DYNAMIC MEMORY IN C++: A DIDACTIC APPROACH

MEMORIA DINAMICĂ ÎN C++: O ABORDARE DIDACTICĂ

CZU: 004.424.74 **https://doi.org/10.59295/spd2025e.110**

Iulia BORIS
ORCID: 0000-0003-4523-2182
E-mail: borisiulia@yahoo.com
Moldova State University, Chisinau, Republic of Moldova

Vadim SIRKELI
ORCID: 0000-0002-2479-4218
E-mail: vadim.sirkeli@gmail.com
Moldova State University, Chisinau, Republic of Moldova

Denis NIKA
ORCID: 0000-0002-3082-3118
E-mail: dlnika@yahoo.com
Moldova State University, Chisinau, Republic of Moldova

Sumar. *Alocarea și dealocarea dinamică a memoriei reprezintă aspecte esențiale ale programării de sisteme, în special în limbaje fără colectare automată a gunoiului, precum C și C++. Studierea acestui subiect necesită din partea studenților nu doar dezvoltarea abilităților tehnice, ci și dobândirea unei înțelegeri profunde a principiilor de gestionare a resurselor, adresării memoriei și funcționării managerului de memorie al sistemului de operare. Acest articol prezintă recomandări metodologice care contribuie la simplificarea înțelegerii de către studenți a conceptelor legate de alocarea dinamică a memoriei. Sunt discutate tehnici eficiente de predare, orientate spre formarea unor competențe solide și prevenirea dificultăților frecvente în învățarea acestui subiect. Strategiile propuse sunt axate pe îmbunătățirea calității pregătirii studenților și pe cultivarea unei atitudini responsabile față de gestionarea resurselor sistemului.*

Cuvinte-cheie: *alocarea memoriei, operatorii new și delete, C++, educație în domeniul informaticii.*

1. INTRODUCTION

Teaching programming languages, especially system-level languages like C or C++ [1], requires particular attention to the topic of memory management [2-3]. In contrast to many high-level programming languages that provide automatic memory management and garbage collection (such as Python, Java, or Scala) [4-5], C++ gives the programmer full control over memory operations. While this control enables greater efficiency and flexibility, it also introduces a higher level of complexity and risk. Programmers must explicitly allocate and deallocate memory using specific operators, which requires

careful attention to detail and a solid understanding of how memory is structured and managed at a low level.

This manual approach to memory management is both one of C++'s greatest strengths and one of its primary learning obstacles. For students, especially those new to systems programming, the lack of automated memory handling can be daunting. Without a proper grasp of memory allocation principles, students are prone to making serious errors such as memory leaks, dangling pointers, double deletions, or undefined behavior due to accessing uninitialized memory.

A key focus within this area is the study of pointers and dynamic memory allocation operators *new* and *delete*. These operators allow the creation and destruction of objects on the heap, enabling more flexible program structures such as dynamic arrays, linked data structures, and complex object lifetimes. However, because these operations interact closely with the system's memory model, they are also a common source of bugs and logical errors in student code.

This topic is both theoretically rich and technically demanding. It requires students to understand abstract concepts such as memory regions (stack vs. heap), variable lifetime, object construction/destruction, and the *Resource Acquisition Is Initialization* (RAII) paradigm. Unfortunately, these ideas are often poorly understood or absorbed when taught without a strong pedagogical framework.

As a result, teaching memory management in C++ demands a well-structured methodological approach that introduces complexity gradually, reinforces learning through practical examples, and connects theoretical concepts with real-world programming practices. The key instructional considerations for effectively teaching memory management in C++ are discussed below.

2. METHODS

Effective system programming is impossible without an understanding of the fundamental principles of how an operating system works and how it interacts with the computer's hardware components. From this perspective, the topic of *dynamic memory allocation* is one of the most suitable subjects for demonstrating the importance of low-level programming principles. To enhance the comprehension of the material, it is recommended to take the following methodological aspects into account:

2.1. Sequence of Material Presentation

One of the common methodological mistakes is introducing dynamic memory allocation operators too early. Before being introduced to *new* and *delete*, students should have a solid grasp of the following fundamental concepts:

- variables and their scopes;
- stack and heap memory;
- differences between static, automatic, and dynamic memory;

- pointers and references;
- arrays and memory layout.

Only after these foundational topics are well understood does it make sense to proceed to the new and delete operators. It is also important to logically connect this topic with the concept of RAII (Resource Acquisition Is Initialization) as well as with constructors and destructors in classes. This way, students can understand not just the syntax but also the philosophy behind safe resource management.

2.2. Supporting Theory with Practical Analogies

To facilitate understanding of dynamic memory management, it is recommended to use analogies. For example:

Office and rental space: A variable on the stack is like a desk in your office: you use it while working. Dynamic memory is like renting a space: you need to sign a contract (call *new*) and remember to cancel it (*delete*), otherwise you'll keep paying for unused space (a memory leak).

Memory as a warehouse: Pointers are "addresses of storage units", and *new* is a "storage request form": the system tells you exactly where the space has been allocated.

Such metaphors not only make the material more accessible but also help students retain the core principles of memory management.

