Chapter 9 Summary

-Understand how subprogram call works.
-Understand the design issues of subprograms.
-Understand the mapping of actual and formal parameters.
-Understand parameter pass modes: *pass by value, pass by results, pass by value result, pass by name, pass by reference.*
-Understand pro and cons of using each mode and why each mode can produce different results.
-Know referencing environments and the effect of dynamic scoping/static scoping in subprograms.
- In case of nested subprogram, know the concept of *ad-hoc binding, deep binding, shallow binding.*
-Know the issues of various parameter types: multi-dimensional array, subprogram and the issues of type checking on parameters.

Chapter 10 Summary

-Understand how subprograms are implemented by using the stack.
-Understand the actions on subprogram call and return.
-Understand the concept of ARI, its format, instance, how it is used.
-Understand how the ARI instances are created along the subprogram calls including recursive calls.
-Know the use of static link, dynamic link.
-Understand the concept of static chain, dynamic chain, nested depth, local offset.
-Understand the issues about nonlocal variables upon dynamic scoping and static scoping.
-Know the techniques of static chain and display stack and the pros and cons of them.
-Know the concepts of deep access and shallow access and the pros and cons of them.

Exercise: Chapter 9

1. What are differences between process abstraction and data abstraction?
2. What are actual parameters and formal parameters?
3. What is the role of *prototype* in C?
4. What are the approaches in parameter mapping?
5. What are drawbacks of *keyword mapping?*
6. Consider C++ where default values for parameters are allowed. What is the scheme of parameter mapping that is used?
7. What is the main problem for pass-by-value-result?
8. Discuss pros and cons of pass-by-value-result and pass-by-reference?
9. What are the cases that pass-by-reference gives the results differed from pass-by-name?
10. Explain how pass-by-name can be implemented.
11. What is the problem of passing array as arguments? What about the case of multidimensional array? It is good or bad to specify the format of the array at the arguments? Discuss on this issue.
12. What is the use of subprogram as parameters? How C language deals with this? Why modern language like C++/JAVA do not need the feature anymore?
13. With the subprogram as parameters, what are the design issues?

14. Relate *shallow binding* and *deep binding* to dynamic scoping and static scoping. Explain.

15. What is co-routine? How does it differ from general subprograms?

16. Explain how the co-routine is implemented using stacks.

17. Can you consider exception as subprograms? Explain how exceptions work. Give examples of exceptions of the language you know.

18. Consider the following program.

```
void main() {
      int j=3,i = 1; list[10] = {0,1,2,3,4,5,6,7,8,9};
      xxx(i,list[i], j);
      xxx(j,i,list[j]);
      for (j=0; j < 10; j++)
        printf("\nlist[j]=%d",list[j]);
      printf("\nj=%d i=%d",j,i);
}

void xxx (int a,b,c) {
   int temp =a;
   a = b;
   b = c;
   c = temp;
}
```

From the above code, what are actual parameters and formal parameters?
What is the printout if assume the following parameter passing method?
(a) Pass by value
(b) Pass by reference
(c) Pass by name
(d) Pass by value-result

19. Consider the following program.
What is the printout if assume the following parameter passing method?
(e) Pass by value
(f) Pass by reference
(g) Pass by name
(h) Pass by value-result

```
int k=0;
void main() {
      int i = 1; list[10] = {0,1,2,3,4,5,6,7,8,9};
      xxx(list[i],i);
      for (int j=0; j < 10; j++)
        printf("\nlist[j]=%d",list[j]);
      printf("\nk=%d",k);
}

void xxx (int a,b) {
   a = k+1;
   k = k+2;
   b = b+2;
   b = a+b+1;
}
```

20. Consider the following program. What is the printout if assume the following parameter passing method?
   (a) Pass by reference
   (b) Pass by value-result (copy back from left-to-right)
   (c) Pass by name

```
int f(int x, int y)
{
         g++;
         x = y + g;
         g++;
         return x;
}
int main(int argc, char *argv[])
{
        int q = 0,i;
        int list[] =  {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
        q= f(g,q+1);
        q=f(list[g],g);
        printf("%d %d",q,g);
          for (i=0; i< 21; i++)
               printf("%d ",list[i]);

}
```

21. Consider the following C-like program which allows nested function declaration. Consider when passing function as parameter. What is printed out if we use
(a) shallow-binding
(b) Deep binding
(c) Ad-hoc binding

```
int x=3;
f()
{
        int x=1;
        g()
        {
                int x=0;
                d (&h);

        }
        h()
        {
                   cout >> x;
        }
        d((int *z)())
        {
                int x=2;
                (* z)();
        }

    g();
}
main()
{   int x = 4;
        f();
}
```

Exercise: Chapter 10

1. Why do we need stacks for implementation of parameter passing? Explain.
2. What are actions on calls and returns of subprogram?
3. What is the use of ARI? Is the format and the length fixed? How is it generated?
4. Is it necessary that the compiler knows the format of ARI? Is the order of pushing parameters to the stack important to ARI format? Discuss it.
5. Compare and contrast the techniques of deep access, shallow access Explain in the aspects the time spent in execution and storage space.

6. Compare and contrast the techniques of static chain and display. Explain in the aspects the time spent in execution and storage space.
7. Consider the following program. Draw the stack content assuming the following ARI form.

Local variables
Parameters
Dynamic link
Static link
Return address

```
void P1 ()
{
    printf("Stop ..");
}
int S1 (int n)
{
     int x=0;
     if (n == 0) return x;
     else return n+ S1(n-1);
}
main ()
{
    S1 (4);
    P1();
}
```

8. Consider the following code.

```
proc big1
var x,y;  // locals of big1
{
    proc big2 (n)
    var y,z  // locals
    {
      y=2;
      z=3;
      z= y+z+x+n; // *** 2
      call big3(z,1);
    }
    proc big3(u,v)
    {
       y = y+u+v;
      print(y); // ***** 1
    }


   x=1; y=2;
   call big2(4);
}
```

Assume the calling sequence is big1-> big2-> big3.

(a) What is printed at line 1?

(b) Using the *display*, draw the display stack and ARI stack at line 1?

(c) Using shallow access, draw the stacks of variables with the central tables at line 1?

(d) Calculate nested depth and local offset of variables *y,z,x* at line 2.

9. Consider the code in the following. Answer the following questions.

```
procedure main
  var x,y;
   procedure B();
   var z;
   begin
           z = x+y;
           print(z); //***** 1
   end;
   procedure A (b)
        var y,z;
          procedure C (p,q)
           begin
                       if (p!=q && p > 0 && q > 0)
                     C(p-1,q-1);
                   else B();
               end;
        begin
               y=b;
               z=x+y;
               C(y,z);
        end;

begin
       x=1; y=1;
       A(2);
end.
```

(a) Draw a static structure in the form of tree of the code.

(b) Draw the stack content when first reaches line 1. Assume the ARI format as in the text. What are the nested depth and local offset of the variable *z* ?

(c) How many times C is called?

(d) If static scoping is used, what is printed out?

(e) If dynamic scoping is used, what is printed out?

(f) If dynamic scoping and shallow access is used to implement it, draw the stack containing all variables showing the deepest levels of the calls B.