





AI-Generated Programming Solutions: Impacts on Academic Integrity and Good Practices

Chung Man Tang¹, Vanessa S. C. Ng¹, Henry M. F. Leung¹ and Joe C. H. Yuen²

¹*School of Computing and Information Science, Anglia Ruskin University, Cambridge CB1 1PT, U.K.*

²*School of Engineering and Applied Science, Aston University, Birmingham B4 7ET, U.K.*

Keywords: AI-Powered Chatbots, Academic Integrity, ChatGPT, Assessment, Educational Technology.

Abstract: As AI-powered chatbots, notably ChatGPT, gain widespread popularity, their integration into academic settings raises concerns about preserving academic integrity. Students increasingly employ these chatbots to generate answers for assignments and, notably, programming problems. Existing countermeasures, such as plagiarism checkers equipped with AI writing detection capabilities, struggle to detect AI-generated computer programs. To thoroughly examine this challenge, we conducted an experiment, presenting diverse programming problems to ChatGPT. Alarming findings revealed its remarkable proficiency in generating correct solutions across various topics, complexities of problems, and programming languages. To explore the implications, we engaged a focus group of programming teachers, resulting in the identification of key practices and strategies to respond to AI-generated work. These insights provide valuable guidance for educators seeking to maintain integrity while adapting to the evolving role of AI in education.

1 INTRODUCTION


In recent years, AI-powered chatbots, particularly exemplified by ChatGPT, a Large Language Model (LLM), have surged in popularity. These intelligent conversational agents have demonstrated remarkable capabilities and the potential to enhance work efficiency across various domains (Baidoo-Anu & Ansah, 2023; Deng & Lin, 2022; Kalla & Smith, 2023). However, their widespread adoption has not been without consequences, especially in the academic realm (Hong, 2023; Rasul et al., 2023).


One of the striking developments is the utilization of ChatGPT by students to facilitate their academic work (Cotton, Cotton, & Shipway, 2023; Lo, 2023). Beyond simple text-based assistance, students have been found employing ChatGPT for generating answers to assignments, essays, and more. In a rather concerning trend, ChatGPT have even ventured into the realm of computer programming, providing solutions to a variety of programming problems (Rahman & Watanobe, 2023; Surameery & Shakor,


2023). This has implications for academic integrity, raising questions about how to combat academic misconduct when AI-generated work becomes indistinguishable from genuine student efforts (Ventayen, 2023).


To tackle this pressing issue, some countermeasures have emerged (Gao et al., 2022), such as plagiarism checkers equipped with AI writing detection capabilities (Turnitin, 2023). These tools can detect and flag essays that appear to be generated by AI. While they have been effective in handling written assignments, they still fall short when it comes to detecting AI-generated computer programs.

In light of these challenges, this paper embarks on a comprehensive investigation to ascertain the extent of ChatGPT's proficiency in generating accurate program solutions for programming assignments. To achieve this, we collected a diverse set of programming problems from textbooks available in our university library, covering topics commonly taught in elementary programming courses. Some require code creation from scratch, while others

^a <https://orcid.org/0000-0003-2842-6196>

^b <https://orcid.org/0000-0002-2972-530X>

^c <https://orcid.org/0000-0002-7753-0136>

^d <https://orcid.org/0000-0001-9346-4987>

demand the debugging or enhancement of provided code. We also ensured diversity by soliciting solutions for generic programming topics in widely adopted languages like C/C++, Java, and Python, as well as for language-specific problems.

Our findings were striking and, in some ways, alarming. ChatGPT demonstrated the ability to generate correct solutions for the majority of the programming problems we presented, regardless of the topic, language, coding requirements, or complexity of the problem descriptions. To further explore the implications and gather expert insights, we presented our experimental process and results to a focus group consisting of staff members who specialize in teaching programming courses.

The consensus among the focus group members was unequivocal: AI-generated programming solutions pose a significant challenge to academic integrity, and no easy solution exists to tackle this issue within the specifications of programming problems. While program descriptions alone couldn't sufficiently challenge ChatGPT, the focus group engaged in an insightful discussion, resulting in the identification of a list of good practices and strategies to address the issue of AI-generated work.

