

PROGRAMMING with AI

For this assessment, you can use Generative AI to create code for submission.

Transparency is required.

Where generative AI is used to produce code for assessed work, you must be specific about what was produced by the AI, what changes you made to it, and what you wrote yourself. You should do this using inline comments in the code, using the appropriate notation for the programming language you are using. Just like using code from other external sources (textbooks, GitHub repositories, stack overflow etc.), there should be no doubt in the marker's mind which parts of the work were completed by you and which parts were produced by generative AI.

Lecturers may give more detailed guidance.

In some cases, lecturers may permit you to use generative AI to solve more complex problems. They will communicate with you if this is the case. This document should serve as a default position, which lecturers may supersede. If you are ever unclear about what is and is not acceptable for a given assignment, ask your module lecturer for specific help. You may find that early in your programming education, you may not be able to make good use of generative AI because the problems you are solving are correspondingly simple and may only be trivialised using AI. As you progress, you will solve larger problems and it will become more appropriate to use generative AI to produce boilerplate code or to solve smaller sub-problems.

Generative AI within Integrated Development Environments

This guidance applies equally to query-driven generative AI such as ChatGPT and Bard, as it does to “autocomplete” generative AI within an integrated development environment (IDE), such as GitHub co-pilot, Tabnine or IntelliCode. In either instance, these tools should be used judiciously and with caution. Violating this policy inadvertently using an autocomplete generative AI or similar tool still constitutes academic misconduct.

What you cannot do

You **cannot and must not** simply prompt an AI tool to write code on the topic of your assignment.

If we assume that your task is to create an OpenGL graphics engine in C++.

You **cannot and must not** input a prompt like this to an AI tool:

- Write an OpenGL graphics engine in C++.
- Give examples of OpenGL graphics engines in C++ with an explanation.

If you do this, then the code is no longer yours. If you submit work that is not your own, then you will be committing **academic misconduct**, which carries severe penalties.

You will find it difficult to use generative AI for very simple problems without trivialising that problem and risking committing academic misconduct.

What you can do: Ask questions to explore the problem

You can ask generative AI specific questions which inform the development of your code, or aid in your understanding of the topic. For example:

- Why is it bad to block the event loop?
- Why do GPUs work so effectively with triangles?
- How do I multiply a 4x4 matrix?
- How do I calculate the determinant of a matrix?

These sorts of questions are always acceptable because they do not produce any code but explain concepts which will inform your development. These sorts of questions will usually not yield answers beyond the capabilities of a search engine but may allow a more probing contextual discussion.

What you can do: Snippets

Aligned with common programming practices, it may be appropriate to prompt generative AI for small snippets of code. In the previous example, this might be to generate boilerplate, e.g.:

- Write a C++ class with three integer attributes: x, y and z. Create getters and setters for them.

However, within the scope of the example, a prompt like this would **not be acceptable**:

- Write a 3D vector class in C++ including dot and cross product.

It is appropriate to use generative AI to create boilerplate code, or to generate solutions to trivial sub-problems. You should avoid using generative AI to trivialise significant components of the work you have been required to undertake.

This may, at times, be a difficult distinction to make, you should always ask your lecturer if unsure.

What you can do: Debugging

Several AI debugging tools are now available, capable of detecting errors in code and making suggestions for their remedy. These tools are broadly comparable with static code analysis tools, and are permitted – however you must again be transparent about their use.

Example

Combining the above, if we wanted to create a vector class as a component of our graphics engine, we could prompt our generative AI engine:

```
> Write a C++ class with three integer attributes: x, y and z. Create getters and setters for them.
```

