# Template in C++



- Mục đích của template (mẫu) là hỗ trợ tái sử dụng mã.
- Có 2 loại mẫu: hàm mẫu (*template function*) và lớp mẫu (*template class*).
- Hàm/lớp mẫu là hàm/lớp tổng quát (generic function/class), không phụ thuộc kiểu dữ liệu.
  - Mã người dùng phải khai báo kiểu dữ liệu cụ thể khi sử dụng hàm/lớp mẫu.
- Khai báo hàm/lớp mẫu chỉ tạo "khung".
  - Trình biên dịch sẽ tạo mã thực thi từ "khung" chỉ khi nào lớp/hàm mẫu được dùng đến.

# Hàm mẫu

- Giải thuật độc lập với kiểu dữ liệu được xử lý.

Ví dụ: Tìm số lớn nhất: **`max = (a > b) ? a : b;`**

- Cài đặt bằng ngôn ngữ lập trình

```
int max(int a, int b) {
  return a > b ? a : b;
}
int m = 43, n = 56;
cout << max(m, n) << endl; // 56
double x = 4.3, y = 5.6;
cout << max(x, y) << endl; // 5
```

- Quá tải hàm **`max()`** là một giải pháp.

```
double max(double a, double b);
```



- Quá tải hàm sẽ gây ra tình trạng "lặp lại mã".

  → Sử dụng hàm mẫu.

```
template <class TYPE>
TYPE max(const TYPE & a, const TYPE & b) {
   return a > b ? a : b;
}
int m = 43, n = 56;
cout << max(m, n) << endl;   // 56
double x = 4.3, y = 5.6;
cout << max(x, y) << endl;   // 5.6
```

- Chương trình vẫn thực thi với kiểu dữ liệu **`string`**.

```
string s = "abc", t = "xyz";
cout << max(s, t) << endl;   // xyz
```

```
template <class TYPE>
int count(const TYPE *array, int size, TYPE val) {
  int cnt = 0;
  for (int i = 0; i < size; i++)
    if (array[i] == val) cnt++;
   return cnt;
}

double b[3] = {3, -12.7, 44.8};
string c[4] = {"one", "two", "three", "four"};
cout << count(b, 3, 3) << endl;  // illegal
cout << count(c, 4, "three");    // illegal
cout << count(b, 3, 3.0) << endl;      // legal
cout << count(c, 4, string("three"));  // legal
```



## Mô phỏng mảng hai chiều

```
template <class TYPE>
void dim2(TYPE ** & prow, int rows, int cols) {
  TYPE * pdata = new TYPE [rows * cols];
  prow = new TYPE * [rows];
  for (int i = 0; i < rows; i++)
    prow[i] = pdata + i * cols;
}
```

```
template <class TYPE>
void display(TYPE **a, int rows, int cols) {
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }
}
/* ---------------------------------------- */
template <class TYPE>
void free2(TYPE **pa) {      // free 2D memory
   delete [] *pa;            // free data
   delete [] pa;             // free row pointers
}
```



```
int main() {
  int **a;
  dim2(a, rows, cols);   // 2D array of integers
  int inum = 1;
  for (i = 0; i < rows; i++)
    for (j = 0; j < cols; j++)
      a[i][j] = i + j;
  display(a, rows, cols);
  free2(a);

  double **b;
  dim2(b, rows, cols);   // 2D array of doubles
  for (i = 0; i < rows; i++)
    for (j = 0; j < cols; j++)
      b[i][j] = (i + j) * 1.1;
  display(b, rows, cols);
  free2(b);
```

```
  Complex **a;
  dim2(a, 3, 4);
  for (i = 0; i < 3; i++)
    for (j = 0; j < 4; j++) {
      a[i][j].real(1.1);
      a[i][j].imag(2.2);
    }
  display(a, 3, 4);
  free2(a);
  …
}
```



# Lớp mẫu

- Lớp mẫu cho phép tạo ra những lớp tổng quát.
  - Loại trừ khả năng sử dụng *lặp mã lệnh* khi xử lý những kiểu dữ liệu khác nhau.
  - Thiết kế thư viện thuận tiện và dễ quản lý hơn.
- Những lớp chỉ làm việc trên một kiểu dữ liệu thì không nên tổng quát hóa.
  - Lớp **`Complex`**: chỉ làm việc với **`double`**.
  - Lớp **`String`** (user-defined): chỉ làm việc với ký tự.
- Những lớp chứa (*container class*) như **`Stack`**, **`List`**, … nên được tổng quát hóa.

```
class Stack {
  private:
    int *v;             // pointer to integer data
    int top;            // top of Stack
    int len;            // length of Stack
  public:
    Stack(int size = MAXLEN) : top(0) {
      v = new int[len = size];
    }
    ~Stack() { delete [] v; }
    void push(int d) { v[top++] = d; }
    int pop() { return v[--top]; }
    bool empty() const { return top == 0; }
    bool full() const { return top == len; }
    int length() const { return len; }
    int nitems() const { return top; }
};
```



```
template <class TYPE>
class Stack {
  private:
    TYPE * v;
    int top;
    int len;
    void copy(const Stack<TYPE> &);
    void free() { delete [] v; }
  public:
    Stack(int size = MAXLEN);
    Stack(const Stack<TYPE> & s) { copy(s); }
    ~Stack() { free(); }
    void push(const TYPE & d) { v[top++] = d; }
    TYPE pop() { return v[--top]; }
    Stack<TYPE> & operator=(const Stack<TYPE> &s);
    …
};
```

```
template <class TYPE>
Stack<TYPE>::Stack(int size) {
  v = new TYPE [len = size];
  top = 0;
}

template <class TYPE>
void Stack<TYPE>::copy(const Stack<TYPE> & s) {
  v = new TYPE [len = s.len];
  top = s.top;
  for (int i = 0; i < len; i++)
    v[i] = s.v[i];
}
```