In this paper, we will detail our experimental methodology, present our findings, and discuss the practical recommendations that have emerged from the deliberations of our focus group. These recommendations hold the potential to offer a way forward in maintaining the integrity of programming assignments while acknowledging the growing influence of AI technologies in academia.

2 AI IN PROGRAM ASSESSMENT

In the ever-evolving landscape of science and technology education, attaining proficiency in programming skills stands as a paramount pursuit for students in computer science and other fields. Programming assignments and exercises play a pivotal role in honing these skills, demanding students to apply problem-solving techniques and unleash their creativity (Krauthwohl, 2002). They also serve as vital assessment tasks for teachers to evaluate students' proficiency and provide constructive feedback for improvement. However, this integral aspect of learning programming poses challenges for both students and teachers.

Students often grapple with intricate problems, investing substantial time and effort in deciphering and addressing programming challenges. Simultaneously, teachers, burdened with assessing

numerous programming exercises and assignments, encounter a time-consuming and potentially error-prone process (Tang, Yu, & Poon, 2009). Traditional methods, such as visually inspecting source code or manually testing programs with predefined cases, contribute significantly to the workload of programming teachers.

Recognizing the imperative for efficiency and an enriched learning experience, several universities have embraced automated program assessment systems over the past decades (Ala-Mutka, 2005; Tang, Yu, & Poon, 2023). These systems not only alleviate the workload for teachers but also provide students with immediate feedback (Lee et al., 2018), fostering a more engaging and interactive learning environment.

A significant development in these automated systems is the integration of plagiarism checkers (Ng, Li, & Ngai, 2004; Yu, Poon, & Choy, 2006). Plagiarism has historically been a concern, and these checkers, designed to identify code similarity within a batch of submissions, have proven effective in curbing academic malpractice (Yu, Poon, & Choy, 2006).

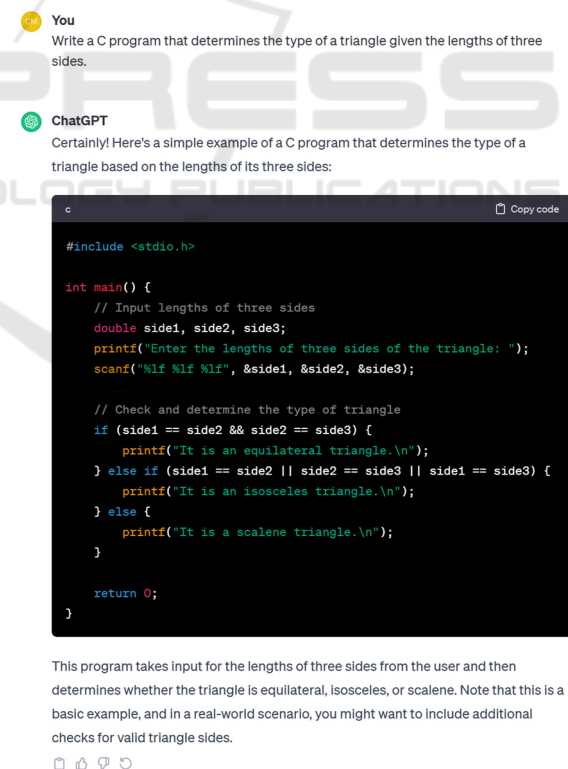


Figure 1: Program source code generated by ChatGPT.

However, the landscape is evolving with the advent of AI in recent years. A concerning trend is

emerging, where students leverage AI tools to effortlessly generate solutions for their programming assignments. Figure 1 illustrates an instance where ChatGPT swiftly generates source code for a task like, “Write a C program that determines the type of a triangle given the lengths of three sides.” The allure of eliminating effort, with ChatGPT providing a solution in seconds along with relevant comments, poses a temptation for students to bypass the rigor of genuine problem-solving.

Unlike conventional plagiarism checkers that compare submitted code against the other submissions in the pool or existing solutions, the randomness inherent in AI-generated programs complicates detection. Each generation may yield different source code, rendering traditional methods inadequate. While there have been advancements in detecting AI-generated essays (Turnitin, 2023), the lack of semantic meaning in program source code impedes the application to function in the programming domain.

