COMP524 Spring 2012 HW-3

Student Name: Duo Zhao PID: 720090360

1 3.1

Indicate the binding time (when the language is designed, when the program is linked, when the program begins execution, etc.) for each of the following decisions in your favorite programming language and implementation. Explain any answers you think are open to interpretation.

1.1 a

The number of built-in functions (math, type queries, etc.)

In C, It's generally defined at the language design time. However, it may be enlarged during the implementation time. e.g. sizeof() is a built-in function in C, which is essentially an unary operator. Most Other functions are not built into language itself, but are provided by standard library, such as ¡stdio.h¿. Some compliers may support these standard library built-in functions.

1.2 b

The variable declaration that corresponds to a particular variable reference (use)

In C, it's at the compile time, since C uses the static scope.

1.3 c

The maximum length allowed for a constant (literal) character string

In C, the actual number is supposed to set at the language implementation time, though at the design time, a variable indicating the maximum length may be introduced.

1.4 d

The address of a particular library routine

In C, it's generally at the link time, especially for the static library. For some dynamic-link library, the bind time may be at the load time or run-time(DLL)

1.5 e

The total amount of space occupied by program code and data

In C, it's at the run time. Since the data may include input, which can only be definitely determined at the run time.

2 3.4

Give three concrete examples drawn from programming languages with which you are familiar in which a variable is live but not in scope.

2.1 1, C++

```
1  #include<iostream>
2  using namespace std;
3  int i = 1;
4  int main(){
5    int i = 2;              //local variable i is defined
6
7    // output: i = 2, the outer i is shadowed, albeit alive
8    cout<<"i = "<<i<<endl;
9
10   //output: i = 1, the scope must be explicit
11   cout<<"::i = "<<::i<<endl;
12
13   return 0;
14 }
```

Same scenario in Java,

2.2 2, Java

```
1  public class{
2          public static int i = 1;
3          public static void main(String[] args){
4                  int i = 2;
5          }
6  }
```

2.3 3, Another example in Java

```
1  class ABC{
2          private int i;
3          public ABC(){
4                  i = 0;
5          }
6          public void setField(int i){
7                  this.i = i;
8          }
9          public static void main(String[] args){
10                 ABC abc = new ABC();
11                 abc.setField(1);
12         }
13 }
```

The *this.i = i* assignment statement. The outer field i is shadowed by the variable passed by the parameter list. a reference this.i must be used to indicate the object field rather than the passed parameter.

3 3.7

3.1 a

The problem arised from Brad's careless use of L = reverse(L), which resulted in memory leak without garbage collection. Brad create a list of new nodes without releasing the the resources that the old list occupies. Therefore, after a loop of operations, the memory will be exhausted.

3.2 b

This time, Brad deleted the old list successfully and release the old list nodes. However, the data that the new and old list share was also released as well. Therefore, the data pointer in the new list node will be a dangling reference, which refers to unkown memory and the result is unpredicatable.