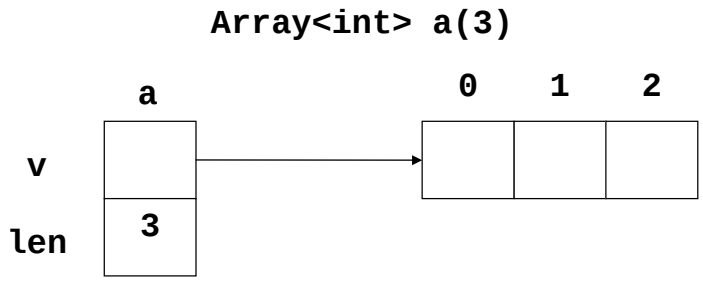
```
template <class TYPE>
Stack<TYPE> & Stack<TYPE>::operator=(const
                                    Stack<TYPE> & s) {
  if (this != &s) {
    free();
    copy(s);
  }
  return *this;
}

template <class TYPE>
void store(const TYPE * b, int len) {
  Stack<TYPE> s(len);
  …
}
```

```
int main() {
  Stack<int> s(10);
  s.push(100);
  cout << s.pop() << endl;

  Stack<string> * st = new Stack<string>(5);
  st->push("abc");
  cout << st->pop() << endl;
  delete st;

  int buf[10];
  store(buf, 10);
  string title[20];
  store(title, 20);
  …
}
```



## Xây dựng lớp mảng 1 chiều tổng quát

- Mảng 1 chiều của C++ gặp những hạn chế sau:
  - Khai báo kích thước mảng tĩnh.
  - Không kiểm tra giới hạn mảng.
  - Không cho phép gán hai mảng cùng kiểu.
- Mảng 1 chiều tổng quát sẽ khắc phục những hạn chế trên.



```
template <class TYPE>
class Array {
  private:
    TYPE * v;
    int  len;
    void range(int) const;
    void copy(const Array<TYPE> &);
    void free() { delete [] v };
  public:
    Array(int length = 1);
    Array(const Array<TYPE> & a);
    ~Array() { free(); }
    Array<TYPE> & operator=(const Array<TYPE> &);
    TYPE & operator[](int);
    const TYPE & operator[](int) const;
    int length() const { return len; }
};
```



```
class ArrayError {
  private:
    char buf[80];
  public:
    ArrayError(int s) {
      sprintf(buf, "%d is an illegal length", s);
    }
    ArrayError(int index, int maxindex) {
      sprintf(buf, "subscript %d out of bounds,
        max subscript = %d", index, maxindex - 1);
    }
    void response() const { cerr << buf << endl; }
};
```

```
template <class TYPE>
void Array<TYPE>::range(int i) const {
  if (i < 0 || i >= len) throw ArrayError(i, len);
}

template <class TYPE>
Array<TYPE>::Array(int length) {
  if (length <= 0) throw ArrayError(length);
  v = new TYPE [len = length];
}

template <class TYPE>
Array<TYPE>::Array(const Array<TYPE> & a) {
  copy(a);
}
```



```
template <class TYPE>
Array<TYPE> & Array<TYPE>::operator=
                          (const Array<TYPE> & a) {
  if (this != &a) {
    free();
    copy(a);
  }
  return *this;
}

template <class TYPE>
ostream & operator<<(ostream & os, const
                                  Array<TYPE> & a) {
  for (int i = 0; i < a.length(); i++)
    os << a[i] << ' ';
  return os;
}
```

```
template <class TYPE>
TYPE & Array<TYPE>::operator[](int i) {
  range(i);
  return v[i];
}

template <class TYPE>
const TYPE & Array<TYPE>::operator[](int i) const {
  range(i);
  return v[i];
}
```



```
int main() {
  try {
    Array<int> a(10), b(10);
    for (int i = 0; i < a.length(); i++)
      a[i] = i + 1;
    cout << a << endl;
    b = a;
    cout << b << endl;

    Array<double> c(10);
    for (i = 0; i < c.length(); i++)
      c[i] = 0.5 * i;
    const Array<double> d = c;
    cout << d << endl;
  }
  catch (const ArrayError & e) {
    e.response();    return 1;
  }
  return 0;
}
```