As the academic community grapples with this new challenge, it becomes imperative to explore immediate solutions to address the current situation. While researchers work towards identifying ways to deter AI-generated programs, educators and institutions must navigate the evolving landscape, considering strategies to uphold academic integrity and ensure a fair and rigorous evaluation of students’ programming abilities. More importantly, we should explore the potential and power of AI in enhancing the quality of teaching and learning, and assessments, rather than merely considering it as a threat to academic integrity (Rahman & Watanobe, 2023).

In conclusion, integrating AI tools into programming education presents opportunities and challenges (Meyer et al., 2023). Balancing the use of AI for educational enhancement while safeguarding the learning process integrity is crucial in adapting to this technological shift.

3 ChatGPT EXPERIMENT

The experimental design we implemented comprises a systematic evaluation of ChatGPT’s ability to generate programming solutions. To commence the experiment, we acquired a comprehensive set of practice exercises from computer programming textbooks, covering a wide range of elementary programming topics. These exercises served as the foundational elements for evaluating ChatGPT’s performance. Our methodology involved inputting these exercises into ChatGPT, guiding the model to

generate solution programs for each exercise in single or multiple prominent programming languages: C/C++, Java, and Python. This multilingual evaluation was intentionally designed to assess the model’s proficiency across varying programming paradigms. After generating the solution programs, we performed an exhaustive analysis to assess their correctness. This scrutiny specifically targeted the functional accuracy of the solution programs, evaluating how well they aligned with the specified requirements of the exercises. The findings were subsequently presented to a focus group comprising experienced programming teachers. Leveraging their extensive expertise, we facilitated a thorough discussion to gather valuable insights and recommendations. This discourse, extending beyond the evaluation of ChatGPT’s performance, included proposals for refining the design of programming problems and suggesting best practices for addressing assessment challenges. In the subsequent subsections, we delve into the specifics of our experiment design.

3.1 Exercise Acquisition

The exercises sourced from textbooks cover various topics. Our selection criteria were twofold: exercises had to produce tangible program source code outcomes, and these outcomes needed to be assessable (Tang, Yu, & Poon, 2023). Our staff members, drawing on collective experience and professional judgment, meticulously curated the exercises to ensure their suitability and alignment with specific learning outcomes (Biggs, 2003).

We carefully selected 50 programming exercises from reputable textbooks (Deitel & Deitel, 2012 & 2016; Heathcote, 2017; Malik, 2013) with an emphasis on diversity. Within this set, 30 exercises presented concise problem descriptions, each focused on a single programming problem within specific topics. Covering fundamental concepts in elementary programming courses, these exercises were evenly distributed across four categories: *fundamentals* (e.g., variables, data types, operators, and expressions), *control structures* (e.g., loops and conditionals), *data structures* (e.g., arrays, lists, stacks, and queues), and *algorithms* (e.g., searching, sorting, and recursion). Importantly, these categories were language-independent, allowing solutions in C/C++ (referred to as C++ hereafter for simplicity), Java, and Python. Each category comprised 5 exercises, totaling 20 exercises and 60 program solutions across three languages. We also introduced two language-specific categories: *low-level memory manipulation* (e.g., manual memory allocation/deallocation, pointers,

value and reference semantics) exclusively designed for C++, and *Object-Oriented structures* (e.g., classes, objects, inheritance, polymorphism, and encapsulation) tailored for both C++ and Java. Each language-specific category included 5 exercises, contributing to a total of 10 exercises.

In addition to exercises with concise descriptions, we introduced 10 exercises with more extensive problem statements, addressing multiple problems across various topics, primarily in the programming languages of C++ and Java.

Beyond the aforementioned code-from-scratch exercises, we included 10 exercises where the source code was provided, requiring students to modify the existing code. Notably, the source code for these exercises was written in C++, chosen due to its availability in our textbook collection.

Table 1: A summary of the selected exercises.