2.3. Common Challenges and Typical Student Mistakes

In practice, when studying memory management in C++, students often encounter the following difficulties:

- *Misunderstanding the difference between delete and delete[]*

Students are often confused about when to use each operator. It is essential to clearly explain the distinction between a single object and an array of objects in the context of dynamic memory allocation.

- *Memory leaks*

In the early stages, students frequently forget to deallocate memory or do so incorrectly. Therefore, it is important to demonstrate memory-checking tools from the very beginning (ASan, Visual Leak Detector, WinDbg etc.).

- *Use of uninitialized or dangling pointers*

This is one of the most dangerous issues. It's beneficial to show examples where data corruption is visible and explain the concept of undefined behavior.

- *Double deletion of the same memory region.*

Such errors may not manifest immediately, so it is important to emphasize best practices when working with pointers after delete, such as resetting them to *nullptr*.

It is methodologically sound as well to deliberately model such mistakes in early assignments to help students understand their consequences.

2.4. Use of Tools and Debug Environments

Teaching C++ should cover debugging and memory analysis tools as core components. Particularly useful are:

- *Valgrind* for detecting memory leaks and access errors.
- *Address Sanitizer (ASan)*: a built-in memory error detection tool available in *clang* and *gcc*.
- *GDB* or *Visual Studio Debugger* for tracking pointer values and inspecting memory contents.

Integrating these tools into the learning process not only helps prevent errors but also builds professional-grade development skills.

2.5. Gradual Increase in Task Complexity

Practical assignments related to memory management must follow a gradual progression in difficulty:

- Basic memory allocation and deallocation for a single object.
- Working with arrays in dynamically allocated memory.
- Passing pointers to functions.
- Writing simple wrappers (e.g., a class that allocates memory in the constructor and releases it in the destructor).
- Creating custom "*smart pointers*" with basic control features (without using the STL).
- Debugging and analyzing errors in someone else's code.

This kind of progression allows students to develop solid skills without being overwhelmed.

2.6. Connection with Modern C++ and STL

While teaching "manual" memory management remains important, it is equally essential to introduce students to modern alternatives:

- using std::unique_ptr, std::shared_ptr, and std::make_unique;
- benefits of automatic memory management and RAII-based design patterns;
- templates and containers such as std::vector and std::string, which manage memory internally.

In this way, students will gain an understanding not only of the low-level mechanics but also of modern development practices, which is especially important in the context of contemporary C++ (C++11 and beyond).

4. CONCLUSION

Dynamic memory allocation represents a foundational yet complex subject within the study of system-level programming, particularly when working with C++. Due to

its inherent intricacies and the risk of subtle programming errors, it poses significant challenges for students. However, it is also a crucial area for developing a deep understanding of how programs interact with system resources. Effective instruction in this topic requires a comprehensive approach that emphasizes conceptual clarity in memory architecture, pointer behavior, object lifecycles, and secure management of resources.

From a didactic perspective, the teaching of dynamic memory allocation benefits significantly from a carefully structured approach that prioritizes the logical sequencing of foundational topics, the integration of practical metaphors and analogies, and the use of professional debugging and analysis tools. The gradual escalation of task complexity, coupled with explicit modeling of common mistakes, allows students to internalize not only how dynamic memory works but why certain practices are safer and more efficient than others.

This article highlights key strategies for teaching dynamic memory allocation in C++, including a structured progression from basic concepts to *new* and *delete* operators, use of analogies, addressing common student errors, integrating debugging tools, gradually increasing task complexity, and introducing modern C++ features such as smart pointers. These methods aim to develop both conceptual understanding and practical skills while promoting responsible programming practices.

REFERENCES*:*

1. VATICONE, O. C++ System Programming Cookbook. Birmingham: Packt Publishing, 2022. 442 p. ISBN 9781801075104.
2. ROY, P. C++ Memory Management: Write leaner and safer C++ code using proven memory-management techniques. Birmingham: Packt Publishing, май 2025. 434 p. ISBN 978-1-80512-980-6.
3. FRANEK, F. Memory as a Programming Concept in C and C++. Cambridge: Cambridge University Press, 1995. 316 p. ISBN 0-521-52043-6.
4. VAN PUTTEN, M., KENNEDY, S. Java Memory Management: A Comprehensive Guide to Garbage Collection and JVM Tuning. Birmingham: Packt Publishing, November 2022. 146 p. ISBN 978-1-80324-889-9.
5. SEVERANCE, P. (ed.) Python in a Nutshell. 2nd ed. Sebastopol: O'Reilly Media, 2006. Chapter: "Garbage Collection". ISBN 0-596-00188-6.
6. GEEKSFORGEEKS. New and Delete Operators in C++ for Dynamic Memory. In: GeeksforGeeks. [Online]. Available at: https://www.geeksforgeeks.org/new-and-delete-operators-in-cpp-for-dynamic-memory/
7. MAO, L. *Dynamic Memory Management in C++*. In: Lei Mao's Log Book. 2022. Available at: https://leimao.github.io/blog/CPP-Dynamic-Memory-Management/

Acknowledge: *The authors acknowledge the financial support from the Ministry of Education and Research of the Republic of Moldova (research program no. 011208).*