```
template <typename T> T flex_max(T x, T y) {
    return (x > y) ? x : y;
}
```

either "`typename`" or "`class`" will be fine

```
int main() {
    cout << flex_max<int>(3, 7) << endl;
    cout << flex_max<double>(3.0, 7.0)
         << endl;
    cout << flex_max<char>('g', 'e')
         << endl;

    return 0;
}
```

Function Template

```
template <class T> T flex_max(T x, T y) {
    return (x > y) ? x : y;
}
```

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

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

```
char flex_max(char x, char y) {
    return (x > y) ? x : y;
}
```

Function Template

- A generic function that can be used for different data types.

```
template <class T> void try_swap(T& x, T& y) {
    T temp = x;
    x = y;
    y = temp;
}
```

```
int main() {
    int a = 5, b = 3;
    double c = 11.2, d = 23.7;
    try_swap<int>(a, b);
    try_swap<double>(c, d);
    cout << a << ", " << b << endl;
    cout << c << ", " << d << endl;
    return 0;
}
```

Function Template

- A generic function that can be used for different data types.

```
template <class T>
void try_swap(T& x, T& y) {
    T temp = x;
    x = y;
    y = temp;
}
```

```
int main() {
    int a = 5, b = 3;
    double c = 11.2, d = 23.7;
    try_swap<int>(a, b);
    try_swap<double>(c, d);
    cout << a << ", " << b << endl;
    cout << c << ", " << d << endl;
    return 0;
}
```

Example: Bubble sort

https://www.geeksforgeeks.org/templates-cpp/

```
template <class T> void bubbleSort(T a[], int n) {
    for (int i=0; i<n-1; i++)
        for (int j=n-1; i<j; j--)
            if (a[j] < a[j-1])
                swap(a[j], a[j-1]);
}
```

i				j
2	1	6	4	5

```
int main() {
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);

    bubbleSort<int>(a, n);
    for (int i = 0; i < n; i++) // print the sorted array
        cout << a[i] << " ";
    cout << endl;

    return 0;
}
```

Example: Bubble sort

https://www.geeksforgeeks.org/templates-cpp/

```
template <class T> void bubbleSort(T a[], int n) {
    for (int i=0; i<n-1; i++)
        for (int j=n-1; i<j; j--)
            if (a[j] < a[j-1])
                swap(a[j], a[j-1]);
}
```

i			j	
2	1	6	**5**	4

```
int main() {
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);

    bubbleSort<int>(a, n);
    for (int i = 0; i < n; i++) // print the sorted array
        cout << a[i] << " ";
    cout << endl;

    return 0;
}
```

Example: Bubble sort

https://www.geeksforgeeks.org/templates-cpp/

```
template <class T> void bubbleSort(T a[], int n) {
    for (int i=0; i<n-1; i++)
        for (int j=n-1; i<j; j--)
            if (a[j] < a[j-1])
                swap(a[j], a[j-1]);
}
```

i		j		
2	1	6	5	4

```
int main() {
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);

    bubbleSort<int>(a, n);
    for (int i = 0; i < n; i++) // print the sorted array
        cout << a[i] << " ";
    cout << endl;

    return 0;
}
```

Example: Bubble sort

https://www.geeksforgeeks.org/templates-cpp/

```
template <class T> void bubbleSort(T a[], int n) {
    for (int i=0; i<n-1; i++)
        for (int j=n-1; i<j; j--)
            if (a[j] < a[j-1])
                swap(a[j], a[j-1]);
}
```

i	j			
2	**6**	1	5	4

```
int main() {
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);

    bubbleSort<int>(a, n);
    for (int i = 0; i < n; i++) // print the sorted array
        cout << a[i] << " ";
    cout << endl;

    return 0;
}
```

Example: Bubble sort

https://www.geeksforgeeks.org/templates-cpp/

```
template <class T> void bubbleSort(T a[], int n) {
    for (int i=0; i<n-1; i++)
        for (int j=n-1; i<j; j--)
            if (a[j] < a[j-1])
                swap(a[j], a[j-1]);
}
```

j

i

6	2	1	5	4

```
int main() {
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);

    bubbleSort<int>(a, n);
    for (int i = 0; i < n; i++) // print the sorted array
        cout << a[i] << " ";
    cout << endl;

    return 0;
}
```

Class Template

- Similar to function templates, but useful for **classes** (e.g., LinkedList, BinaryTree, Stack, Queue, ...)