Exercise Type / Topic	Prog. Lang.	Ex. Qty.	Result. Progs.
Short exercises			
– <i>Programming basics</i>	C++ &	5	15
– <i>Control structures</i>	Java &	5	15
– <i>Data structures</i>	Python	5	15
– <i>Algorithms</i>		5	15
– <i>Memory manipulation</i>	C++	5	5
– <i>O-O structures</i>	C++/Java	5	5
Long exercises			
	C++	8	8
	Java	2	2
Modifying existing code	C++	10	10
	Total	50	90

In summary, our exercise sampling strategy underwent a thorough selection process, guided by criteria ensuring assessability and alignment with learning outcomes. The outcome is a diverse collection of exercises designed to address various programming concepts and languages. An overview of the selected exercises, detailing topics, languages, quantities for exercises and resulting programs, can be found in Table 1, with comprehensive details for each exercise included in Appendix.

3.2 Program Solution Generation

In the process of soliciting solutions from ChatGPT for programming problems, we presented the questions as chat queries. The specific version of ChatGPT employed in this experiment was the latest GPT-3.5. It's worth noting that we consider the results independent of the browser and platform used.

To preserve the authenticity of the exercises, we directly incorporated most of them from the original textbooks. Our modifications were limited to

essential changes, like adjusting references to additional information. For instance, we altered phrases such as “modify the program in Figure X...” to “modify the following program...” and “use the provided function in Example Y...” to “use the provided function below...,” ensuring clarity while maintaining the core content.

To guide ChatGPT in generating solutions in various programming languages, we incorporated specific instructions within the exercises. Language directives such as “in C” or “Write a [Python] program” were inserted to prompt the model’s language-specific responses. For example, the prompt “Write a Java program to calculate and print a list of all prime numbers from 1 to 100” indicates the addition of the programming language “Java” to prompt the generation of a Java program. Although ChatGPT has the capability to generate new responses if the user is not satisfied, our experiment specifically focused on analysing only the initial response produced by ChatGPT.

Following the generation of responses (solutions), we documented and presented them to two programming teachers for evaluation. The assessment process involved visually inspecting the source code and, in certain cases, executing the programs for testing. The assessment outcomes were categorized as “correct” if the program met all requirements, “partially correct” if it fulfilled major requirements but wasn’t entirely correct, or “incorrect” if it failed to satisfy major requirements. The judgments were based on the professional expertise of the staff. In cases where there was a discrepancy in the assessment results between the two assessors, a moderation process was implemented to reconcile and establish a consensus.

3.3 Results

Our assessment results reveal that ChatGPT demonstrates a notably high proficiency in generating accurate program solutions, surpassing our initial expectations. While we expected effectiveness in simpler exercises, our findings reveal its competence extends to more complex tasks.

Table 2 provides a concise summary of the assessment results. The first column categorizes exercises and topics. In the second column, the number of solutions under assessment is displayed, while the third to fifth columns detail the assessment outcomes. Specifically, the column marked with “✓” signifies the count of correct solutions, the column marked with “○” indicates the count of partially correct solutions, and the column marked with “×”

represents the count of incorrect solutions. For detailed assessment outcomes of each individual exercise, refer to the Appendix.

Table 2: Assessment results for the generated solutions.

Exercise Type / Topic	Assessed	✓	○	✗
Short exercises				
– Programming basics	15	15	0	0
– Control structures	15	15	0	0
– Data structures	15	15	0	0
– Algorithms	15	15	0	0
– Memory manipulation	5	5	0	0
– O-O structures	5	5	0	0
Long exercises	10	6	4	0
Modifying existing code	10	7	3	0
Total	90	83	7	0

Note: ✓ = correct; ○ = partially correct; ✗ = incorrect.

Remarkably, for all short exercises, ChatGPT consistently produced correct solutions across all programming languages. This suggests its capability to generate solutions spans various programming topics and languages.

For longer questions, ChatGPT achieved correct solutions for 6 out of 10 exercises and partially correct solutions for the remaining 4 exercises, without generating any outright incorrect solutions. This prompts an exploration into factors contributing to ChatGPT’s generation of partially correct solutions, revealing potential implications for refining exercise design to pose challenges for the model. Two identified causes include ChatGPT’s tendency to provide a “best solution” based on its knowledge, potentially overlooking nuanced details in complex exercise requirements. Additionally, the absence of prior knowledge to exercise requirements poses a challenge, as some exercises rely on applying knowledge gained in previous exercises or examples. ChatGPT, lacking such context initially, responds based on its existing knowledge, which may not align with exercise expectations.

