Session 5

White-Box Testing (3)

chengbaolei@suda.edu.cn

Objectives

- Review of Session 3-4
  - Basic Concepts of White-Box Testing
  - Logic Coverage
  - Control Flow Graph
  - Basis Path Testing
  - Loop Testing
  - Data Flow Testing

3.1 Basic Concepts

- White-box testing must follow several principles:
  - All independent path in a module must be implemented at least once. (Basis path testing)
  - All logic values require two test cases: true and false. (Logic coverage)
  - Inspection procedures of internal data structure, and ensuring the effectiveness of its structure. (Static Testing + Data Flow Testing)
  - Run all cycles within operational range. (Loop testing)

3.2 Logic Coverage

- Logic coverage
  - Statement coverage
  - Decision coverage
  - Condition coverage
  - Condition/decision coverage
  - Condition combination coverage
  - Path coverage
  - Complete coverage

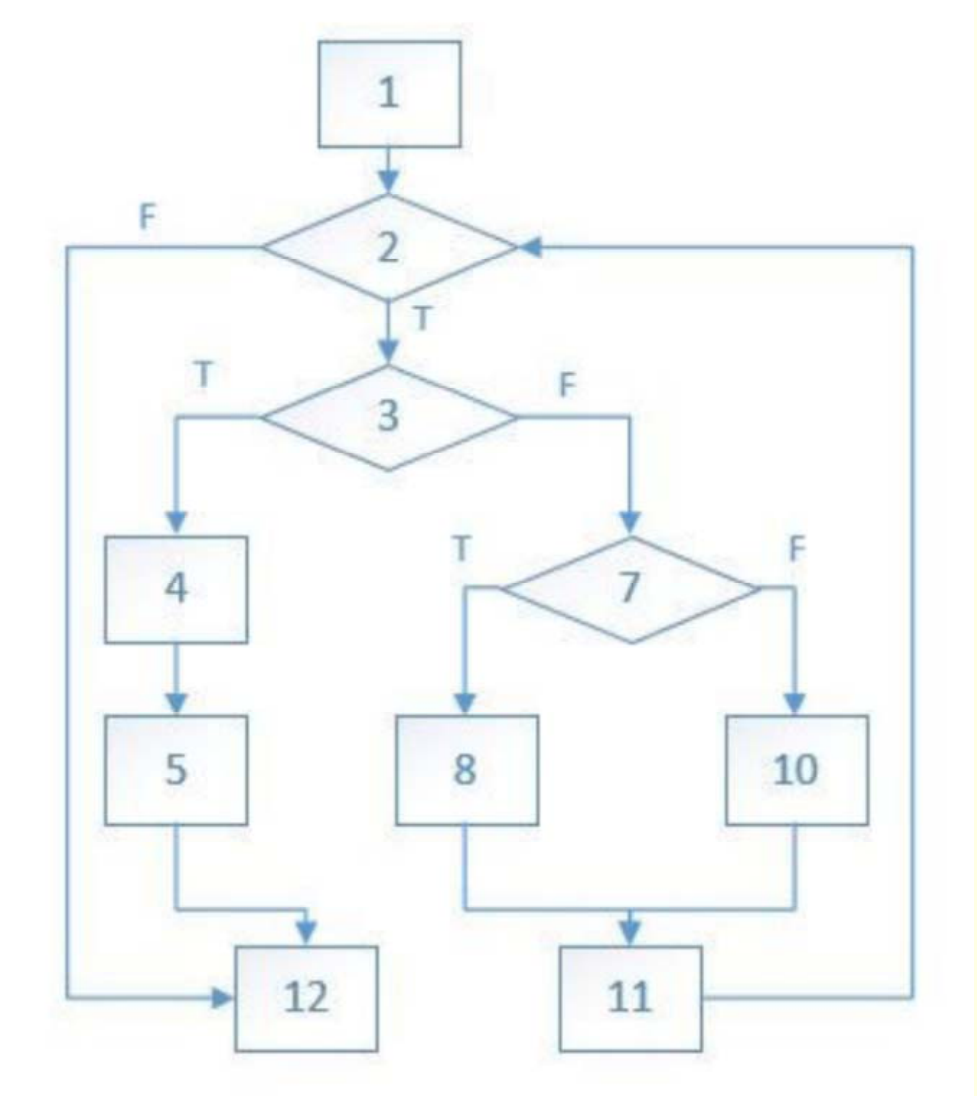
3.3 Control Flow Graph

* Concept
  * During procedure design , in order to more prominent the control flow structure, the procedure can be simplified, the simplified graph is called control flow graph. (vs. program flow graph)
  * On a flow graph:
    * Arrows called *edges* represent flow of control.
    * Circles called *nodes* represent one or more actions.
    * Areas bounded by edges and nodes called *regions*.
    * A *predicate node* is a node containing a condition.

