Part1: Basic knowledge of C++
1. Templates

Templates are a feature of the C++ programming language that allow functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.
Function Templates

• The format for declaring function templates with type parameters is

  template <class identifier> function_declaration;

• template <typename identifier> function_declaration;

• template <typename identifier> function_declaration;
template <typename T>

T const& Max (T const& a, T const& b)
{
    return a < b ? b:a;
}
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    int i = 39;
    int j = 20;
    cout << "Max(i, j): " << Max(i, j) << endl;
    double f1 = 13.5;
    double f2 = 20.7;
    cout << "Max(f1, f2): " << Max(f1, f2) << endl;
    string s1 = "Hello";
    string s2 = "World";
    cout << "Max(s1, s2): " << Max(s1, s2) << endl;

    return 0;
}
If we compile and run above code, this would produce the following result:

- Max(i, j): 39
- Max(f1, f2): 20.7
- Max(s1, s2): World
Class Templates

- The format for declaring class templates with type parameters is

```cpp
template <typename identifier>

class class_name{
    ...
    ...
}
```
template <class T>

class mycontainer {
    T element;

public:
    mycontainer (T arg) {element=arg;}
    T increase () {return ++element;}
};
2. Operator Overloading

- In C++ the overloading principle applies not only to functions, but to operators too. That is, operators can be extended to work not just with built-in types but also classes.
/** Pre-increment operator.

* Update this BSTIterator to point to the inorder successor of the current
* BSTNode, and return a reference to the updated BSTIterator.

*/

BSTIterator<Data>& operator++() {
    curr = curr->successor();
    return *this;
}
main() {
    int i = 10, j = 20;
    swapThemByVal(i, j);
    cout << i << " " << j << endl;
    swapThemByRef(i, j);
    cout << i << " " << j << endl;
}

void swapThemByVal(int num1, int num2) {
    int temp = num1;
    num1 = num2;
    num2 = temp;
}

void swapThemByRef(int& num1, int& num2) {
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
Part2:
PA1
Inorder successor

- In Binary Tree, Inorder successor of a node is the next node in Inorder traversal of the Binary Tree.
Tree traversal

1) Pre-order
   • Visit the root.
   • Traverse the left subtree.
   • Traverse the right subtree.

2) In-order (symmetric)
   • Traverse the left subtree.
   • Visit root.
   • Traverse the right subtree.

3) Post-order
   • Traverse the left subtree.
   • Traverse the right subtree.
   • Visit the root.
1) If X has a right child, do this approach
2) If X has no right child and has no parent

3) If X has no right child and is the left child of its parent
4) If X has no right child and is the right child of its parent, do this approach
Actually, 2) and 3) are all in 4).