On the positive note, to rectify solutions that are not entirely correct, students are required to engage in a critical evaluation of AI-generated solutions. They need to discern the factors contributing to the inability to generate a completely correct solution and subsequently provide additional or rectify missing information to guide ChatGPT in producing accurate solutions. This iterative process not only prompts students to adopt a bug-fixing approach but also cultivates higher-order thinking skills, representing a substantial educational benefit (Krauthohl, 2002).

In conclusion, although ChatGPT exhibits impressive capabilities in generating accurate programs, our study highlights opportunities to

enhance exercises for better challenging the model. Moreover, it emphasizes the pedagogical value for students in critically evaluating and refining AI-generated solutions.

4 DISCUSSIONS

The findings of our experiment were meticulously presented to a focus group comprising four lecturers and senior lecturers experienced in teaching computer programming courses. The objective of this comprehensive deliberation was to rigorously analyse the outcomes of the experiment and explore their potential implications on academic integrity and assessment strategies within the field of computer programming education.

To foster a purposeful exchange of ideas, we initiated the discussion with structured questions encompassing three key aspects:

1. Perception and Awareness:
 - How do you perceive the role of AI, specifically ChatGPT, in generating correct solutions for programming exercises?
2. Challenges to Academic Integrity:
 - In your opinion, what challenges does the capability of AI in generating correct solutions pose to academic integrity in computer programming courses?
3. Impact on Assessment Strategies:
 - How might the use of AI in generating correct solutions influence your current assessment strategies for programming assignments?

Due to space constraints, the fully transcribed discussion summary, guided by the above questions, couldn’t be fully included in this paper. Here are the concise conclusions that emerged:

Participants unanimously agreed that AI introduces challenges in discerning the authenticity of program solutions. The impressive proficiency of ChatGPT in generating solutions for intricate exercises highlights the difficulty in crafting exercises meant to challenge AI-generated solutions. Nevertheless, the excessively “perfect” nature of AI-generated solutions, particularly those utilizing advanced practices and features, may serve as a distinguishing factor from those of average elementary programmers.

Acknowledging the proficiency of AI in generating accurate solutions, participants expressed heightened concerns about the potential exacerbation of academic misconduct. The dependency on AI inhibits the cultivation of essential problem-solving skills, resulting in disparities in skill development

among students. Assessments face accuracy challenges, making it difficult to assess students' genuine understanding and comprehension of underlying concepts, thereby biasing the feedback loop in facilitating effective learning. The widespread use of AI may erode trust between teachers and students, ultimately impacting the authenticity of individual efforts.

In response to these challenges, participants reached a consensus on the imperative to reassess exercise design and assessment strategies to address potential risks associated with the misuse of AI. This shift in assessment focus emphasizes the need for a deeper understanding of the problem-solving process, necessitating comprehensive documentation and explanation of students' approaches. The inclusion of coding interviews or presentations becomes crucial, providing educators with opportunities to scrutinize students' genuine understanding of their solutions. This dynamic approach ensures assessments not only gauge correctness but also delve into the intricacies of the process involved in arriving at a solution, fostering active learning, and contributing to assessment for learning.

Participants collectively acknowledged ChatGPT as a robust tool with the inherent potential to enhance the learning experience. For example, students can utilize ChatGPT for immediate clarification on programming concepts, syntax, and problem-solving approaches. Additionally, ChatGPT's availability round the clock ensures students have continuous access to assistance and information as needed. These aspects echo the findings of Rahman & Watanobe (2023) regarding threats and opportunities, highlighting the potential for further exploration.

5 RECOMMENDATIONS

Based on in-depth focus group discussions, we offer concise recommendations specifically centered on exercise design practices and assessment strategies. Broader topics such as curriculum design, policy, and ethics are reserved for future exploration.

5.1 Exercise Design Practices

The focus group has outlined four essential exercise design practices. First and foremost, they recommend the creation of unique and open-ended problems. This involves developing tailored programming exercises that transcend generic responses by incorporating real-world scenarios or specific requirements. Utilizing open-ended questions prompts students to

provide comprehensive explanations and reasoning alongside their code submissions. This approach actively discourages reliance on AI code generation and fosters a deeper understanding of programming concepts, encouraging students to articulate the intricate thought processes behind their code.

