

מבנה נתונים:

1. מחלקת Block:

ייצוג של בלוק זיכרון (תפוס או פנוי) בתוך מאגר הזיכרון.

שדות:

- size – גודל הבלוק (לא כולל המטא-דאטה).
- isFree – האם הבלוק פנוי או לא.
- next / prev – מצביעים לבלוקים הסמוכים בזיכרון.

מתודות:

- בנאים ואתחולים.
- פונקציות לקבלת גודל / סטטוס תפוס / כתובות סמוכות.
- split() – פיצול בלוק.
- merge() – איחוד בלוקים חופשיים.

2. מחלקת MemoryManager:

מטפלת בהקצאה ושחרור זיכרון על פי אחד האלגוריתמים שנבחר.

שדות:

- m_memoryPool – מצביע ל-MemoryPool (לדוגמה: 1024 בתים).
- m_totalSize – גודל המאגר הכולל.
- m_usedSize – כמות זיכרון מוקצת בפועל.
- מצביע לבלוק ההתחלתי (Block* m_head).

מתודות:

- allocate(size_t size) – הקצאת בלוק לפי האסטרטגיה הנבחרת.
- deallocate(void* ptr) – שחרור בלוק.
- reset(size_t size) – איפוס הזיכרון.
- הדפסת מצב הזיכרון (>>operator).

3. מחלקות היורשות: FirstFitAllocator, BestFitAllocator, WorstFitAllocator

מימוש של שלוש אסטרטגיות הקצאה:

- First Fit – מקצה בבלוק הפנוי הראשון שמתאים.
- Best Fit – מקצה בבלוק הפנוי הקטן ביותר שמתאים.
- Worst Fit – מקצה בבלוק הפנוי הגדול ביותר.

תיאור השלבים:

1. אתחול:

- מוקצה מאגר זיכרון בגודל קבוע מראש.
- מוגדר בלוק ראשי פנוי בגודל כללי.

2. הקצאה:

- לפי גודל מבוקש, מבוצע חיפוש בבלוקים הקיימים לפי אסטרטגיית First/Best/Worst.
- אם הבלוק גדול מדי, הוא מפוצל לשניים.

3. שחרור:

- הבלוק מסומן כ-free.
- במידת האפשר, מתבצע איחוד עם בלוקים חופשיים סמוכים (merge()).

4. הדמיה:

- קטע הקוד MemorySimulator.cpp מפעיל את שלושת האלגוריתמים לפי פרוטוקול ובודק: אחוז הקצאות שנכשלו. שימוש מירבי בזכרון. ניהול נכון של שחרור ואיחוד. בדיקת מקרי קצה.

תיאור התנאים המדויקים שהמחלקות חייבות לקיים:

1. Block:

- חובה לכלול שדה בשם isFree ולנהל אותו בצורה עקבית.
- בעת פיצול, שדה הגודל של כל חלק מעודכן בצורה נכונה.
- יש לשמור על שלמות הרשימה הדו-כיוונית של הבלוקים.

2. MemoryManager:

- להשתמש ב placement new בתוך המאגר.
- אין דליפות, כל שחרור מתבצע כהלכה.
- טיפול במקרי קצה.

3. מחלקות היורשות: FirstFitAllocator, BestFitAllocator, WorstFitAllocator

- מממשות בצורה תקינה את הפונקציה findFreeBlock.
- אין כפילויות לוגיקה, כלומר: הקצאה מבוססת תמיד על הפונקציה המופשטת.

מימוש מטלה מספר 2:

לאחר מימוש כל אחת ממחלקות הבת (FirstFitAllocator, BestFitAllocator, WorstFitAllocator), כלי הAl (כנדרש במשימה) השתמש באותם להשתמש באותם פונקציות בדיקה, כך שביקשתי ממנו בסוף לאחד את זה לקוד אחד מלא.

