בס"ד

<u>מבחן בתכנות מחרפן</u>

לפניכם(כלומר, בקובץ darkness.cpp) הקוד למחלקת אדון האופל...

לכל אדון אופל יש גם מבצר ששייך לו וחיית טרף הסרה למרותו. בנוסף, יש לו מגוון פונקציות משעשעות שהופכות אותו לכל כך מגניב.

שאלה 1:

א.מצאו לפחות 7 מקומות בהם ניתן היה לשנות את הקוד של אדון אופל, חיית טרף ומבצר כדי שיהיה פחות חשוף לשגיאות. אין להצביע על אותו סוג שגיאה יותר מפעם אחת (משתנה אחד שהיה ראוי להיות const סוגר את כל המשתנים שהיו צריכים לקבל const)

ב. הביטו במימוש של useDarkPower.

1.כתבו לפחות 3 דברים שצריך לשנות כדי שיהיה לה פחות פוטנציאל לבאגים

2. האם עתה הפונקציה הזו ממומשת כמו שצריך? אם לא, שפרו אותה. (3 דרכים)

3. איזו מהדרכים היא הטובה ביותר(באופן אינטואיטיבי)? תנו שתי סיבות

שאלה 2:

לפניכם הכרזה של פונקציית אטולסטוי(פונקציות אלו ידועות מאוד. העתיקה ביותר נועדה כדי לכתוב רומנים ברוסית, ומכאן שמה. פונקציות אחרות במשפחה זו נועדו כדי לחלק באפס, להכין טוסטים עם גבינת קשקוול, לייבש ביצות ולשחק עם תינוקות משוממים) הפונקציה הספציפית הזו נכתבה על ידי גבי טייץ, סטודנט מצטיין שמחלק את זמנו בין תכנות, טיגון עופות בעזרת קרינה אלקטרומגנטית(או שזה מישהו אחר בעצם...) ,חרישת שדות וגידול פרות:

```
template<class T>

trash atolstoy_v41(T object)
```

כאשר:

typedef struct trashChecker{ bool isGarbage; unsigned bytesOK;} trash;

הפונקציה הזו מקבלת משתנה (יכול להיות פרימיטיבי, פוינטר, אובייקט, [container!!]) ומחזירה האם הוא מזובל, ואם כן, אז כמה בתים בו הם בכל זאת אינם זבל(אם בכלל יש כאלו. אם לא, המשתנה השני יהיה 0. בכל מקרה שיש ולו בית אחד מזובל, הערך הבוליאני יהיה true)

במקרה והמשתנה אינו מזובל (הערך הבוליאני הוא false) ערך הבתים שהם בסדר יהיה כל גודל המשתנה.

עליכם לכתוב רשימה של unit tests המשתמשים בassert כדי לבדוק את הפונקציה (אל תנסו לחשוב על המימוש שלה, מבחינתכם היא אינה כריעה) עליכם לכתוב לפחות 3 טסטים של מקרי קצה. כתבו כמה שיותר!

.3

א.סמי המסובב לא מזמן נתן לי API שהוא תכנת, כדי שאוכל סוף סוף לבנות את התוכנית בעזרתה אשתלט על העולם. כששאלתי אותו "אחי, זה בטוח מחריגות (exception safe)? " הוא אמר לי: "בטח אחי, בטוח כמו שאני בטוח."

ואולם, כשהשתמשתי במחלקות שלו, לאחר הרצה, גיליתי פתאום שהמון משתנים אצלי נהיו מזובלים באמצע התוכנית. כשהתקשרתי אליו בזעם לשאול מה נראה לו, ענה לי: "אחי, נתתי לך guarantee, לא עפה לך התוכנית, נכון? , אז מה אתה רוצה?"

איזה guarantee בדיוק נתן לי סמי? ומה אני חשבתי שהוא ייתן לי?

ב. לאחר שהתייאשתי מסמי, קראתי לאורקל שמעבר להרים, והוא סיפק לי מכונת טיורינג עם גישה אליו (היא נמצאת בקובץ הקוד תחת השם takeOverTheWorld) שאמורה לבצע משהו...

ואולם, גם הפונקציה הזו הכזיבה ולא נתנה לי את התוצאות הרצויות. האם תוכלו לשפר את הקוד עד כדי strong guarantee?

4. תנו שני הבדלים לפחות, אחד סמנטי ואחד באופן הכתיבה, בין constant לבין konstatn (רמז: זה לא האות הראשונה)

5. מדוע לא כדאי להשתמש בmacro? תנו 3 סיבות.

6. הציגו 4 יתרונות של פנקטורים לעומת פוינטר לפונקציה

7. הביטו בקוד בו מופיעה המחלקה ששמה vicious (בקובץ) . מה היא עושה? מה היא מחשבת?