Secondly, they recommend adopting Test-Driven Development (TDD) (Janzen & Saiedian, 2005). This method prompts students to write tests before coding, fostering critical thinking in problem-solving. TDD guides students to address smaller problems first, ensuring code correctness and cultivating a deep understanding of the problem domain. This approach nurtures an analytical problem-solving mindset, steering students away from predetermined solutions and reducing reliance on AI assistance.

The third recommendation involves the use of versioned assignments. They suggest mandating the use of version control systems, such as Git, particularly for larger assignments. This entails requiring students to submit their work incrementally and commit their source code at different stages of development. The purpose is to promote continuous engagement with the artifact and make it challenging for students to rely solely on AI-generated solutions for the entirety of the assignment.

Lastly, peer collaboration is encouraged as a key exercise design practice. This entails fostering collaborative learning through group assignments in pairs or small groups while maintaining a focus on individual accountability. Such an environment facilitates active discussions and the sharing of ideas, code, and problem-solving approaches among students. Beyond enriching the learning experience, this collaborative approach acts as a deterrent to copying solutions, cultivating a shared understanding and responsibility among students.

5.2 Assessment Strategies

In addition to exercise design practices, the focus group has introduced assessment strategies tailored to the integration of AI. Foremost among these strategies is the endorsement of live coding assessments. This approach requires students to actively solve programming problems during their lab or practical classes, thereby showcasing their coding proficiency and problem-solving skills to their programming teachers. The inherent nature of live assessments renders them inherently less susceptible to plagiarism and AI assistance, particularly within a closely monitored environment. This format also serves as an optimal platform for teachers to provide

immediate feedback, facilitating assessment for learning efficiently.

Building upon the efficacy of live coding assessments, the next recommended strategy involves the incorporation of viva voce presentations. Verbal explanations necessitate a profound understanding of students' programs. Even when presented with an AI-generated solution, students must delve into an intensive examination of the code to comprehend the underlying operations and theories. This exemplifies how AI can serve as a powerful learning tool. Simultaneously, viva voce assessments empower teachers to accurately measure the depth of a student's comprehension and evaluate their ability to articulate and elucidate their thought processes.

In addition to the above strategies, we advocate for the implementation of portfolio-based assessments. Students are encouraged to curate and maintain a coding portfolio throughout the course, with assessments grounded in the progression and improvement demonstrated within this evolving portfolio. This multifaceted strategy adds an additional layer of complexity, creating a more challenging environment for students to rely on external AI-generated solutions. This approach aligns with our commitment to ensuring assessments authentically reflect the unique learning journeys of individual students.

Collectively, these practices and strategies are designed to foster genuine assessment for learning, discourage plagiarism, and ensure that assessments accurately reflect the skills and understanding acquired by students in programming courses.

6 CONCLUSIONS

The recent decades have witnessed a significant leap in computer technology, with AI emerging as transformative forces that permeate our daily lives. This relentless progression signifies not merely a passing trend but an irreversible evolution, poised to reshape educational landscapes and assume a pivotal role in the realm of teaching and learning.

While the integration of AI introduces a conundrum of challenges, particularly in preserving academic integrity, it also holds the promise of enhancing educational experiences. This study, delving into the success of AI in generating precise solutions for programming exercises and assignments, underscores the undeniable influence of these technologies on academic pursuits. The challenge lies not just in acknowledging this

influence but in navigating its intricacies and mitigating potential pitfalls.

The insights gleaned from our focus group discussions have culminated in a robust set of recommendations designed to address the nuances of AI-generated work. These guidelines serve as a proactive response, offering strategies to maintain academic integrity while harnessing the potential of AI in educational settings.

Embracing the power of AI, rather than resisting it, can herald a new era in education. By leveraging the capabilities of these technologies, we can sculpt an educational landscape that is not just adaptive but transformative. The journey ahead involves a delicate balance—navigating challenges while harnessing the boundless potential of AI to foster a future of enriched and innovative education. As we step into this transformative era, it becomes imperative to stay proactive, ensuring that AI becomes an ally in the educational journey rather than a hindrance.