זהו להלן השלד, לכל אחד מהם, ולאחר מכאן, נוכל לראות בבירור את הטסט לכל אחד ממחלקות הבת.

test.cpp 

Edit  Copy 

cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cassert>
#include <crtDBG.h>
#include <cstring>
#include "FirstFitAllocator.h"
#include "BestFitAllocator.h"
#include "WorstFitAllocator.h"

using namespace std;

const int POOL_SIZE = 1024;

void printBlocks(const MemoryManager& allocator)
{
    cout << allocator << endl;
}

void testAllScenarios(MemoryManager& allocator)
{
    cout << "=== Testing: " << allocator.getAlgorithmName() << " ===" << endl;

    // 1. Basic allocation
    void* a = allocator.allocate(100);
    void* b = allocator.allocate(100);
    void* c = allocator.allocate(100);
    assert(a && b && c);
    printBlocks(allocator);

    allocator.deallocate(a);
    allocator.deallocate(c);
    allocator.deallocate(b); // middle one last
    printBlocks(allocator);
}
```

Edit  Copy 

```
// 3. Edge-split allocation (just enough to avoid creating another block)
void* big = allocator.allocate(POOL_SIZE - 100 - sizeof(Block) * 2);
printBlocks(allocator);
assert(big);
allocator.deallocate(big);

// 4. Invalid allocation size (zero or negative)
void* zero = allocator.allocate(0);
assert(zero == nullptr);

// 5. Deallocating nullptr should do nothing
allocator.deallocate(nullptr); // Should be safe

// 6. Allocate as many blocks as possible until memory is exhausted
void* blocks[50];
int count = 0;
while ((blocks[count] = allocator.allocate(64))) {
    count++;
}
cout << "Allocated " << count << " blocks of 64 bytes" << endl;

// 7. Attempt to deallocate an address not from the pool
int dummy = 42;
cout << "[Expected safe behavior] Attempting to deallocate external address..." << endl;
allocator.deallocate(&dummy); // Should not crash

// 8. Double free test
if (count >= 2) {
    allocator.deallocate(blocks[0]);
    allocator.deallocate(blocks[0]); // Should not crash
}

// 9. Test getHeader directly
const Block* header = allocator.getHeader();
assert(header != nullptr);
cout << "Header size: " << header->getSize()
    << ", isFree: " << (header->isFree() ? "true" : "false") << endl;
```



```

// 10. Reset during usage (should clear everything)
allocator.reset(POOL_SIZE);
assert(allocator.getUsedMemory() == 0);
assert(allocator.getFailedAllocations() == 0);

cout << "[All checks passed for: " << allocator.getAlgorithmName() << "]\n" << endl;
}

void testStrategyLogic(MemoryManager& allocator)
{
    cout << "=== Strategy Logic Test: " << allocator.getAlgorithmName() << " ===" << endl;

    allocator.reset(POOL_SIZE);

    // Step 1: Allocate three blocks
    void* b0 = allocator.allocate(50); // Block 0
    void* b1 = allocator.allocate(200); // Block 1
    void* b2 = allocator.allocate(100); // Block 2
    void* b3 = allocator.allocate(300); // Block 3
    void* b4 = allocator.allocate(50); // Block 4 used
    assert(b1 && b2 && b3);

    // Step 2: Deallocate blocks
    allocator.deallocate(b0);
    allocator.deallocate(b1);
    allocator.deallocate(b2);
    allocator.deallocate(b3);
    printBlocks(allocator);

    // Step 3: Allocate 100 bytes, algorithm should select different block
    void* testAlloc = allocator.allocate(100);
    assert(testAlloc);

    // Get header to check size of selected block
    Block* header = (Block*)((char*)testAlloc - sizeof(Block));
    Block* nextBlock = header->getNext();
    int actualSize = nextBlock->getSize();
    printBlocks(allocator);

```



```

// Verify according to strategy
const char* algo = allocator.getAlgorithmName();

if (strcmp(algo, "first fit algorithm") == 0)
{
    assert(actualSize == 84);
    cout << "First Fit selected block size: " << actualSize << " (Expected:use block with size
}
else if (strcmp(algo, "best fit algorithm") == 0)
{
    assert(actualSize == 300);
    cout << "Best Fit selected block size: " << actualSize << " (Expected: use block with size
}
else if (strcmp(algo, "worst fit algorithm") == 0)
{
    assert(actualSize == 184);
    cout << "Worst Fit selected block size: " << actualSize << " (Expected: use block with siz
}
else
{
    assert(false && "Unknown allocator type");
}

cout << "[Strategy logic test passed for: " << algo << "]\n" << endl;
}

```

:FirstFitAllocator

main_first.cpp

```
Edit Copy  
cpp  
  
int main()  
{  
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);  
  
    FirstFitAllocator first(PPOOL_SIZE);  
    testAllScenarios(first);  
    testStrategyLogic(first);  
  
    std::cout << " All extreme tests completed successfully.\n";  
    std::cout << "Leaks: " << _CrtDumpMemoryLeaks() << std::endl;  
    return 0;  
}
```

:Results of the test

