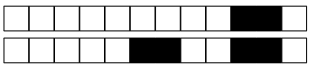

Rispondere alle domande a risposta multipla annerendo la casella corrispondente alla risposta corretta. Ogni domanda ha una ed una sola risposta corretta.

Cognome e Nome: .....

Matricola: .....

Domanda 1 Il concetto di *variabile modificabile*:

- ☐ E' l'unico concetto utilizzabile quando si parla di variabili
- ☐ E' imposto dall'architettura di Von Neumann (variabili non modificabili richiederebbero macchine astratte caratterizzate da memoria a sola lettura)
- ☐ Nessuna delle altre risposte
- ☐ E' tipico del paradigma di programmazione imperativo
- ☐ Permette di evitare il fenomeno dell'*aliasing*

Domanda 2 La memoria gestita staticamente:

- ☐ Nessuna delle altre risposte
- ☐ E' allocata esplicitamente dal programma a tempo di esecuzione, ma una volta allocata è staticamente legata al programma e non può essere liberata fino alla sua terminazione
- ☐ E' una memoria a sola lettura
- ☐ E' allocata dal compilatore prima dell'esecuzione del programma. Le entità allocate staticamente in memoria risiedono in una zona fissa di memoria durante tutta l'esecuzione del programma
- ☐ E' allocata prima dell'esecuzione del programma. Le entità allocate staticamente possono essere deallocate durante l'esecuzione del programma, per liberare memoria

Domanda 3 Dato il frammento di programma (espresso in pseudo-codice) della Figura 1, quanto vale la variabile globale `c` dopo aver eseguito `topolino()`, assumendo scope dinamico?

- ☐ Nessuna delle altre risposte
- ☐ Non è possibile dirlo
- ☐ 6
- ☐ 3
- ☐ 14

Domanda 4 Si consideri lo pseudo-codice di Figura 2. Qual'è il valore di ritorno di `f(1,r(1),1)` se i parametri sono passati *per valore*?

- ☐ Nessuna delle altre risposte
- ☐ Si ha ricorsione infinita
- ☐ 1
- ☐ Non è possibile dirlo senza conoscere il tipo di scope (statico o dinamico) utilizzato
- ☐ 0

```
int a, b, c;

void pippo(void)
{
  int a;

  a = 6;
  b = 5;
}

void pluto(void)
{
  int c;
  int b;

  pippo();
  c = 3;
  a = 4;
}

void topolino(void)
{
  int a;

  a = 1;
  b = 10;
  pluto();

  c = a + b;
}
```

Figure 1: Esempio di pseudocodice

```
int r(int x)
{
  return r(x - 1);
}

int f(int a, int b, int c)
{
  if (c == 1) return a; else return b;
}
```

Figure 2: Esempio di pseudocodice

```
int x, y, z;

void minni(void)
{
  x = 4;
  y = 8;
}

void paperino(void)
{
  int y;

  minni();
  y = 1;
  z = 666;
}

int topolino(void)
{
  int x;

  x = 5;
  y = 15;
  z = x + y;
  paperino();

  return z - y - x;
}
```

Figure 3: Esempio di pseudocodice

Domanda 5 Se gli array sono memorizzati *per righe* ed `int a[100][100]` è un array multi-dimensionale di interi (si assuma che la dimensione di un intero sia 4 byte) con `a[0][0]` che ha indirizzo `0x5000`, qual'è l'indirizzo di `a[5][10]`?:

- ☐ 0x5510
- ☐ Nessuna delle altre risposte
- ☐ 0x51FE
- ☐ 0x53ED
- ☐ 0x57F8

Domanda 6 Dato il frammento di programma (espresso in pseudo-codice) contenuto in Figura 3, qual'è il valore di ritorno di `topolino()`, assumendo scope statico?

- ☐ -3
- ☐ 0
- ☐ 14
- ☐ Non è possibile dirlo
- ☐ Nessuna delle altre risposte

```
int c = 2;

int pippo(int a)
{
  c = c + 2;
  return a * 2;
}

int pluto(void)
{
  return(pippo(c + 1));
}
```

Figure 4: Esempio di pseudocodice

Domanda 7 Si consideri lo pseudo-codice di Figura 4. Qual'è il valore di ritorno di `pluto()` se i parametri sono passati *per riferimento*?

- ☐ Dipende dal tipo di scope (statico o dinamico) utilizzato
- ☐ 10
- ☐ Non è possibile passare `c + 1` per riferimento
- ☐ Nessuna delle altre risposte
- ☐ 6

Domanda 8 β-riducendo $(\lambda x.xy)(\lambda z.zx)(\lambda z.zx)$ si ottiene:

- ☐ $yx(\lambda z.zx)$
- ☐ $(\lambda x.xy)yx$
- ☐ Nessuna delle altre risposte
- ☐ xyz
- ☐ La riduzione non termina

Domanda 9 β-riducendo $(\lambda n.\lambda f.\lambda x.f((nf)x))(\lambda f.\lambda x.f(f(f(fx))))$ si ottiene:

- ☐ La riduzione non termina
- ☐ fx
- ☐ $\lambda f.\lambda x.ffffffx$
- ☐ Nessuna delle altre risposte
- ☐ $\lambda f.\lambda x.f(f(f(fx)))$

Domanda 10 Un *compilatore* da un linguaggio $\mathcal{L}$ ad un linguaggio $\mathcal{L}_O$ è:

- ☐ L'implementazione di una macchina astratta scritta nel linguaggio $\mathcal{L}_O$, che capisce programmi scritti nel linguaggio $\mathcal{L}$
- ☐ Una implementazione di macchine astratte indipendente dalla macchina fisica
- ☐ Un programma che trasforma un programma $P^{\mathcal{L}}$ (espresso nel linguaggio $\mathcal{L}$) in un programma $P^{\mathcal{L}_O}$ (espresso nel linguaggio $\mathcal{L}_O$) tale che per ogni input I si ha $P^{\mathcal{L}}(I) = P^{\mathcal{L}_O}(I)$
- ☐ Nessuna delle altre risposte
- ☐ Un programma scritto nel linguaggio $\mathcal{L}_O$ che riceve come ingresso un programma $P^{\mathcal{L}}$ (espresso nel linguaggio $\mathcal{L}$) ed il suo input I generando lo stesso output che genera $P^{\mathcal{L}}$ con input I

Domanda 11 In presenza di variabili modificabili:

- ☐ Nessuna delle altre risposte
- ☐ Non esistono valori denotabili
- ☐ Esistono un *Ambiente* che associa valori denotabili (fra cui le locazioni di memoria) a nomi ed una *Memoria* che associa locazioni di memoria a valori memorizzabili
- ☐ Il comando di assegnamento non ha effetti collaterali
- ☐ La valutazione del comando di assegnamento restituisce sempre un valore

Domanda 12 L'allocazione dinamica della memoria:

- ☐ Può essere fatta sia dallo *stack* che dallo *heap*
- ☐ Nessuna delle altre risposte
- ☐ E' sempre effettuata solo dal compilatore o dall'interprete
- ☐ Può essere fatta solo dallo *stack*
- ☐ Può essere fatta solo dallo *heap*