3.4 Basis Path Testing

* How to design test cases for basis path testing
  * Using the design or code, draw the corresponding control flow graph.
  * Determine the cyclomatic complexity of the flow graph.
  * Determine a basis set of independent paths.
  * Prepare test cases that will force execution of each path in the basis set.

```
   int Test(int i_count, int i_flag){
1     int i_temp=0;
2     while (i_count>0){
3           if (0 == i_flag){
4                 i_temp=i_count+100;
5                 break;
            }
6           else{
7                 If (1==i_flag){
8                       i_temp=i_temp+10;
                  }
9                 else{
10                      i_temp=i_temp+20;
                  }
            }
11          i_count – ;
      }
12    return i_temp;
   }
```

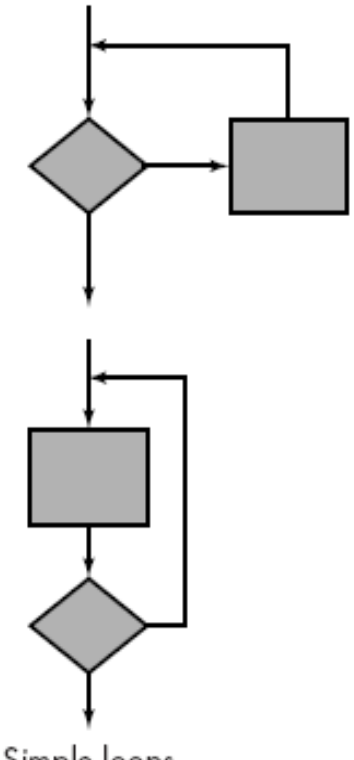


3.5 Loop Testing

- Simple Loops
- Nested Loops
- Concatenated Loops
- Unstructured Loops

3.5 Loop Testing – Simple Loops

* The following set of tests can be applied to simple loops, where n is the maximum number of allowable passes through the loop.
  * Skip the loop entirely.
  * Only one pass through the loop.
  * Two passes through the loop.
  * m passes through the loop where m < n.
  * n - 1, n, n + 1 passes through the loop



Simple loops

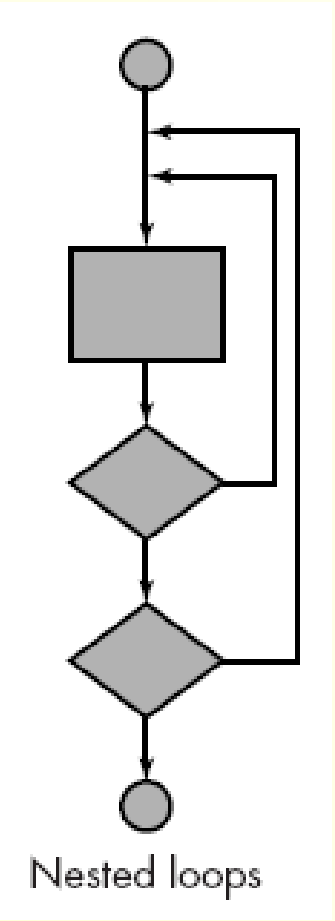
Example

```
int  iSum(int j)
{
    int i=1, s=0, a=100;
    while (i<=j && i<=a)
    {
          s=s+i;
            i=i+1;
    }
    return s;
}
```

Test Case 1: j=0 实际循环
Test Case 2: j=1 实际循环
Test Case 3: j=2 实际循环
Test Case 4: j=50 实际循环
Test Case 5: j=99 实际循环
Test Case 6: j=100 实际循环
Test Case 7: j=101 实际循环

3.5 Loop Testing – Nested Loops

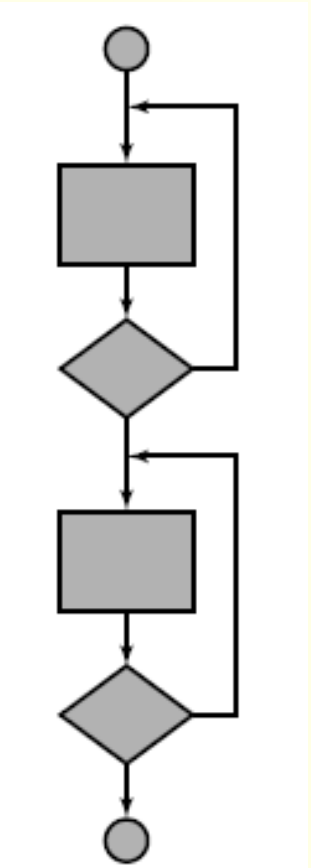
- Start at the innermost loop. Set all other loops to minimum values.
- Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values.
- Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to "typical" values.
- Continue until all loops have been tested.