```
=== Testing: first fit algorithm ===  
Block number 0  
Size: 100  
Status: used  
Block number 1  
Size: 100  
Status: used  
Block number 2  
Size: 100  
Status: used  
Block number 3  
Size: 660  
Status: free  
peak usage: 300 bytes  
total size: 1024 bytes  
used size: 300 bytes  
Free size(not include headers!): 724 bytes  
Actual Free size you have: 660 bytes  
failed allocations: 0 bytes  
  
Block number 0  
Size: 100  
Status: free  
Block number 1  
Size: 892  
Status: free  
peak usage: 300 bytes  
total size: 1024 bytes  
used size: 0 bytes  
Free size(not include headers!): 1024 bytes  
Actual Free size you have: 992 bytes  
failed allocations: 0 bytes  
  
Block number 0  
Size: 100  
Status: free  
Block number 1  
Size: 892  
Status: used  
peak usage: 892 bytes  
total size: 1024 bytes  
used size: 892 bytes  
Free size(not include headers!): 132 bytes  
Actual Free size you have: 100 bytes  
failed allocations: 0 bytes  
  
Allocated 12 blocks of 64 bytes  
[Expected safe behavior] Attempting to deallocate external address...  
Warning you attempted to deallocate pointer that outside of memory pool.  
  
Header size: 100, isFree: true  
[All checks passed for: first fit algorithm]
```

```

=== Strategy Logic Test: first fit algorithm ===
Block number 0
Size: 50
Status: free
Block number 1
Size: 200
Status: free
Block number 2
Size: 100
Status: free
Block number 3
Size: 300
Status: free
Block number 4
Size: 50
Status: used
Block number 5
Size: 228
Status: free
peak usage: 700 bytes
total size: 1024 bytes
used size: 50 bytes
Free size(not include headers!): 974 bytes
Actual Free size you have: 878 bytes
failed allocations: 0 bytes

Block number 0
Size: 50
Status: free
Block number 1
Size: 100
Status: used
Block number 2
Size: 84
Status: free
Block number 3
Size: 100
Status: free
Block number 4
Size: 300
Status: free
Block number 5
Size: 50
Status: used
Block number 6
Size: 228
Status: free
peak usage: 700 bytes
total size: 1024 bytes
used size: 150 bytes
Free size(not include headers!): 874 bytes
Actual Free size you have: 762 bytes
failed allocations: 0 bytes

First Fit selected block size: 84 (Expected:use block with size 200))
[Strategy logic test passed for: first fit algorithm]

All extreme tests completed successfully.
Leaks: 0

```

משימה 2.2: משוב OOP על מחלקת FirstFitAllocator

הסבר	מתקיים?	עיקרון OOP
המחלקה יורשת מ־ <code>MemoryManager</code> ומרחיבה אותה	✓	ירושה
מממשת את הפונקציות הווירטואליות <code>allocate</code> ו־ <code>getAlgorithmName</code>	✓	פולימורפיזם
הכותרות נמצאות בקובץ <code>FirstFitAllocator.h</code> , המימוש ב־ <code>implementation.cpp</code>	✓	הפרדת ממשק ממימוש
המחלקה מוסיפה פונקציונליות (<code>findFirstFit</code>) מבלי לשבור את מחלקת הבסיס	✓	שימוש נכון בהרחבה
השמות תיאוריים והפונקציות ממוקדות	✓	שמות ברורים ופונקציות קצרות
לפי דרישות המטלה – לא להשתמש ב־ <code>override</code>	✗ כוונה לא	שימוש במילת <code>override</code>

:BestFitAllocator

main_best.cpp

```
Edit Copy

int main()
{
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

    BestFitAllocator best(POOL_SIZE);
    testAllScenarios(best);
    testStrategyLogic(best);

    std::cout << " All extreme tests completed successfully.\n";
    std::cout << "Leaks: " << _CrtDumpMemoryLeaks() << std::endl;
    return 0;
}
```

:Results of the test

```
=== Testing: best fit algorithm ===
Block number 0
Size: 100
Status: used
Block number 1
Size: 100
Status: used
Block number 2
Size: 100
Status: used
Block number 3
Size: 660
Status: free
peak usage: 300 bytes
total size: 1024 bytes
used size: 300 bytes
Free size(not include headers!): 724 bytes
Actual Free size you have: 660 bytes
failed allocations: 0 bytes

Block number 0
Size: 100
Status: free
Block number 1
Size: 892
Status: free
peak usage: 300 bytes
total size: 1024 bytes
used size: 0 bytes
Free size(not include headers!): 1024 bytes
Actual Free size you have: 992 bytes
failed allocations: 0 bytes

Block number 0
Size: 100
Status: free
Block number 1
Size: 892
Status: used
peak usage: 892 bytes
total size: 1024 bytes
used size: 892 bytes
Free size(not include headers!): 132 bytes
Actual Free size you have: 100 bytes
failed allocations: 0 bytes

Allocated 12 blocks of 64 bytes
[Expected safe behavior] Attempting to deallocate external address...
Warning you attempted to deallocate pointer that outside of memory pool.

