# תכנות מתקדם ושפת ++C – תרגיל 2

## Heap Memory Manager

תאריך הגשה:

**הנחיות:** קראו היטב את ההנחיות, אי עמידה בהנחיות אלו תגרור הורדת ציון.

- העבודה וההגשה ביחידים בלבד!
- יש להגיש למערכת ה moodlearn עד מועד ההגשה, אם יש סיבה מוצדקת לאיחור בהגשה יש לדווח עד יומיים לפני מועד ההגשה, לאחר מכן לא תתקבל שום סיבה לאיחור.
- יש להגיש את כל הקבצים בתיקייה המכילה את קבצי הפרויקט cpp. וקבצי h. ובנוסף יש להגיש קובץ cmake ואת קובץ ה makefile שנוצר, את התיקייה יש לכווץ לקובץ ששמו הוא ת"ז של הסטודנט.
  חובה לכתוב בקובץ CmakeLists.txt עבור כל פקודה תיעוד מפורט מה הפקודה עושה.
- שימו לב שחובה שהפרויקט יעבור קומפילציה בקומפיילר g++. ניתן להשתמש גם ב ++C11
- שימו לב לתכנות נכון ולפי כל הכללים של "תכנות מונחה עצמים" שנלמדו בכיתה. בהחלט מותר להוסיף פונקציות או מחלקות נוספות מעבר למה שאתם נדרשים בתרגיל ולהגדיר את הקשרים ביניהם לצורך תכנות נכון בפרט יש לשים לב:
  - מתי יש להשתמש בירושה ובפולימורפיזם.
  - אלו פונקציות יהיו וירטואליות.
  - מתי להשתמש במחלקות אבסטרקטיות.
  - לשמור על עיקרון הכימוס.
  - לשים const במקומות שצריך.
  - מתי להשתמש במצביע, מתי ברפרנס ומתי בערך רגיל.
  - חובה לתעד כל פונקציה שאתם כותבים.
  - יש להימנע ממספרי קסם.
  - חובה עליכם לבצע בדיקות שאין לקוד שלכם דליפת זיכרון.
- לפני שאתם מתחילים לעבוד על התרגיל מומלץ לעבור ולהבין את כל המטלה כדי שתוכלו לכתוב ולעצב את הקוד בצורה הנכונה ביותר.

## *המוטיבציה*

שימוש בהקצאות זיכרון דינאמיות בתוכנית (malloc, free, new, delete) פונות ל-kernel של מערכת ההפעלה בבקשה להקצאת ושחרור זיכרון. זה אומר שמערכת ההפעלה צריכה לעבור מ-user space code, בו היא מריצה את התוכניות השונות, ל-kernel code כל פעם שיש בקשת זיכרון בתוכנית. אם זה נעשה בצורה תדירה, זה משפיע על מהירות התוכנית, והיא תרוץ יותר לאט. כתיבת מערכת ניהול זיכרון שניגשת פחות למערכת ההפעלה בבקשות להקצאת ושחרור זיכרון תורמת ליצירת תוכניות יעילות ומהירות יותר, אם הן נכתבות בצורה נכונה.

# כללי

בתרגיל זה נממש Heap Memory Manager. הוא ינהל את כל מה שקשור לבקשות של הקצאת זיכרון דינאמית ויהווה תחליף לגישות למערכת ההפעלה. הוא ישפיע על כלל ההקצאות הדינאמיות, עבור כל הטיפוסים שתשתמשו בהם בתרגיל.

התרגיל יהיה מורכב מכמה חלקים:

1. **כתיבת שכבת התשתית** – בה נטפל בניהול זיכרון ה-heap עצמו.
2. **כתיבת הממשק** – בה נממש את operator new, operator new[], operator delete, operator delete[].
3. **בדיקת המערכת שכתבתם**.

# חלק ראשון : כתיבת התשתית

### 1. מחלקת MemPool

בתשתית המערכת נגדיר מחלקה בשם **MemPool** שתהיה Singleton והיא תחזיק את ה-pool של הזיכרון, ממנו נקצה את הזיכרון למי שיבקש. המחלקה צריכה לנהל רק את ה-pool הזה, בלי קשר למי משתמש בה. ה-pool הוא בעצם מערך רציף של char-ים שמוקצה על ה-heap (המקורי).

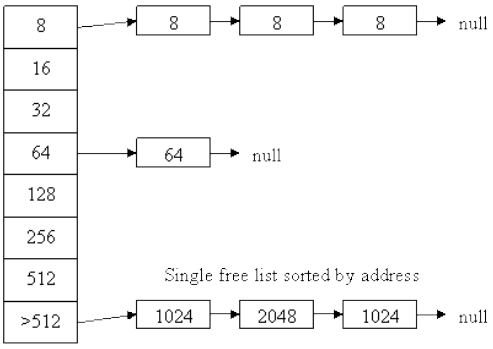
### 2. מחלקת MemoryManager

בנוסף, נגדיר מחלקה בשם **MemoryManager**, שתהיה אחראית לאתחל בעליית המערכת את ה-pool לפי גודל שיתקבל בהרצת התוכנית, וכן לממש את הממשק שקשור לבקשות הקצאת ושחרור הזיכרון הדינאמיות.