Nested loops

3.5 Loop Testing – Concatenated Loops

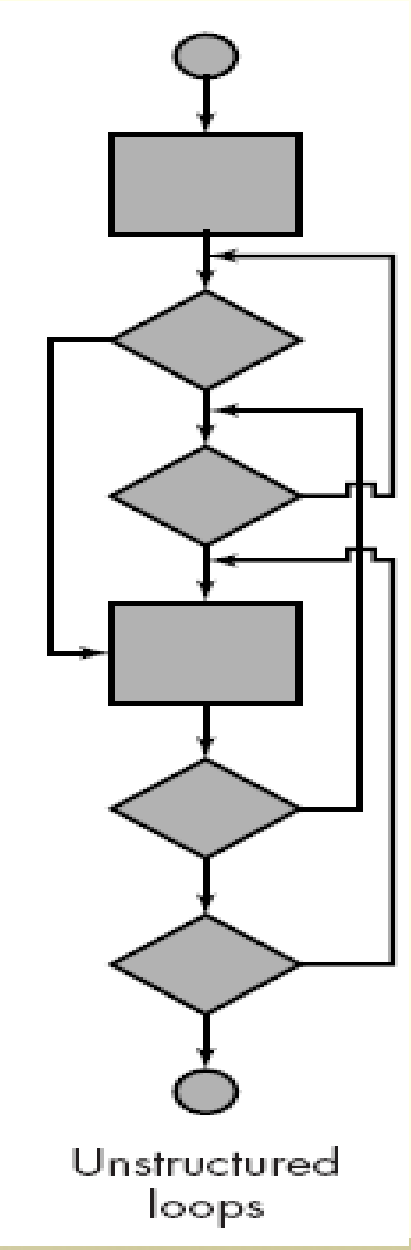
- If each of the loops is independent of the other:
  - Concatenated loops can be tested using the approach defined for simple loops,
- Else
  - the approach applied to nested loops is recommended.



Concatenated loops

3.5 Loop Testing – Unstructured Loops

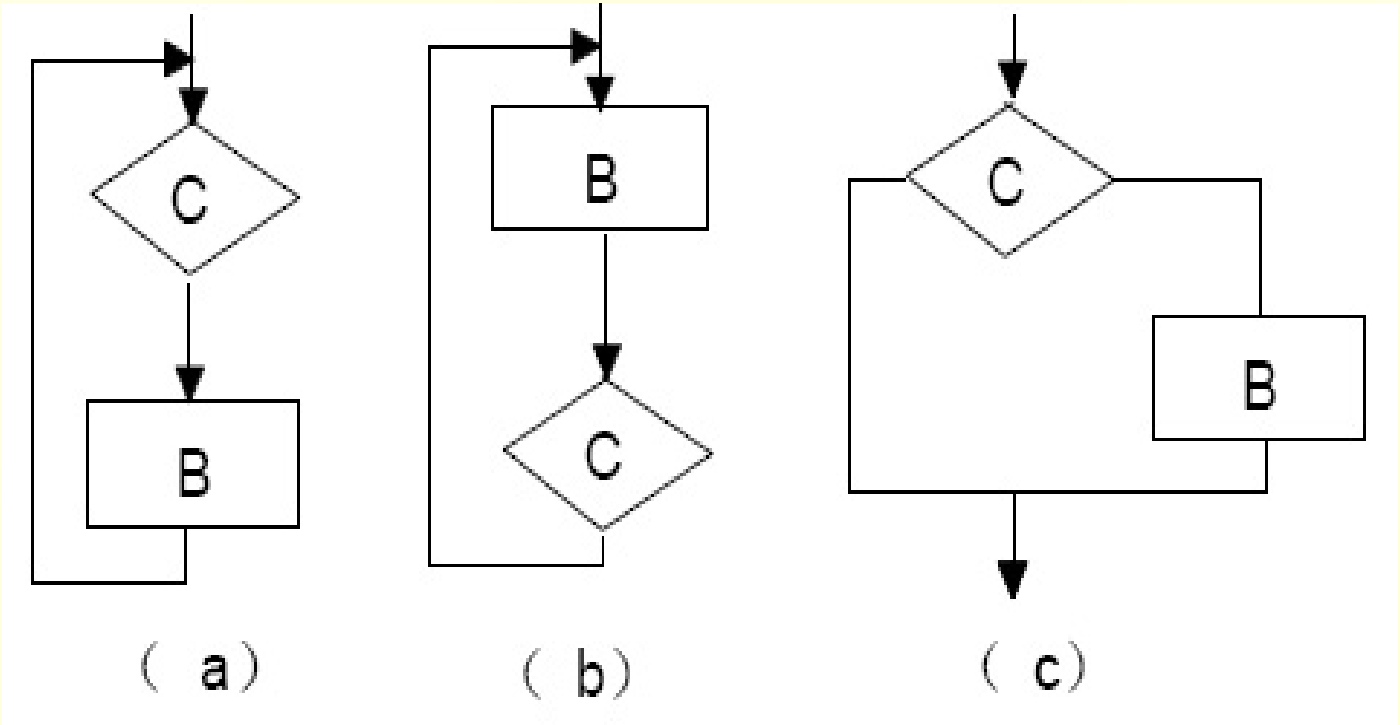
- Suggestion: whenever possible, this kind of loops should be redesigned.
- avoid to use "go to" control construct.