REFERENCES

- Ala-Mutka, K.M.(2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2), 83-102.
- Baidoo-Anu, D., & Ansah, L. O. (2023). Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning. *J. of AI*, 7(1), 52-62.
- Biggs, J. (2003). Aligning teaching for constructing learning. *Higher Education Academy*, 1(4), 1-4.
- Cotton, D. R., Cotton, P. A., & Shipway, J. R. (2023). Chatting and cheating: Ensuring academic integrity in the era of ChatGPT. *Innovations in Education and Teaching International*, 1-12.
- Deitel, P. & Deitel, H. (2012). *Java how to program*, Pearson. 9th Edition.
- Deitel, P. & Deitel, H. (2016). *C How to program with an introduction to C++*, Pearson, 8th Edition.
- Deng, J., & Lin, Y. (2022). The benefits and challenges of ChatGPT: An overview. *Frontiers in Computing and Intelligent Systems*, 2(2), 81-83.
- Gao, C. A., Howard, F. M., Markov, N. S., Dyer, E. C., Ramesh, S., Luo, Y., & Pearson, A. T. (2022). Comparing scientific abstracts generated by ChatGPT to original abstracts using an artificial intelligence output detector, plagiarism detector, and blinded human reviewers. *BioRxiv*, 2022-12.
- Heathcote, P. M. (2017). *Learning to program in Python*, PG Online Limited, 1st Edition.
- Hong, W. C. H. (2023). The impact of ChatGPT on foreign language teaching and learning: opportunities in education and research. *Journal of Educational Technology and Innovation*, 5(1).

Janzen, D., & Saiedian, H. (2005). Test-driven development concepts, taxonomy, and future direction. *Computer*, 38(9), 43-50.

Kalla, D., & Smith, N. (2023). Study and analysis of ChatGPT and its impact on different fields of study. *International Journal of Innovative Science and Research Technology*, 8(3).

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory into practice*, 41(4), 212-218.

Lee, V. C., Yu, Y. T., Tang, C. M., Wong, T. L., & Poon, C. K. (2018). ViDA: A virtual debugging advisor for supporting learning in computer programming courses. *Journal of Computer Assisted Learning*, 34(3), 243-258.

Lo, C. K. (2023). What is the impact of ChatGPT on education? A rapid review of the literature. *Education Sciences*, 13(4), 410.

Malik, D. S. (2013). *C++ programming: from problem analysis to program design*, Cengage Learning, 6th Ed.

Meyer, J.G., Urbanowicz, R.J., Martin, P.C., O'Connor, K., Li, R., Peng, P.C., ... & Moore, J.H. (2023). ChatGPT and large language models in academia: opportunities and challenges. *BioData Mining*, 16(1), 20.

Ng, S. C., Li, T. S., & Ngai, H. S. (2004). *Plagiarism detection in programming assignments*, in Murphy, D., Carr, R., Taylor, J., & Wong, T. M. (Eds), Distance Education and Technology: Issues and Practice, Open University of Hong Kong Press, 2004, pp. 366-377. ISBN: 962-7707-47-3

Rahman, M. M., & Watanobe, Y. (2023). ChatGPT for education and research: Opportunities, threats, and strategies. *Applied Sciences*, 13(9), 5783.

Rasul, T., Nair, S., Kalendra, D., Robin, M., de Oliveira Santini, F., Ladeira, W. J., ... & Heathcote, L. (2023). The role of ChatGPT in higher education: Benefits, challenges, and future research directions. *Journal of Applied Learning and Teaching*, 6(1).

Surameery, N. M. S., & Shakor, M. Y. (2023). Use ChatGPT to solve programming bugs. *International Journal of Information Technology & Computer Engineering (IJITC) ISSN: 2455-5290*, 3(01), 17-22.

Tang, C. M., Yu, Y. T., & Poon, C. K. (2009). An approach towards automatic testing of student programs using token patterns. In *17th International Conference on Computers in Education, ICCE 2009* (pp. 188-190).