<u>**תשובות**(ייתכנו תשובות טובות נוספות, אלו רק התשובות שלי)</u>

<u>שאלה 1:</u>

1.א. 1. לא קבוע ב{}, לפעמים כך ולפעמים אחרת 2. קובנציית שמות משתנים/פונקציות לא קבועה 3. יש משתנים כמו power שראוי שיהיו unsigned 4. אין const על משתני מחלקה קבועים כמו m_alias או שמו של המבצר. 5. אין const על פונקציות שלא משנות את האובייקט כמו getAlias 6. אפשר להוסיף assert בפונקציות כמו growUp כדי לוודא שהן באמת פעלו 7. שגיאה קטלנית! מחיקת

BEAST וstrongHold בפונצקיות כמו setBeast מבלי לבדוק שאולי בכלל קיבלנו את אותו האובייקט! תיקונים אפשריים הם: 1. לבדוק קודם שזה לא אותו אחד(נאיבי) 2. להשתמש בauto_ptr במקום בפוינטרים הרגילים כדי שנוכל לעקוף את כל הבעיה הזו. 8. שימוש בשם alias עם הסבר ליד, אפשר פשוט לשנות לname כדי שיבינו מה רוצים מהחיים. 9. בקונסטקטורים יש ארגומנטים עם שמות חלקיים, כיוון שלא משתמשים ב<-this כדי לאתחל את משתני המחלקה, זה מעצבן ומבלגן.

10. BEAST ולא beast (אותיות גדולות איפה שלא צריך) 11. M_mastery בעל אפשרויות מסוימות בלבד, היה עדיף לעשות enum 12. שם ארוך מדי לאחת הפונקציות, צריך לדעת גם מתי לקצר 13. כל ה getter וsettersי ממש מבולגנים, גם זה יוצר בעיות 14. בדיחות טיפשיות לאורך הקוד, יש אנשים שזה יפריע להם

אפשר עוד להוסיף...

ב. 1 : 1. אין סוגריים על תנאי ה if 2. המשתנה casualties לא מאותחל כשהוא מוכרז 3. הscope שלו גדול מדי 4. התיעוד מראה שאם הוחזר 0, ייתכן שהפונקציה נכשלה, אולם ייתכן גם שהערך שהוחזר על ידי הפונקציות המופעלות הוא באמת 0. מכאן שהערך החוזר אינו ברור במשמעותו, וצריך להחזיר struct, שאחד ממשתניו יהיה הצלחה/ כישלון

2. 3 דרכים: הראשונה היא להשתמש בפוינטר לפונקציות blaze, horrify וכו' שיישלח בזמן השימוש. הדרך השנייה היא בעזרת פנקטור, והשלישית היא בעזרת מחלקות בן לdarklord שידרסו את המימוש. דוגמאות לכך ניתן למצוא בקובץ התשובות.

לפנקטור יש יתרונות על פני האחרים, כפי שנראה בשאלה 6

<u>תשובה 2:</u>

בקובץ התשובות (solutione). (פונקציית assertionCases)

<u>תשובה 3:</u>א. סמי המסובב הזה כמובן נתן לי no leak guarantee, כשאני חשבתי על לפחות basic.

ב. התשובה בקובץ solutione (לא, זו לא שגיעט קטיוו, אני פשוט מעדיף לקרוא לזה ככה)

<u>תשובה 4:</u> ההבדל הוא שבזמן שconstant רוצה לדבר על משתנה רגיל/מחלקה שלא אמור להשתנות, אבל בפועל במקרים מסוימים אנו מסוגלים ועשויים לשנותו (static_cast או mutable) constant הוא קבוע ברמת הdefine (גם אם לא כך הוא מוגדר) שלא אמור להשתנות לעולם.

ההבדל מבחינת הקוד, הוא ש konstant הינו משתנה גלובלי/סטטי (כדי לשמור על ייחודו ועל העובדה שהוא אינו אמור לעבור שינוי על ידי שום חלק בקוד), ואילו constant יכול להיות כל משתנה, או משתנה מחלקה.

<u>תשובה 5:</u> 1. Macro עלולים לתת שגיאות מאוד מעצבנות מכיוון שהם אינם שומרים על סוגריים או על {}, ולכן אפשר לקבל כל מיני מרעין בישין של שגיאות לוגיות 2. Macro מחליף כל מקום בקוד עם השם שניתן לו, מה שגורם לפעמים שהוא יחליף שמות משתנים או פונקציות שבכלל לא תכננתם עבורם את הגורל הזה!

3. קשה להשתמש במאקרו בעל יותר משורה אחת.

4. לא מודע לטיפוסים(types)

תשובה 6: 1. פנקטור הרבה יותר מותאם לסביבה מונחית עצמים 2. לפנקטור אפשר לתת state, כלומר משתנים שיישמרו איתו ויתנו אינדקציה לגבי השימוש בו 3. לפנקטור אפשר להוסיף בקלות כל מיני מאפיינים משעשעים, שמאפשרים לשנות את צורת מימוש הפונקציה מבלי לכתוב אחת נוספת (customizable) בלע"ז 4. פנקטורים מאפשרים שימוש בtemplate יחד איתם בצורה יחסית נוחה 5. אפשר להשתמש בinline איתם 6. הקריאה להם היא קריאה ישירה, ולא קריאה עקיפה 7. החתימה של הפונקציה יכולה להשתנות בהתאם לנוחיות.

תשובה 7: מחשבת את סדרת פיבונאצ'י, בזמן קומפילציה (האם זה סותר את העובדה שזמן החישוב שם אמור להיות אקספוננציאלי? מחשבות?)