Header size: 100, isFree: true
[All checks passed for: best fit algorithm]
```

```

=== Strategy Logic Test: best fit algorithm ===
Block number 0
Size: 50
Status: free
Block number 1
Size: 200
Status: free
Block number 2
Size: 100
Status: free
Block number 3
Size: 300
Status: free
Block number 4
Size: 50
Status: used
Block number 5
Size: 228
Status: free
peak usage: 700 bytes
total size: 1024 bytes
used size: 50 bytes
Free size(not include headers!): 974 bytes
Actual Free size you have: 878 bytes
failed allocations: 0 bytes

Block number 0
Size: 50
Status: free
Block number 1
Size: 200
Status: free
Block number 2
Size: 100
Status: used
Block number 3
Size: 300
Status: free
Block number 4
Size: 50
Status: used
Block number 5
Size: 228
Status: free
peak usage: 700 bytes
total size: 1024 bytes
used size: 150 bytes
Free size(not include headers!): 874 bytes
Actual Free size you have: 778 bytes
failed allocations: 0 bytes

Best Fit selected block size: 300 (Expected: use block with size 100)
[Strategy logic test passed for: best fit algorithm]

All extreme tests completed successfully.
Leaks: 0



```

משימה 2.2 – משוב על מחלקת BestFitAllocator

הסבר	מתקיים?	עיקרון OOP
המחלקה יורשת באופן תקין את <code>MemoryManager</code>	✓	יחשה
מממשת את הפונקציות הווירטואליות <code>allocate()</code> ו- <code>getAlgorithmName()</code>	✓	פולימורפיזם
הכותרות נמצאות ב- <code>BestFitAllocator.h</code> , המימוש ב- <code>implementation.cpp</code>	✓	הפרדת ממשק ממימוש
המחלקה מוסיפה את <code>findBestFit()</code> בלבד, מבלי לשבור או לשכתב את פונקציות הבסיס	✓	שימוש נכון בהרחבה
השמות תיאוריים, ממוקדים וברורים	✓	שמות ברורים ותיעוד
לפי דרישות המטלה – לא להשתמש במילת <code>override</code>	✗ כוונה לא	שימוש במילת <code>override</code>

:WorstFitAllocator

main_worst.cpp 

```
Edit  Copy  cpp

int main()
{
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

    WorstFitAllocator worst(POOL_SIZE);
    testAllScenarios(worst);
    testStrategyLogic(worst);

    std::cout << " All extreme tests completed successfully.\n";
    std::cout << "Leaks: " << _CrtDumpMemoryLeaks() << std::endl;
    return 0;
}
```

:Results of the test

```
=== Testing: worst fit algorithm ===
Block number 0
Size: 100
Status: used
Block number 1
Size: 100
Status: used
Block number 2
Size: 100
Status: used
Block number 3
Size: 660
Status: free
peak usage: 300 bytes
total size: 1024 bytes
used size: 300 bytes
Free size(not include headers!): 724 bytes
Actual Free size you have: 660 bytes
failed allocations: 0 bytes

Block number 0
Size: 100
Status: free
Block number 1
Size: 892
Status: free
peak usage: 300 bytes
total size: 1024 bytes
used size: 0 bytes
Free size(not include headers!): 1024 bytes
Actual Free size you have: 992 bytes
failed allocations: 0 bytes

Block number 0
Size: 100
Status: free
Block number 1
Size: 892
Status: used
peak usage: 892 bytes
total size: 1024 bytes
used size: 892 bytes
Free size(not include headers!): 132 bytes
Actual Free size you have: 100 bytes
failed allocations: 0 bytes

Allocated 12 blocks of 64 bytes
[Expected safe behavior] Attempting to deallocate external address...
Warning you attempted to deallocate pointer that outside of memory pool.
Header size: 64, isFree: false
[All checks passed for: worst fit algorithm]
```