בעליית התוכנית יתקבל ארגומנט שיגיד כמה בתים אנו רוצים להגדיר כ-"heap", ומחלקת MemoryManager תקצה מערך בגודל הזה ב-MemPool. זה יתבצע באמצעות flag של S- לתוכנית. למשל: ./myProg -S 2000. תקצה "heap" של 2000 בתים.

כל בקשה של הקצאת זיכרון תגיע (בצורה ישירה או עקיפה) למחלקת MemoryManager ותבקש ממנה קטע מהמערך בגודל הרצוי. האלגוריתם בו תשתמשו על מנת למצוא את הבלוק הפנוי הראשון המתאים יהיה FirstFit, אך עליכם לבנות את התוכנית כך שנוכל בקלות לעבור בין FirstFit לבין אלגוריתמים אחרים כגון BestFit וכד'. (הצעה למימוש אפשר למצוא באינטרנט תחת Strategy Pattern).

כדי לייעל את החיפוש על ה-pool אחר בלוק פנוי, נשתמש ברשימה מקושרת (free-list) שתשמור מצביעים לבלוקים הפנויים ב-pool. נחזיק מספר רשימות מקושרות, לפי הגודל של הבלוקים. כדי ליצור מערכת יעילה, נאפשר בלוקים שגודלם הוא חזקה של 2 בלבד. את רשימת הבלוקים הפנויים נבנה כ-hash table שממפה את הגודל המבוקש (אחרי הנרמול לחזקה של 2) לרשימה מקושרת של כל הבלוקים הפנויים בגודל 8 בתים, בגודל 16 בתים וכו', כך עבור גדלים שהם חזקות של 2, כפי המתואר באיור הבא (הלקוח מההרצאה):




עליכם להשתמש במבני הנתונים של ספריית STL (עליכם להסביר את הבחירה). ניתן למצוא הרבה תיעודים והסברים באינטרנט.

אופן העבודה עם ה-free list יהיה כדלהלן: בכל בקשה להקצאת זיכרון אפשר לחפש זיכרון פנוי בגודל המתאים ב-free list, או לייצר בלוק חדש בגודל המתאים על ה-MemPool. בכל מקרה שחרור של בלוק מכניס את הבלוק למקום המתאים ב-free list. אנו משאירים לכם את ההחלטה כיצד להגדיר את האלגוריתם לבחירה מהיכן לקחת את הבלוק המבוקש. על התרגיל כתבו בהערה מה ה-design שלכם ומה סיבוכיות הזמן של המימוש שלכם. היעזרו בחומר הנלמד בהרצאה בנושא.

לסיכום, מחלקת MemoryManager צריכה לספק ממשק ברור לבקשות של הקצאות זיכרון ושחרור זיכרון, ועושה זאת תוך שימוש ב-free list ו-MemPool. ניתן להגדיר את המתודות כסטאטיות לשם נוחות.

## חלק שני: כתיבת הממשקים new ו-delete

כפי ראינו בהרצאות, כל פניה לפונקציה new, מובילה קודם כל ל-operator new (אותו הדבר לגבי מערכים בהתאמה). מימוש האופרטורים new ו-[]new עבור כלל הטיפוסים (בסיסיים ושאינם כאלה) אמורים לא להגיע לפונקציות שמערבות את מערכת ההפעלה, אלא ל-MemoryManager.

לכן, עליכם להגדיר מחדש את operator new, operator new[], operator delete, operator delete[] כך שיעבדו עם MemoryManager בצורה נכונה ללא דליפות זיכרון.

## חלק שלישי: בדיקת המערכת שלכם

את המערכת שבניתם נרצה לבדוק ב-2 דרכים:

- <u>נכונות</u>: עליכם לממש תוכנית בדיקה המשתמשת ב MemoryManager:

עליכם לממש רשימה מקושרת מעגלית דו כיוונית גנרית.
עליכם לממש תוכנית ראשית אשר תבדוק את הרשימה על אובייקטים מסוג std::pair<char*,long*>* - אין להשתמש במחלקת string.

- valgring: עליכם לממש מנגנון בדיקת דליפת זיכרון הדומה ל valgrind שיציג בתצוגה מינימאלית את כמות הזיכרון הדולף (בבתים) כאשר מריצים את התוכנית עם ה v– flag.
דוגמת הרצה:

```
./myprog –S 2000 –v
…
Bytes allocated and not released : 128
```

המימוש צריך להיות כפוף לרעיון ה Single Responsibility Principle (וה SOLID).
עליכם להסביר את צורת המימוש.

- בונוס(20 נקודות):
א. בסעיף ה valgrind לעשות תצוגה מתקדמת המציגה את הנתונים הבאים:
  1. כמה בלוקים הוקצו וכמה בתים סה"כ.
  2. כמה בלוקים מכל גודל הוקצו – יש להציג היסטוגרמה.
  3. עבור בלוקים דולפים:
     a. כמה בלוקים דולפים וכמה בתים סה"כ.
     b. היכן הוקצו הבלוקים (רשימה של המקומות בהם התבצעה ההקצאה).
ב. בנוסף, יש לכתוב gtest למערכת ניהול הזיכרון.

## נקודות חשובות:

- אין ליצור אובייקטים מלבד אובייקטי המערכת של ניהול הזיכרון על ה stack. כל יצירה של אובייקט תהיה אך ורק דרך האופרטורים new או []new.
- יש להגדיר את ה flags כ case sensitive.
- יש לתעד את כל הפונקציות
- הבונוס יינתן רק למי שעשה את כל המטלה.