3.1 Templates in C++

3.1

Template in C++

- A mechanism to **parameterize** the **target data type** and **instantiate** it to a proper data type at compile time.
- Using the keyword **template** followed by parameter type list **<typename T>** or **<class T>**.

3.1.1

Function Template

Ordinary function

```
int sum(int * data, const int SIZE)
{
  int  sum=0 ;
  for( int i =0; i < SIZE; i++)
  {
    sum  +=  data[i];
  }
  return sum;
}
```

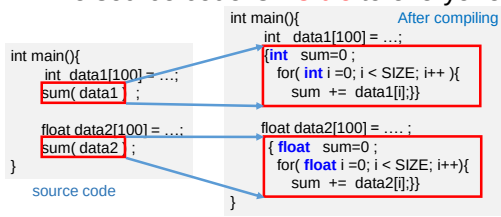
Template function

```
template < class T >
T sum(T * data, const int SIZE)
{
  T sum=0 ;
  for( int i =0; i < SIZE; i++)
  {
    sum  +=  data[i];
  }
  return sum;
}
```

3.1.1

Function Template

- The compiler will insert the code with proper data type at compile time.
- The source code is visible to everyone!



3.1.2
P3.4

The Class *Bag* Containing int

```
class Bag
{
public:
  Bag(int bagCapacity =
10);                        // Constructor
 ~Bag();                    // Destructor

  int Size() const;         // Return the number of elements

  bool IsEmpty() const;     // Check if bag is empty

  int Element() const;      // Return an element in the bag

  void Push(const int);     // Insert an integer into the bag
  void Pop()                // Delete an integer from the bag

private:
  int *array;               // Integer array that stores the
data
  int capacity;             // Capacity of array
  int top;                  // Position of top element
};
```


3.1.2
P3.5

Implement *Bag* Operations

```
Bag::Bag( int bagCapacity):capacity( bagCapacity ) {
   if(capacity < 1) throw "Capacity must be > 0";
   array = new int [ capacity ];
   top = -1;
}

Bag::~Bag(){ delete [] array; }

inline int Bag::Size() const { return top + 1; }

inline bool Bag::IsEmpty() const { return Size() == 0; }

inline int Bag::Element() const {
   if(IsEmpty()) throw "Bag is empty";
   return array [0];  // Always return the first element
}

void Bag::Push(const int x) {
   if(capacity == top+1) ChangeSize1D(array,capacity,2* capacity);
   capacity *= 2;
   array[++top]=x;
}

void Bag::Pop( ) {
   if(IsEmpty()) throw "Bag is empty, cannot delete";
   int deletePos = top / 2; // Always delete the middle element
   copy (array+deletePos+1, array+top+1, array+deletePos);
   top--;
}
```

3.1.2 P3.6

Define Class Template *Bag*

```
template<class T>
class Bag
{
public:
  Bag(int bagCapacity = 10);  // Constructor
  ~Bag();                     // Destructor

  int Size() const;           // Return the number of elements
  bool IsEmpty() const;       // Check if bag is empty
  T& Element() const;         // Return an element in the bag

  void Push(const T&);        // Insert an element into the bag
  void Pop()                  // Delete an element from the bag

private:
  T *array;                   // Data array
  int capacity;               // Capacity of array
  int top;                    // Position of top element
};
```


3.1.2 P3.7

Implementation of Some *Bag* Operations

```
template<class T>
Bag<T>::Bag( int bagCapacity):capacity( bagCapacity ) {
  if(capacity < 1) throw "Capacity must be > 0";
  array = new T [ capacity ];
  top = -1;
}

template<class T>
void Bag<T>::Push(const T& x) {
  if(capacity == top+1) ChangeSize1D(array,capacity,2* capacity);
  capacity *= 2;
  array[++top]=x;
}

template<class T>
void Bag<T>::Pop() {
  if(IsEmpty()) throw "Bag is empty, cannot delete";
  int deletePos = top/2; // Always delete the middle emelent
  copy (array+deletePos+1, array+top+1, array+deletePos);
  array[top--].~T();
}
```


3.1.2

An Example Using Template

```
int main() {
    // T is instantiated as int
    Bag<int> Bag1;
    Bag1.Push(10);
    …

    // T is instantiated as MyData
    Bag<MyData> Bag2;
    Bag2.Push(MyData());
    …
}
```