סדנת תכנות ב - C++

מועד ב - פתרון הבחינה

מרצה: אורי מוסנזון

27.7.05

הנחיות

- משך הבחינה: שעתיים.
- יש לענות על כל שאלות הבחינה.
- ניתן להשתמש בכל חומר עזר כתוב. אין להשתמש במכשירי חישוב.
- הכתיבה תתבצע כולה על דפי הבחינה.
- שאלות אמריקאיות: יש להקיף בעיגול את האות שבתחילת השורה המתאימה.
- שאלות הדורשות כתיבה: כתבו רק בשורות המיועדות לכתיבה, הן אמורות להספיק.

במקרה של חריגה בשורה או שתיים, אם הסיבה היא רק סגנון כתיבה, התשובה תחשב תקינה.

- יש להקפיד על רישום ברור של מספר תעודת הזהות!

(8%).1

נתון הקוד הבא (מתייחס לשאלות 1-4):

```
#include <iostream>
class A {
  static int _total;
  int _serial;
  A& _mine;
  A* _iKnow;
public:
  A() : _serial(_total),_mine(*this),_iKnow(this) {++_total;}
  A(A& mine, A* iKnow) : _serial(_total),_mine(mine),_iKnow(iKnow) {++_total;}
  void print() {
    std::cout << _serial;
    if(_iKnow != this) {
      _mine.print();
      _iKnow->print();
    }
    std::cout << '|';
  }
};

int A::_total = 0;

int main() {
  A a,a1(a,&a),a2(a,&a1);
  a2.print();
}
```

מה יהיה הפלט?

_____

תשובה:

20|10|0|||

(4%).2

האם בסוף ריצת התוכנית יישאר זיכרון שלא שוחרר (memory leak) ?

תשובה: _____

תשובה: לא

(5%).3

מה יקרה לתוכנית זו, אם נוסיף ל – A את הפונקציה הציבורית הבאה:

```
~A() { delete _iKnow;}
```

_____

_____

_____

תשובה:

הוספת dtor המוחק את האובייקט עליו מצביע אובייקט מסוג A תגרום למחיקה כפולה של זיכרון. האובייקט a לדוגמה ימחק פעם אחת כאובייקט לוקאלי על המחסנית ופעם שניה ע"י האובייקט a1 שמכיר אותו. בעיה דומה תהיה תהייה עם a1. התנהגות התוכנית במקרה זה איננה מוגדרת.

(5%).4

נניח שמחליפים את פונקצית ה – main של הקוד המקורי להיות:

```
int main() {
  A a,a1(a);
  a1.print();
}
```

מה יהיה הפלט אז?_____

תשובה:

00|0||

(12%).5

נתון הקוד הבא (מתייחס לשאלות 7-5):

```
#include <iostream>
struct Node {
  int _data;
  Node *_next;
  Node(int data, Node* next=0) :_data(data),_next(next) {}
};

class List {
   Node *_head;
public:
  List() : _head(0) {}
  void insert(int data) { _head = new Node(data,_head); }
};

int main() {
  int array[] = {7,6,5,4,3,2,1};
```

```
 List l1,l2,l3;
 l2.insert(17);
 for(int i=0; i<7; ++i)
  l1.insert(array[i]);
 l3 = l2 = l1;
}
```

הוסיפו את הקוד הדרוש על מנת לדאוג לשחרור זיכרון ולהשמת רשימות. לאחר ההוספה שלכם, main אמורה להחזיק שלוש רשימות שוות ובלתי תלויות ואז לשחרר את כולן בצורה תקינה.
מספר דגשים:

- יש להשתמש ב – const בכל מקום מתאים.
- יש להימנע משכפול קוד.
- יש להימנע מהתייחסות מיוחדת למקרים פרטיים אותם ניתן לכלול כמקרים טיפוסיים.

