CSCU9Y4 **Spring 2018**

Programming Language Paradigms

Tutorial 18/4

Reading for this tutorial: Chapter 4 (esp 4.3-4.5), chapter 5 (esp 5.5) and the first part of chapter 6 (esp 6.2-6.3) of Introduction to Programming Languages (Bansal). Chapter 3 (mostly 3.8), Chapter 6 (mostly 6.2 and 6.4) and Chapter 8 (mostly 8.2) of Comparative Programming Languages, but reading more of these chapters will improve your understanding.

Alternatively, Chapter 3 (mostly 3.5 and 3.6), 5, and 6.3 of Watt, Programming Language Design Concepts, or Chapters 6.9, 9.5 and 12 of Sebesta, Concepts of Programming Languages. Or google "pointer variables", "reference variables", "memory allocation of variables", "garbage in programming languages", "dangling pointer in programming languages", "parameter passing mechanisms" (and the specific mechanisms), "inheritance method overloading", "inheritance method redefinition" etc. Also, Supplementary Material 4.

1. This question is about comparing storage allocation in C++ and Java.

   a) We have a Java class definition `Jobj`

   ```
   class Jobj{
      ...
      public Jobj(String v) {..} // constructor
      // you can assume a getter and setter for the class
      ...
   }
   ```

   and a class `Tester` that contains the `test` method shown below:

   ```
   public Jobj test () {
       Jobj jv1 = new Jobj("Hello");
       return jv1;
   }
   ```

   Describe the storage allocation that occurs when we execute the following code:

   ```
   Jobj mytest1 = new Jobj("Apple");
   mytest1 = test();
   ```

   You should particularly think about the contents of `mytest1` before and after the call to `test`. What is the effect of the declaration of `jv1`? What will exist at run time when we exit from a call of the `test` method? Which entities are on the stack and which are on the heap?

   b) We also have a C++ class definition `Cobj` and class `Tester` that contains the two methods `test1` and `test2`:

```
class Cobj{
  ...
public:
  Cobj(String v) {..} // constructor
  ...
};

Cobj* test1() {
    Cobj *cv1 = new Cobj("Hello");
     ...
     return cv1; }

Cobj* test2() {
    Cobj cv2("Hello");
     ...
     return &cv2; }
```

i. Describe the effect of the declaration of `cv1`. What will exist at run time when we exit from a call of `test1`?

ii. Describe the effect of the declaration of `cv2`. What will exist at run time when we exit from a call of `test2`?

iii. What is a dangling reference? Does `test1` or `test2` produce a dangling reference?

2. Consider the following C++ code where the operator `&` means *address of*. Describe what happens as the statements are executed.

```
int a, *p1, *p2;
a  =  7;
p1 = &a;
p2 = p1;
a = *p2+3;
p1 = p1+1;
```

3. The following code produces a dangling reference:

```
Stype* fun() {
    Stype a[10];
    ...
    return a;
} // fun
```

(e.g. consider the call: `Stype* b = fun();`)

How should the function definition be changed to avoid this problem?

4. Describe the difference between *call by value* and *call by constant-value*. What is the advantage of call by constant-value over call by value? What is the drawback of call by value or call by constant-value when structured variables are being passed?

5. *Call by reference* and *call by value-result* are used to get data in and out of a subprogram. Describe how they differ and the circumstances in which one may be preferred to the other.

6. In Java, we can have multiple methods with the same name. This may be overloading or overriding (redefinition). What is the difference between overloading and overriding (redefinition), and in what circumstances can they occur?