3.5 Loop Testing

- The easy way to test loops is using the method Z path coverage.
- Z path coverage method is a program in Loop structure will be simplified into IF structure test method.
- Simplify cycle test only once, or zero times. In this case, the effect of loop structure and IF branches are same, namely the loop body or execution, or skip.

Example



3.5 Loop Testing

* Loop Testing & Basis Path Testing

(3) for(int i=1; i<arraty_size; i++)

3.5 Loop Testing

Insertion Sort

```
(1)  void insertionSort(int numbers[], int array_size)( {
(2)      int i, j, index;
(3)  for (i=1; i < array_size; i++)      {
(4)      index = numbers[i];
(5)      j = i;
(6)      while ((j > 0) && (numbers[j-1] > index))  {
(7)      numbers[j] = numbers[j-1];
(8)      j = j - 1;
(9)      }
(10)     numbers[j] = index;
(11)  }
(12) }
```

3.1 3.2 3.3

6.1 6.2

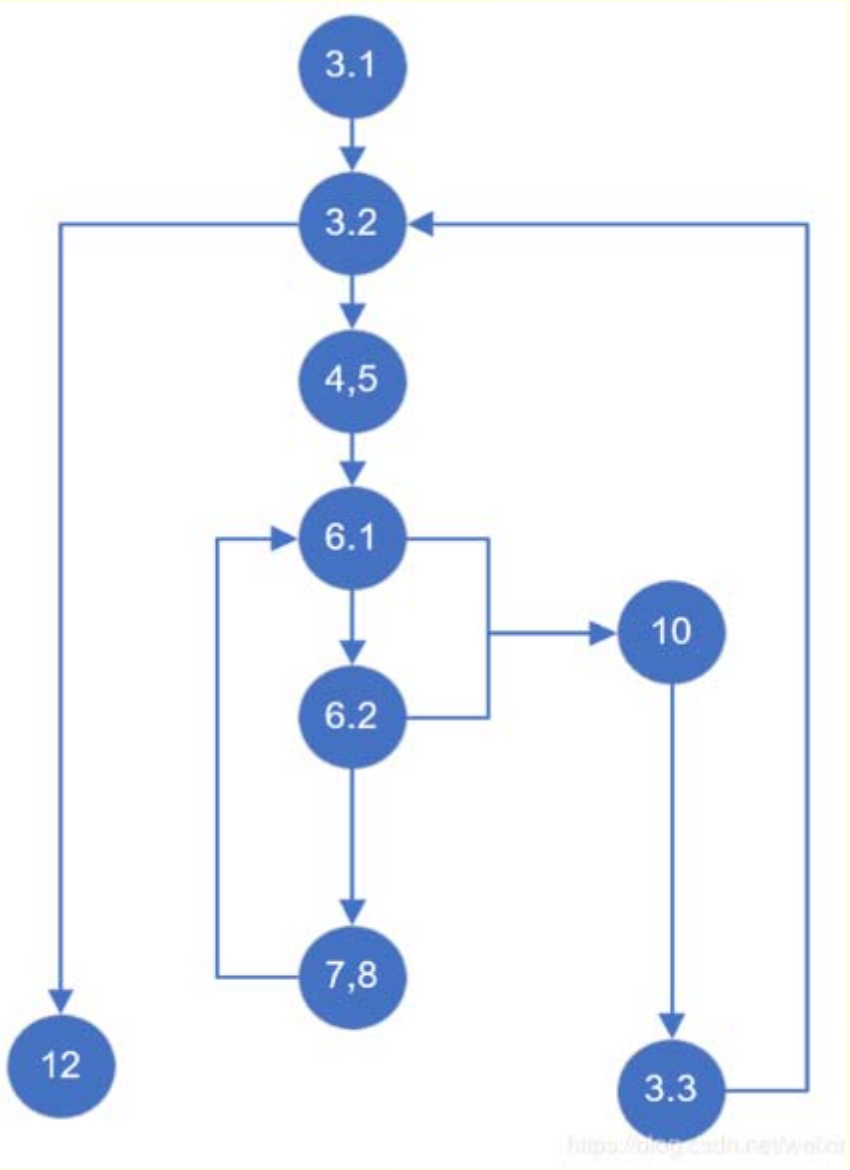
Update CFG

3.5 Loop Testing

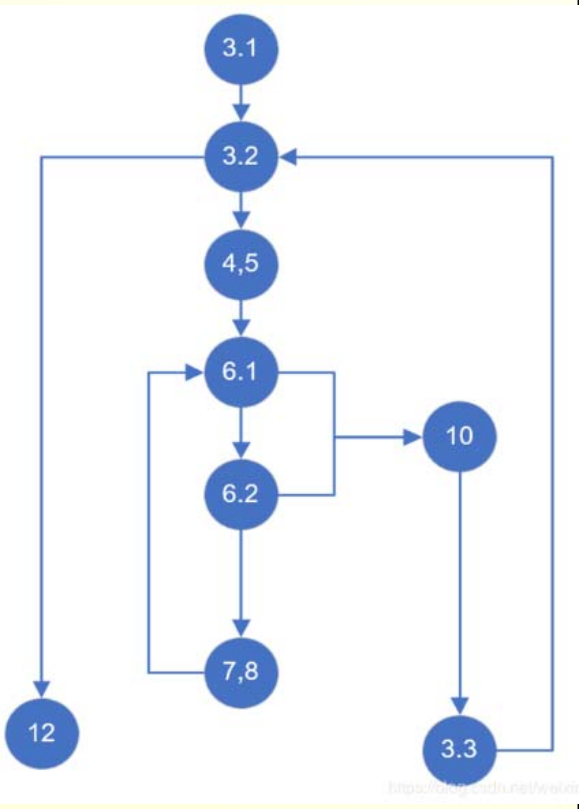
Insertion Sort

```
(1)  void insertionSort(int numbers[], int array_size)( {
(2)      int i, j, index;
