BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI, PILANI CAMPUS
FIRST SEMESTER 2017 – 2018
PRINCIPLES OF PROGRAMMING LANGUAGES (CS F301)
MIDSEM EXAMINATION

Date: 11.10.2017 **Weightage: 30 % (60 M)**
Duration: 90 min. **Type: Closed Book**
Note: Answer all parts of the question together.
Answers must be brief. **Total no. of pages: 4**

Q1. Assume that you have been tasked with designing a new programming language for first time programmers of class 6th students to teach fundamental physics and concept of standard units. Students will be expected to write code to evaluate different physical formulae like that of speed, momentum, force, etc. All calculations will be done on real values where each unit of measure will be represented by its type. A type can either be a *basic type* corresponding to seven basic SI units viz. meter, kilogram, second, ampere, kelvin, mole, and candela, or it can be a *derived type* formed by mathematically multiplying or dividing values of basic types. A type error should be thrown whenever dimensional (unit) analysis fails on a formula assignment i.e. assigning value of meter/sec to variable of type kelvin should not be allowed. Additionally, the language supports "set" as homogenous collection of values. Binary mathematical operators (+, -, *, /) have been overloaded to work on sets of compatible types. Example, setA = setB + setC; means operator + is applied on each ordered pair from (setB, setC) to create elements of setA, and so on.

For the given scenario, among each of the following parts, choose one option and justify your choice. Marks will be awarded only for correct justification for the correct option. There will be no marks for only writing the correct option.
[2 + 2 +2 + 2 + 2 = 10 Marks]

Part	Choice to be made for the given language	
a	Concept	Translation System
	Options	Compiler Vs. Interpreter Vs. Hybrid
b	Concept	Type Declaration
	Options	Explicit Vs. Implicit
c	Concept	Type Safety
	Options	Weakly typed Vs. Strongly typed
d	Concept	Type analysis for sets
	Options	Considering sets of same type but of different size as compatible Vs. non-compatible
e	Concept	Type of arguments allowed (assuming inheritance is supported)
	Options	Subtype v/s Generic type

Q2. **[5 + 5 = 10 marks]**

a) Given the following code snippet in C++, identify whether the assignment of values to variables a, b and c at the given locations will be allowed or compilation error will be generated. Write "Allowed / Compilation error" for each variable at locations 1-5:

```
class base
{
   public:
      int a;
   protected:
      int b;
   private:
      int c;
};
class derived1:public Base
{
   void do1()       //Location 1
   { a = 10;  b = 20; c = 30;  }
};
class derived2:private Base
{
   void do2()       //Location 2
   { base :: a = 10;  b = 20;  c = 30; }
};
```

```
class derived3:public derived2
{
   void do3()          //Location 3
   { a = 10;  b = 20;  c = 30;  }
};

int main()
{
   derived1 obj1;       //Location 4
   obj1.a=10;
   obj1.b=20;
   obj1.c=30;
   derived3 obj2;       // Location 5
   obj2.a = 10;
   obj2.b = 20;
   obj2.c = 30;

}
```

b) Given the following C++ code, draw the class instance record for the class D.

```
class A {
      public:
            int a;   int b;
            virtual void func1 () { …}
            virtual void func2() {…}
            void func3 () {…}
            virtual void func4 () {…}
};
class B: public A {
      public:
            int c;
            virtual void func1 () {…}
            virtual void func6 () {…}
};
```

```
class C {
      public:
            int d;
            virtual void func1 () {…}
            virtual void func5 () {…}
};
class D:  public B,  public C {
      public:
            int e, f;
            virtual void func1 () {…}
            virtual void func4()  {…}
};
```

Q3. Consider the following program written in C like syntax: **[5 + 5 = 10 Marks]**

```
main()
{
  float x = 10;
  float y = 20;
  float z = 5;
    function1( )
    {
        float x, y;
        x = 20;
        y = x - 5;
        z = z - 2;
        function2();
        printf (“%f %f %f\n”, x, y, z);
    }
   function2( )
   {  float y;
      y = 10;
      x = x + 5;
      printf("%f %f %f \n", x,y,z);
        function3 (float z)
          {
            printf (“%f %f %f”\n”, x, y, z);
            if (z==1) then break;
            function3 (z-1);
          }
        function3(2);
        printf(“%f %f %f\n”, x, y, z);
   }
function1();
}
```

a) Draw the snapshots of the step by step growth and shrinking of the run-time call stack using activation records appropriately along with the values of the variables in their locations assuming that the language supports **static scoping.** Draw the static and dynamic links on left and right side of the call stack respectively. Assume that the stack grows upwards.
b) What is the output of the above program in the following cases?
   i. Static scoping
   ii. Dynamic scoping.

