CmpE 260 - Principles of Programming Languages
Spring 2019
Assignment 2

Bekir Yıldırım - 2014400054

due: 27.05.2019 - 11:59

Answer 1

(Code lines are designated with color red)
(Print statement with color blue)

```
int n = 50; // global                    //1
print_plus_n(int x) {                    //2
   cout << x + n << endl;                //3
}                                        //4
increment_n() {                          //5
   n = n + 1;                            //6
}                                        //7
main() {                                 //8
   int n;                                //9
   n = 10;                               //10
   print_plus_n(n);                      //11
   increment_n();                        //12
   cout << n << endl;                    //13
   print_plus_n(n);                      //14
}                                        //15
```

a) Static Scoping

Static Scoping is a convention used with many programming languages that sets the scope (range of functionality) of a variable so that it may only be called (referenced) from within the block of code in which it is defined. The scope is determined when the code is compiled.

Explanation

a) First Output

Main function is the entry point of this C++ program. An integer variable n is declared in (9) and assigned a value of 10 in (10). print_plus_n(n) definition is calling with a actual parameter of n for 10 and print_plus_n(int x) declaration in (2) with formal parameter, x, is called and value of actual parameter, 10, is passed to formal parameter, x.

x is assigned with 10 but no declaration is found for the variable n, so search continues in the declarations of the subprogram that declared subprogram print_plus_n(int x), which it is called its static parent. And a declaration for n is found in (1). Therefore, x+n becomes number of 60 so cout prints 60.

b) Second Output

increment_n() statement is defined in main (12) and declaration of this function at (5). This function have no actual and formal parameters so n has no declaration there. To find an assigned value, the search continues and a declaration for n is found in (1). value of n becomes 51 in (6) but this value cannot pass the line (13) since in static scoping, if a variable is static variable, it changes just in that subprogram. So cout in (13) prints 10.

c) Third Output

print_plus_n(n) function is called from main like as First Output, just there is difference in value of n. Because, (23) sets global value of n to 51, moreover third output is 10+51 = 61 and (3) prints 61

Outputs are given below respectively:
60
10
61

b) Dynamic scoping

Dynamic Scoping does not care how the code is written, but instead how it executes. Each time a new function is executed, a new scope is pushed onto the stack. This scope is typically stored with the function's call stack. When a variable is referenced in the function, the scope in each call stack is checked to see if it provides the value.

Explanation

a) First Output

Main function is the entry point of this C++ program. An integer variable n is declared in (9) and assigned a value of 10 in (10). print_plus_n(n) definition is calling with a actual parameter of n for 10 and print_plus_n(int x) declaration in (2) with formal parameter, x, is called and value of actual parameter, 10, is passed to formal parameter, x. In dynamic programming, global n becomes 10 because of assigned value of n in (10). x is assigned with 10 but no declaration is found for the variable n, so search continues in the declarations of the subprogram that declared subprogram print_plus_n(int x). And a declaration for n is found in (1). Therefore, x+n becomes number of 20 so cout prints 20

b) Second Output

increment_n() statement is defined in main (12) and declaration of this function at (5). This function have no actual and formal parameters so n has no declaration there. To find an assigned value, the search continues and a declaration for n is found with 10 in

(1). value of n becomes 11 in (6) and this value can pass the line (13) since in dynamic scoping, if a variable is defined and assigned at any time, it changes in everywhere. So cout in (13) prints (11).

c) Third Output

print_plus_n(n) function is called from main like as First Output, just there is difference in value of n. Because, (23) sets global value of n to 11, moreover third output is 11+11 = 22 and (3) prints 22

Outputs are given below respectively:
20
11
22

Answer 2

```
void function1(int a, int b, int c) {            //1
   int d = -1;                                    //2
   while(a > 0) {                                 //3
      c = c / b;                                  //4
      a = a + d;                                  //5
    }                                             //6
}                                                 //7
void function2(int a, int b) {                    //8
   int c = b;                                     //9
   b = a;                                         //10
   a = c;                                         //11
}                                                 //12
int main() {                                      //13
   int x = 2;                                     //14
   int y = 10;                                    //15
   int z = 1500;                                  //16
   cout << x << "," << y << "," << z << endl;     //17
   function1(x,y,z);                              //18
   cout << x << "," << y << "," << z << endl;     //19
   function2(x,z)                                 //20
   cout << x << "," << y << "," << z << endl;     //21
}                                                 //22
```

a) Pass-by-value

Main function is the entry point of this C++ program and x,y,z variables are initialized by 2,10 and 1500 respectively. cout at (17) prints 2,10,1500 (17).
Later, function1(x,y,z) definition (18) is calling its declaration in (1) and x,y,z actual parameters which have values 2,10,1500 respectively are sent to formal parameters a,b and c

respectively. But in function1, not see any statement about ability to change local variables in main function. So, (19) prints 2,10,1500.
At last, function2(x,z) is similar as function1 in other words, function2 has not also any statement about ability to change local variables in main function. So, (21) prints 2,10,1500.

Outputs are given below respectively:
2,10,1500
2,10,1500
2,10,1500

b) Pass-by-reference

Main function is the entry point of this C++ program and x,y,z variables are initialized by 2,10 and 1500 respectively. cout at (17) prints 2,10,1500 (17).
Later, function1(x,y,z) definition (18) is calling its declaration in (1) and x,y,z actual parameters which have values 2,10,1500 respectively are sent to formal parameters a,b and c respectively. Local variable d is initialized with value '-1'. In pass by reference, any change to formal parameters directly affects x,y and z. In the while loop, until a is 0, it will continue subtracting 1 from a and dividing c by b. loop will turn 2 times. When the loop exits, a, b and c take values 0, 10 and 15 respectively, so by the pass-by-reference, x, y and z values have changed. cout prints 0,10,15 (19)
At last, function2(x,z) is called with values actual parameters x and z, (0,15), and a and b take them. In the function2 body, a local variable c is initialized with value 15 from b. b takes value 0 from a, finally a takes value 15 from c. So, the last values of x,y and z are 15, 10 and 0 respectively. cout prints 15,10,0 (21)

Outputs are given below respectively:
2,10,1500
0,10,15
15,10,0

c) Pass-by-name

Main function is the entry point of this C++ program and x,y,z variables are initialized by 2,10 and 1500 respectively. cout at (17) prints 2,10,1500 (17).
Later, function1(x,y,z) definition (18) is calling its declaration in (1) and x,y,z actual parameters which have values 2,10,1500 respectively are sent to formal parameters a,b and c respectively. Local variable d is initialized with value '-1'. In pass by name, any change to formal parameters directly affects x,y and z since pass-by-name in an inout mode parameter transmission method that does not correspond to a single implementation. In the while loop, until a is 0, it will continue subtracting 1 from a and dividing c by b. loop will turn 2 times. When the loop exits, a, b and c take values 0, 10 and 15 respectively, so by the pass-by-name, x, y and z values have changed. cout prints 0,10,15 (19)
At last, function2(x,z) is called with values actual parameters x and z, (0,15), and a and b take them. In the function2 body, a local variable c is initialized with value 15 from b. b takes value 0 from a, finally a takes value 15 from c. So, the last values of x,y and z are 15, 10 and 0 respectively. cout prints 15,10,0 (21)

Outputs are given below respectively:
2,10,1500
0,10,15
15,10,0