תשובה:

```
class List {
…
~List() {clear();}

 void clear() {
  Node *p;
  while(_head) {
   p= _head;
   _head = _head->_next;
   delete p;
  }
 }
…
List& operator =(const List& other) {
  clear();
  Node **my=&_head,*his=other._head;
  while(his) {
   *my = new Node(his->_data);
   my = &((*my)->_next);
   his = his->_next;
  }
  return *this;
 }
```

6. (4%)

לרשימה של השאלה הקודמת הוסיפו את הפונקציות הבאות:

```
class List {
private:
 static int size(Node* p) {
   if(!p)
    return 0;
   return size(p->_next)+1;
  }
…
public:
int size() const {
   return size(_head);
}
```

…

}

בחרו את התשובה הנכונה:

א. הפונקציה שגויה ואיננה מחזירה את מספר האברים ברשימה.

ב. הפונקציה תקינה, ואופטימלית מבחינת יעילות.

ג. פונקצית size המשתמשת בלולאה פשוטה תהיה יעילה יותר מבחינת זמן הריצה אך לא מבחינת דרישת זיכרון מחסנית.

ד. פונקצית size המשתמשת בלולאה פשוטה תהיה יעילה יותר מבחינת זמן הריצה ומבחינת דרישת זיכרון מחסנית.

תשובה: ד.

7. (4%)

לגבי אותה פונקצית size מהשאלה הקודמת, בחרו את התשובה הנכונה:

א. אין שום הגיון בהצהרה על size(int) כסטטית.

ב. אין שום הגיון בהצהרה על size(int) כפרטית.

ג. הצהרה על size(int) כסטטית, עשויה לשפר את היעילות.

ד. יש להוסיף const גם בסוף ההצהרה של size(int).

תשובה: ג.

8. (8%)

נתון הקוד הבא (מתייחס לשאלות 8-12):

```
#include <iostream>
class A {
public:
  virtual void foo() = 0;
};

template <typename T>
class B : public A {
public:
  T _data;
  B(T data) : _data(data) {}
  virtual void foo() {
    std::cout << "[";
    _data.foo();
    std::cout << "]";
  }
};
class C: public A {
public:
  virtual void foo() { std::cout << "I'm C"; }
};
int main() {
  A **array = new A*[3];
  array[0] = new B<C>(C());
  array[1] = new B<B<C> >(*(B<C>*)array[0]);
  array[2] = new B<B<B<C> > >(*(B<B<C> >*)array[1]);
  for(int i=0; i<3; ++i)
    array[i]->foo();
}
```

מה יהיה הפלט?

_____

תשובה:

[I'm C][[I'm C]][[[I'm C]]]

(5%).9

הסבירו מה יקרה אם ישנו את השורה השלישית של main להיות:

```
array[1] = new B<B<C> >(*array[0]);
```

הסבר:_____

_____

תשובה: הטיפוס של [array[0] הוא *A ולכן לאחר השינוי ישלח ל ctor משתנה מטיפוס A. ה - ctor מצפה ל <B<C היורש מ-A. הקומפיילר יודיע שהוא איננו מוכן להמיר <B<C ל - A (או שפשוט אין לו ctor המקבל טיפוס מתאים)

(5%).10

הסבירו מה יקרה אם ישנו את השורה השלישית של main להיות:

```
array[1] = new B<B<C> >((B<C>)(*array[0]));
```

הסבר:_____

_____

תשובה: כאן יש ניסיון להמיר <u>אובייקט</u> מסוג A <u>לאובייקט</u> מסוג <B<C. מכיוון שלא כתבנו המרה כזו (ע"י ctor או ע"י אופרטור מתאים) הקומפיילר יתלונן שהיא איננה אפשרית.

(4%).11

התבוננו בלולאה המופיעה בשורות האחרונות של פונקצית ה - main בקוד המקורי, ובחרו את התשובה הנכונה:

א. הלולאה קוראת שלוש פעמים לפונקציה בשם foo אבל בכל פעם זו פונקציה אחרת.
ב. הלולאה משתמשת במנגנון compile time polymorphism בכדי להחליט לאיזה פונקצית foo לקרוא.
ג. בכל הפעלה של foo מהלולאה, מופעלות בצורה עקיפה, כל הפונקציות בשם foo של היורשים של A.
ד. למעשה קיימת רק פונקצית foo אחת והיא זאת שנקראת.

תשובה: א.

(3%).12

בפונקציה ()foo של <B<T מופיעה השורה הבאה:

```
_data.foo();
```

האם ההחלטה איזה פונקצית foo תקרא ע"י שורה זו, מתבצעת בזמן ריצה או בזמן קומפילציה?

