Cognome e Nome:_____ Matr.:_____

**Linguaggi di programmazione – A**
**27 giugno 2018**

Esercizio 1.A – Rispondere, in maniera concisa, alle seguenti domande (12 punti)

1. In cosa l'aritmetica dei puntatori differisce dall'aritmetica standard? Si mostri un esempio in C in cui l'aritmetica dei puntatori differisce dall'aritmetica standard?

2. Come si caratterizza l'ereditarietà singola? Come si caratterizza l'ereditarietà multipla?

3. Cos'è una remote procedure call (RPC)?

4. In un linguaggio imperativo, cosa sono i valori memorizzabili?

5. Per quali operazioni i linguaggi regolari sono chiusi?

6. In quale parte della memoria viene memorizzato un array.

7. Quali caratteristiche deve avere un linguaggio di programmazione affinché le funzioni siano considerate oggetti di secondo livello?

8. Cosa si intende per scoping dinamico?

9. Cosa distingue i linguaggi class based dai linguaggi prototype based?

10. Quali sono le caratteristiche del polimorfismo di sottotipo? Quali sono le caratteristiche del polimorfismo parametrico?

11. Quali sono le caratteristiche del passaggio dei parametri per riferimento?

12. A cosa servono le eccezioni? Come accade quando viene generata un'eccezione?

13. Nella gestione dello scoping statico, cosa si intende per display?

14. Cos'è il P-code?

15. Nella memoria, che funzione svolge l'heap?

16. Quali sono le caratteristiche di un strong type system? Quali sono le caratteristiche di un weak type system?

17. Cosa sono i tipi intervallo? Cosa sono i tipi enumerazione?

1. Cognome e Nome:_____ Matr.:_____

**– 27 giugno 2018– A**

## Esercizio 2.A – Grammatiche (4 punti)

1. Si consideri la grammatica:

$$L \rightarrow \epsilon \mid aLbLb \mid bLaLb \mid bLbLa$$

e l'espressione regolare

$$E = (abb|bab|bba)^*$$

- si scrivano tutte le parole di lunghezza minore di 8 generate dalla grammatica $L$,
- si stabilisca se la grammatica è ambigua,
- si descriva il linguaggio generato,
- si stabilisca se è un linguaggio regolare,
- lo si confronti con il linguaggi generato dall'espressione regolare $E$, e si stabilisca se sono uno contenuto nell'altro,
- si abbozzino delle regole YACC-Happy, per la creazione di un programma che riconosco il linguaggio generato dalla grammatica $L$ e dalla stringa costruisco l'albero di derivazione.

Esercizio 3.A – Stack di attivazione (6 punti)

1. Si mostri l'evoluzione dello stack di attivazione e dell'output dei due frammenti di programma seguenti. Si ipotizzi che il linguaggio C-like abbia scoping statico, assegnamento che calcola l-value prima di r-value, valutazione delle espressioni da sinistra a destra e indici vettori inizianti da 0:

```
char x [10] = il_proprio_cognome_nome ;
int i = 1 ;
char magic(val  int j, ref char y)
{
  y = x[j++] = x [++j];
  write (y , j ) ;
  return x[i++];
}
write ( magic (i, x[++i] ) ) ;
write (i) ;
```

```
int x = 5 , y = 7 ;
int F( name int v , valresul int y ) {
  y +=  v;
  write (v, x, y)
  return (v + x);
}
{
   int x = 3;
   int Q( ref int v , val int y ) {
     v = F(v++, y);
     write (v, x++, y);
    return (x + v);
   }
   write (Q(y, F(y++, x));
   write (x, y);
}
write (x, y);
```

1. Cognome e Nome:_____ Matr.:_____

**– 27 giugno 2018– A**

## Esercizio 4.A – Haskell (7 punti)

1. Il grafo di una funzione parziale $f : A \to B$ è definito come l'insieme di coppie $\{(a, b) \mid b = f(a)\}$, ossia l'insieme di coppie in cui i primi elementi costituiscono il dominio della funzione e i secondi elementi definiscono il comportamento della funzione, sui corrispondenti primi elementi.

   Scrivere una funzione Haskell che preso il grafo di una funzione $f$ da $A$ in $B$, rappresentato come lista di coppie, ed una lista $l$ di elementi di tipo $A$, applica la funzione $f$ a tutti gli elementi della lista.

   Si tenga presente che la funzione $f$ può essere parziale (non definita su alcuni elementi), nel caso la lista $l$ contenga un elemento su cui $f$ non è definita, l'elemento viene rimosso dalla lista.

   Scrivere inoltre una funzione Haskell che, dato un lista di coppie, controlli che la lista non contenga due coppie con il primo elemento uguale.

   Si definisca inoltre il tipo Haskell delle funzioni definite e delle funzioni ausiliarie.

Esercizio 5.A – Sistema di assegnazione di tipo (4 punti)

1. Nel sistema di tipi per il linguaggio imperativo presentato nei lucidi, costruire la derivazione di tipo per il comando:

   ```
   { Nat x = 3; dec(Nat y) {y = y-x+1 }; while (x < 0) { dec(x) } }
   ```

   Nota: i primi passi di derivazione, a partire dagli assiomi, se elementari o ripetizioni di derivazioni già fatte, possono essere omessi.

   In alternativa si consideri il seguento comando semplificato:

   ```
   { Nat x = 3; Nat y = x+1; while (x < 0) { x := x - y } }
   ```