(3)  for (i=1; i < array_size; i++)      {
(4)      index = numbers[i];
(5)      j = i;
(6)      while ((j > 0) && (numbers[j-1] > index))  {
(7)      numbers[j] = numbers[j-1];
(8)      j = j - 1;
(9)      }
(10)     numbers[j] = index;
(11)  }
(12) }
```

3.1 3.2 3.3 6.1 6.2



3.5 Loop Testing



Insertion Sort

```
(1)  void insertionSort(int numbers[], int array_size)( {
(2)      int i, j, index;
(3)  for (i=1; i < array_size; i++)      {
(4)      index = numbers[i];
(5)      j = i;
(6)      while ((j > 0) && (numbers[j-1] > index))  {
(7)      numbers[j] = numbers[j-1];
(8)      j = j - 1;
(9)      }
(10)     numbers[j] = index;
(11)   }
(12) }
```

3.1　3.2　3.3　6.1　6.2

路径1：3.1–3.2–12
路径2：3.1–3.2–4,5–6.1–10–3.3–3.2–12
路径3：3.1–3.2–4,5–6.1–6.2–10–3.3–3.2–12
路径4：3.1–3.2–4,5–6.1–6.2–7,8–6.1–10–3.3–3.2–12

- **程序中6.1在未进入while循环自减时j=1恒成立，故不存在任何数据能使程序执行路径2.**

- In session 3-5, you have learned
  - **Basic Concepts of White box testing**
  - **Logic Coverage**
  - **Control Flow Graph**
  - **Basis Path Testing**
  - **Loop Testing**
  - **Data Flow Testing**