Q4. Consider the following program written in C like syntax: **[2 + 2 + 3 + 3 = 10 marks]**

```
function test(int x, int y, int z)
{
x = y+z;
y = z+1;
printf("%d %d %d\n", x, y, z);
}
main()
{
int arr [10] ={1,9,2,7,5,6,4,8,3,10};
int  i = 2;
int  j = 3;
test(arr[i], i, i + j);
for(int a=0;a<10;a++)
{
printf("%d", arr[a]);
}
printf("\n %d %d",i ,j);
}
```

What values will be printed by the program for each of the following cases: (suppose array index start with 0):

a) All parameters are passed by value.
b) Pass 'arr[i]' and 'i' by reference and 'i +j' by value.
c) Pass 'arr[i]' and 'i' by result (assuming that the values are copied back in right-to-left order and binding is done at return time) and 'i+j' by value.
   Additionally assume that variable 'x' and 'y' are initialized to '2', '3' respectively.
d) All parameters are passed by name.

Q5. **[7 + 3 = 10 marks]**

a) Write the BNF and EBNF grammar for the 'switch case' statement according to the rules as mentioned below. An example 'switch case' statement has also been shown for clarity**:**

1. A switch statement begins with switch keyword.
2. The switch keyword is followed by parenthesis () within which there is a condition variable of integer type whose range is 1-9.
3. The case statements are enclosed within curly braces {}.
4. A case statement begins with the keyword 'case' followed by integer constant ranging from 1-9 followed by a colon. This is the value to which the condition variable in switch statement is compared with, for selecting a particular case.
5. Each case contains one or more statements.
6. A statement is an assignment statement where the variables are identifiers between 'a-z' and expression can be addition or subtraction of two variables or the variable itself, followed by semicolon.
7. Each case ends with a break statement which is followed by semicolon.
8. There can be one or more case statements within a switch statement.
9. A switch statement can have an optional default case, which must appear at the end. The syntax of the default case is same as for the case statement except that the default case does not have integer constant for comparison and no break is needed in the default case.

Example:

```
switch (a)
  {
        case 1 :  z = x; x = y + z; break;
        case 2:  z = x - y; break;
        default : z = x;
  }
```

b) Given a grammar G with start symbol "S" and non-terminals P, Q, R, S, T, check whether the given grammar is ambiguous or not by generating the parse tree for the string: "aabbccdd".

S → PQ | R
P → aPb | ab
Q → cQd | cd
R → aRd | aTd
T → bTc | bc

Q6. Consider following C code snippet and answer the questions that follow: **[4 + 6 = 10 Marks]**

```
struct _node{struct _node *left; int data; struct _node *right;};
tyepdef struct _node node;
int main() {
        node *m;
        m = (node*) malloc (sizeof(node)); m->data = 10;
        m-> right = (node*) malloc(sizeof(node));
        m -> right ->data = 20;
        m -> left = (node*) malloc (sizeof(node));
        m -> left -> data = 30;
        m -> left -> left = (node*) malloc (sizeof(node));
        m -> left -> left -> data = 40;
        m -> right -> right = m -> left;
        m -> left -> left -> right = m -> right;
        m -> left -> left -> left = NULL;
        m -> left = (node*) malloc (sizeof(node));
        m -> left -> data = 50;
Location P (for part a):
        m -> right = NULL;
        m -> left = m -> right;
Location Q (for part a):
Location A (for part b): Before starting garbage collection
Location B1 (for part b): At the end of mark phase of Mark & Sweep garbage collection
Location B2 (for part b): At the end of sweep phase of Mark & Sweep garbage collection
        [ N lines of code, where N >= 0 ]
}
```

a) Assume that only Reference Counting Garbage Collection Mechanism is used for the above code. Draw memory diagrams for locations P and Q. Follow the instructions to draw a memory diagram as explained below.

b) Assume that only Mark and Sweep Garbage Collection mechanism is used for the above code. The status of the Mark bit could be either Garbage (G) or Live (L). Follow the instructions to draw a memory diagram as explained below.

Memory Diagrams: Draw separate memory diagrams for Locations (P & Q for part a) and (A, B1, and B2 for part b) clearly showing all pointers (left, right, etc.), status of reference counter/mark bits, and dynamically allocated memory blocks, if any. In diagrams of Q and B2, clearly show the pointers of "free list", if any, or show them as NULL. Assume that "free list" represented by an array of 5 pointers (i.e. 1x5 table, where each pointer can point to a currently "free/available" memory block.