תשובה:_____

תשובה: בזמן קומפילציה

(7%).13

כתבו את המחלקה Barnash כך שניתן יהיה להשתמש בה כך:

```
#include <iostream>
class Barnash {
//   הקוד שלכם
};

int main() {
  Barnash b1(“Uzi”),b2(“Yoda”),b3(“Zalman”);
  (b1 >> (b2 >> (b3 >> std::cout)));
}
```

ולקבל את הפלט:

```
ZalmanYodaUzi
```

תשובה:

```
class Barnash {
  char* _name;
public:
  Barnash(char* name) : _name(name) {}
  std::ostream& operator>>(std::ostream& os) {
    return os << _name;
  }
};
```

14.(8%)

נתון הקוד הבא (מתייחס לשאלות 14-17):

```
#include <iostream>

template <typename T1=int, typename T2=double>
class A {
public:
  T1 _d1;
  T2 _d2;
  A(T1 d1=T1(), T2 d2=T2()): _d1(d1),_d2(d2) {}
};

template <typename T1,typename T2>
std::ostream& operator<<(std::ostream& os, const A<T1,T2>& a) {
  return os << '(' << a._d1 << ',' << a._d2 << ')';
}

int main() {
  A<> a1(1,2);
  A<A<> > a2,a3(a1);
  std::cout << a1 << a2 << a3 << std::endl;
}
```

מה יהיה הפלט?

_____

<u>הערה</u>: לכל טיפוס פרימיטיבי יש מעין default ctor המאתחל את המשתנה לאפס. לדוגמה:

```
typedef int* Ptr;
int main() {
  int i1 = int(); double *i2 = new double(); Ptr i3 = Ptr();
  std::cout << i1 << ' ' << *i2 << ' ' << i3 << '\n';
}
```

ייתן את הפלט: 0 0 0

תשובה:

(1,2)((0,0),0)((1,2),0)

.15 (5%)

כמה פעמים נפרשת פונקצית ה - Template "operator<<(…)" בקוד זה?_____

תשובה: פעמיים.

.16 (7%)

נניח שמוסיפים לקוד הקודם את פונקצית ה - Template הבאה:

```
template <typename T1,typename T2>
std::ostream& operator<<(std::ostream& os, const A<T1*,T2*>& a) {
  return os << '(' << *(a._d1) << ',' << *(a._d2) << ')';
}
```

וכן משנים את פונקצית ה - main, להיות:

```
int main() {
  A<> a1(1,2);
  A<A<> > a3(a1);
  A<A<>*, A<A<>,double>* > a4(&a1,&a3);
  a1._d1 += 2;
  std::cout << a4 << std::endl;
}
```

מה יהיה הפלט? הסבירו את תשובתכם במילים.

תשובה:

((3,2),((1,2),0))

המשתנה a4 מחזיק הצבעות למשתנים a1 ו a3. a3 מחזיק העתק של a1. שינוי של a1 להכיל את 3 במקום 1, משפיע רק על a4 אבל לא על a3. פורמט ההדפסה נישמר תודות לפונקציה ספציפית שהוספה ודואגת לA-ים המחזיקים מצביעים.

הערה: שימו לב שבכל פעם שלקריאה מסוימת מתאימות מספר פונקציות Template, נבחרת זו המקבלת את הטיפוס הספציפי יותר.

.17 (6%)

כעת מוסיפים את פונקצית ה - Template הבאה:

```
template <typename T1, typename T2, typename T3>
void foo(A<T1**,A<T2,T3*> > a) {}
```

את פונקצית ה - main משנים להיות:

```
int main() {
  foo(A<int****,A<int*,double**> >());      //1
  foo(A<A<>**,A<A<>*****,int**> >());  //2
  foo(A<A<int**>,A<float,float*> >());      //3
}
```

עליכם לקבוע לגבי כל שורת קוד ב - main אם היא עוברת קומפילציה ואם כן, איזה טיפוסים יקבלו T1, T2 ו- T3 בהפעלתה.

1._____

2._____

3._____

תשובה:

1. **T1 = int**, T2=int*, T3 = double***
2. **T1 = A<int,double>, T2 =A<int,double>*****, T3 =int***
3. **Compilation error. A<int**> does not match the pattern T1****