Tang, C. M., Yu, Y. T., & Poon, C. K. (2023). An automated system with a versatile test oracle for assessing student programs. *Computer Applications in Engineering Education*, 31(1), 176-199.

Turnitin. (2023, November 11). *Turnitin's AI detector capabilities*. <https://www.turnitin.com/solutions/topics/ai-writing/ai-detector/>

Ventayen, R. J. M. (2023). OpenAI ChatGPT generated results: Similarity index of artificial intelligence-based contents. *Available at SSRN 4332664*.

Yu, Y. T., Poon, C. K., & Choy, M. (2006, October). Experiences with PASS: Developing and using a programming assignment assessment system. In *2006 Sixth International Conference on Quality Software (QSIC'06)* (pp. 360-368). IEEE.

APPENDIX

A list of selected exercises from textbooks.

Ex. No.	Exercise Description	Text-book [†]	Page	Result [*]
Short exercise				
<i>Topic: Fundamentals</i>				
2.23	Largest and Smallest Integers	C1	98	✓
10	Average score	C2	118	✓
2.15	Arithmetic	J	102	✓
2.18	Displaying Shapes with Asterisks	J	103	✓
2	Circle's area and circumference	P	16	✓
<i>Topic: Control structures</i>				
3.34	Floyd's Triangle	C1	142	✓
4.12	Prime Numbers	C1	183	✓
6	Sum of evens and odds	C2	330	✓
4.18	Credit Limits Calculator	J	182	✓
4.22	Tabular Output	J	183	✓
<i>Topic: Data structures</i>				
6.14	Union of Sets	C1	298	✓
12.10	Reversing the Words of a Sentence	C1	544	✓
4	Counting scores in ranges	C2	585	✓
22.11	Palindrome Tester	J	970	✓
2	Student List	P	36	✓
<i>Topic: Algorithms</i>				
5.39	Recursive Greatest Common Divisor	C1	241	✓
5.41	Recursive Prime	C1	242	✓
18.17	Print an Array Backward	J	832	✓
19.5	Bubble Sort	J	862	✓
1	Anagram	P	49	✓
<i>Topic: Memory manipulation</i>				
12.8	Inserting into an Ordered List	C1	544	✓
12.19	Depth of a Binary Tree	C1	547	✓
12.20	Recursively Print a List Backward	C1	548	✓
13	Circular linked lists	C2	1148	✓
14.6	Dynamic Array Allocation	C1	580	✓
<i>Topic: Object-oriented structures</i>				
19.10	Account Inheritance Hierarchy	C1	798	✓
20.16	CarbonFootprint Abstract Class: Polymorphism	C1	843	✓
23.3	Operator Overloads in Templates	C1	917	✓
9.8	Quadrilateral Inheritance Hierarchy	J	429	✓
21.7	Generic isEqualTo Method	J	939	✓
Long exercises				
3.16	Mortgage Calculator	C1	137	✓
5.36	Towers of Hanoi	C1	240	✓
7.17	Simulation: The Tortoise and the Hare	C1	353	○
7.22	Maze Traversal	C1	356	✓
7.32	Polling	C1	363	✓
12.12	Infix-to-Postfix Converter	C1	544	○
15	Memory Game	C2	588	✓
16	Airplane Seating Assignment	C2	589	○
16.5	Random Sentences	J	748	✓
22.26	Insert/Delete Anywhere in a Linked List	J	975	○
Modifying existing code				
3.27	Validating User Input	C1	140	✓
4.15	Modified Compound-Interest Program	C1	183	✓
4.22	Average Grade	C1	185	✓
5.20	Displaying a Rectangle of Any Character	C1	238	✓
5.33	Guess the Number Modification	C1	240	✓
6.32	Linear Search	C1	305	✓
12.16	Allowing Duplicates in a Binary Tree	C1	547	○
12.23	Level Order Binary Tree Traversal	C1	548	○
16.11	Modifying Class GradeBook	C1	657	○
17.13	Enhancing Class Rectangle	C1	711	✓

[†]Textbooks: C1: Deitel & Deitel (2016); C2: Malik (2013); J: Deitel & Deitel (2012); P: Heathcote (2017).

^{*}Result: ✓ = correct; ○ = partially correct; × = incorrect