The generative AI tool responds:

```
Yes, here is a C++ class with three integer attributes: x, y, and z, and getters and setters for them:
```

```
C++
```

```
#include <iostream>
```

```

class Point {
private:
    int x;
    int y;
    int z;

public:
    Point(int x, int y, int z) {
        this->x = x;
        this->y = y;
        this->z = z;
    }

    int GetX() const {
        return x;
    }

    void SetX(int x) {
        this->x = x;
    }

    int GetY() const {
        return y;
    }

    void SetY(int y) {
        this->y = y;
    }

    int GetZ() const {
        return z;
    }

    void SetZ(int z) {
        this->z = z;
    }

    void Print() const {
        std::cout << "Point: (" << x << ", " << y << ", " << z << ")" << std::endl;
    }
};

int main() {
    Point point(1, 2, 3);
    point.Print();

    point.SetX(4);
    point.SetY(5);
    point.SetZ(6);
}

```

```

    point.Print();

    return 0;
}

```

Use code with caution. [Learn more](#)

This code will create a Point object with coordinates (1, 2, 3), print it to the console, and then change the coordinates to (4, 5, 6) and print it again.

We should now indicate that this is not our original work by adding an inline comment at the top of the file. In this case:

```

//The boilerplate for this class was created through generative AI (Bard).
Specifically, the class structure, x, y and z attributes, their getters and setters, the
constructor and the print method were produced in this way.

```

Having trivialised some of the boilerplate code with generative AI, we can then manually add the more advanced functionality ourselves. In this case, I will rename the class, separate the header and add a dot and cross product method. I could write code such as:

```

Vector Vector::CrossProduct(Vector sV)
{
    Vector nV;
    nV.SetX(GetY() * sV.GetZ() - GetZ() * sV.GetY());
    nV.SetY(GetZ() * sV.GetX() - GetX() * sV.GetZ());
    nV.SetZ(GetX() * sV.GetY() - GetY() * sV.GetX());
    return nV;
}

int Vector::DotProduct(Vector sV)
{
    return GetX() * sV.GetX() + GetY() * sV.GetY() + GetZ() * sV.GetZ();
}

```

Then update the comment:

```

//The boilerplate for this class was created through generative AI (Bard).
Specifically, the class structure, x, y and z attributes, their getters and setters, the
constructor and the print method were produced in this way.
//The dot and cross product methods were created from scratch without the use of
generative AI.

```

Giving a final implementation of this class as:

```
Vector.h
```

```
#pragma once
```

//The boilerplate for this class was created through generative AI (Bard). Specifically, the class structure, x, y and z attributes, their getters and setters, the constructor and the print method were produced in this way.

//The dot and cross product methods were created from scratch without the use of generative AI.

```
#include <iostream>
```

```
class Vector
{
private:
    int x;
    int y;
    int z;

public:
    Vector();
    Vector(int x, int y, int z);

    int GetX() const;
    int GetY() const;
    int GetZ() const;

    void SetX(int x);
    void SetY(int y);
    void SetZ(int z);

    void Print() const;

    Vector CrossProduct(Vector sV);
    int DotProduct(Vector sV);
};
```

Vector.cpp

```
#include "Vector.h"

Vector::Vector(){
    this->x = 0;
    this->y = 0;
    this->z = 0;
}

Vector::Vector(int x, int y, int z) {
    this->x = x;
    this->y = y;
    this->z = z;
}
```

```

int Vector::GetX() const {
    return x;
}

void Vector::SetX(int x) {
    this->x = x;
}

int Vector::GetY() const {
    return y;
}

void Vector::SetY(int y) {
    this->y = y;
}

int Vector::GetZ() const {
    return z;
}

void Vector::SetZ(int z) {
    this->z = z;
}

void Vector::Print() const {
    std::cout << "Point: (" << x << ", " << y << ", " << z << ")" << std::endl;
}

Vector Vector::CrossProduct(Vector sV) {
    Vector nV;
    nV.SetX(GetY() * sV.GetZ() - GetZ() * sV.GetY());
    nV.SetY(GetZ() * sV.GetX() - GetX() * sV.GetZ());
    nV.SetZ(GetX() * sV.GetY() - GetY() * sV.GetX());
    return nV;
}

int Vector::DotProduct(Vector sV) {
    return GetX() * sV.GetX() + GetY() * sV.GetY() + GetZ() * sV.GetZ();
}

```