Example: A Generic Array

```
template <class T> class Array {
private:
    T* ptr;
    int size;

public:
    Array(T arr[], int s);
    void print();
};
```

```
template <class T>
Array<T>::Array(T arr[], int s) {
    ptr = new T[s];
    size = s;
    for (int i = 0; i < size; i++)
        ptr[i] = arr[i];
}
```

```
template <class T>
void Array<T>::print() {
    for (int i = 0; i < size; i++)
        cout << " " << *(ptr + i);//OK!
    cout << endl;
}
```

```
int main() {
    int arr[5] = { 1, 2, 3 };
    Array<int> a(arr, 3);
    a.print();
    return 0;
}
```

Example: A Generic Array

```
template <class T> class Array {
private:
    T* ptr;
    int size;

public:
    Array(T arr[], int s);
    void print();
};
```

```
template <class T>
Array<T>::Array(T arr[], int s) {
    ptr = new T[s];
    size = s;
    for (int i = 0; i < size; i++)
        ptr[i] = arr[i];
}
```

```
template <class T>
void Array<T>::print() {
    for (int i = 0; i < size; i++)
        cout << " " << ptr[i]; // OK!
    cout << endl;
}
```

```
int main() {
    int arr[5] = { 1, 2, 3 };
    Array<int> a(arr, 3);
    a.print();
    return 0;
}
```

Another Example: Matrix

https://www.cs.uregina.ca/Links/class-info/115/07-templates/

```
class Matrix {
  private:
    int twoDimArray[MAXROWS][MAXCOLS];
    int rows;
    int cols;
  public:
    Matrix(); // constructor
    void printMatrix();
    void setElement(int row, int col, int value); //set an element of the matrix
    void setMatrix(int [][MAXCOLS]); //set the twoDimArray to what is sent
    void addMatrix(int [][MAXCOLS]); //add an array to twoDimArray
    void addMatrix(int [][MAXCOLS], int[][MAXCOLS]); //add two arrays together
};
```

Try to make it a template class

Another Example: Matrix

https://www.cs.uregina.ca/Links/class-info/115/07-templates/

```
template <typename M_type>
class Matrix {
  private:
    M_type twoDimArray[MAXROWS][MAXCOLS];
    int rows;
    int cols;
  public:
    Matrix();
    void printMatrix();
    void setElement(int row, int col, M_type value); //set an element of the matrix
    void setMatrix(M_type [][MAXCOLS]); //set the twoDimArray to what is sent
    void addMatrix(M_type [][MAXCOLS]); //add an array to twoDimArray
    void addMatrix(M_type [][MAXCOLS], M_type[][MAXCOLS]); //add two arrays together
};
```

Class Instantiation

https://www.cs.uregina.ca/Links/class-info/115/07-templates/

```
Matrix<float> floatMatrix;
```

Member Function Definition

https://www.cs.uregina.ca/Links/class-info/115/07-templates/

```
void Matrix::addMatrix(int otherArray[][MAXCOLS]) {
   for (int i=0; i< rows; i++) {
      for(int j=0; j< cols; j++) {
       twoDimArray[i][j] += otherArray[i][j];
      }
   }
}
```

```
template <typename M_type>
void Matrix<M_type>::addMatrix(M_type otherArray[][MAXCOLS]) {
   for (int i=0; i< rows; i++) {
      for(int j=0; j< cols; j++) {
       twoDimArray[i][j] += otherArray[i][j];
      }
   }
}
```

Function Overloading vs. Templates

- When should we use templates?
  - When we want to perform **the same action** just on different types.

```
template <typename T>
T foo(const T& a, const T& b) { return a + b; }
```

Function Overloading vs. Templates

- When should we use templates?
  - When we want to perform **the same action** just on different types.
- When should we use function overloading?
  - When we may apply **different operations on different types**.

```
class Foo{ void run() const {} };

void foo(int i) { std::cout << "i = " << i << "\n"; }
void foo(const Foo& f) { f.run(); }
```

Exercise: `print_max()`

```
Template ... {
/* implement the function template print_max*/
}
```

```
int main() {
    int a[6] = { 10, 50, 30, 40, 20, -20 };
    float b[] = { 2.3, 0.0, -1.2, 17.2 };
    char c[] = "TKUCS";
    int n1 = sizeof(a) / sizeof(a[0]);
    int n2 = sizeof(b) / sizeof(b[0]);
    int n3 = sizeof(c) / sizeof(c[0]);
    print_max<int>(a, n1);
    print_max<float>(b, n2);
    print_max<char>(c, n3);
    return 0;
}
```

Output:

```
50
17.2
U
```