```

=== Strategy Logic Test: worst fit algorithm ===
Block number 0
Size: 50
Status: free
Block number 1
Size: 200
Status: free
Block number 2
Size: 100
Status: free
Block number 3
Size: 300
Status: free
Block number 4
Size: 50
Status: used
Block number 5
Size: 228
Status: free
peak usage: 700 bytes
total size: 1024 bytes
used size: 50 bytes
Free size(not include headers!): 974 bytes
Actual Free size you have: 878 bytes
failed allocations: 0 bytes

Block number 0
Size: 50
Status: free
Block number 1
Size: 200
Status: free
Block number 2
Size: 100
Status: free
Block number 3
Size: 100
Status: used
Block number 4
Size: 184
Status: free
Block number 5
Size: 50
Status: used
Block number 6
Size: 228
Status: free
peak usage: 700 bytes
total size: 1024 bytes
used size: 150 bytes
Free size(not include headers!): 874 bytes
Actual Free size you have: 762 bytes
failed allocations: 0 bytes

Worst Fit selected block size: 184 (Expected: use block with size 300)
[Strategy logic test passed for: worst fit algorithm]

All extreme tests completed successfully.
Leaks: 0

```

משימה 2.2: משוב על WorstFitAllocator – עקרונות OOP

הסבר	מתקיים?	עיקרון OOP
יורשת מ־ <code>MemoryManager</code>	✓	ירושה
מממשת <code>allocate</code> ו־ <code>getAlgorithmName</code>	✓	פולימורפיזם
כותרות בקובץ <code>h</code> , מימוש ב־ <code>implementation.cpp</code>	✓	הפרדת ממשק ממימוש
מוסיפה <code>findWorstFit</code> בלבד	✓	הרחבה מדויקת
הקוד ברור וממוקד	✓	שמות פונקציות תיאוריים
לפי דרישות המטלה – לא משתמשים	✗ בכוונה לא	שימוש ב־ <code>override</code>

מימוש מטלה מספר 3:

```
--- Scenario: Random Allocations (first fit algorithm) ---
Failed Allocations: 0%
Peak Usage       : 408 bytes

--- Scenario: Increasing Size Allocations (first fit algorithm) ---
Failed Allocations: 73%
Peak Usage       : 1512 bytes

--- Scenario: Decreasing Size Allocations (first fit algorithm) ---
Failed Allocations: 95%
Peak Usage       : 1960 bytes

--- Scenario: Fragmentation Test (first fit algorithm) ---
Failed Allocations: 66.9643%
Peak Usage       : 1600 bytes

--- Scenario: Burst Allocations (first fit algorithm) ---
Failed Allocations: 69%
Peak Usage       : 1535 bytes

--- Scenario: Mixed Overload (first fit algorithm) ---
Failed Allocations: 50%
Peak Usage       : 1440 bytes

--- Scenario: Random Allocations (best fit algorithm) ---
Failed Allocations: 0%
Peak Usage       : 408 bytes

--- Scenario: Increasing Size Allocations (best fit algorithm) ---
Failed Allocations: 73%
Peak Usage       : 1512 bytes

--- Scenario: Decreasing Size Allocations (best fit algorithm) ---
Failed Allocations: 95%
Peak Usage       : 1960 bytes

--- Scenario: Fragmentation Test (best fit algorithm) ---
Failed Allocations: 66.9643%
Peak Usage       : 1600 bytes

--- Scenario: Burst Allocations (best fit algorithm) ---
Failed Allocations: 69%
Peak Usage       : 1535 bytes

--- Scenario: Mixed Overload (best fit algorithm) ---
Failed Allocations: 50%
Peak Usage       : 1440 bytes

--- Scenario: Random Allocations (worst fit algorithm) ---
Failed Allocations: 0%
Peak Usage       : 408 bytes

--- Scenario: Increasing Size Allocations (worst fit algorithm) ---
Failed Allocations: 73%
Peak Usage       : 1512 bytes

--- Scenario: Decreasing Size Allocations (worst fit algorithm) ---
Failed Allocations: 95%
Peak Usage       : 1960 bytes

--- Scenario: Fragmentation Test (worst fit algorithm) ---
Failed Allocations: 66.9643%
Peak Usage       : 1600 bytes

--- Scenario: Burst Allocations (worst fit algorithm) ---
Failed Allocations: 69%
Peak Usage       : 1535 bytes

--- Scenario: Mixed Overload (worst fit algorithm) ---
Failed Allocations: 48%
Peak Usage       : 1408 bytes
```