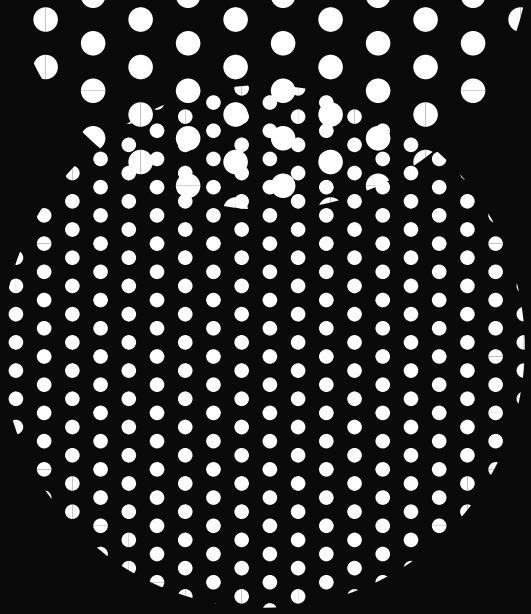

Macros and Inline Function

* We use them when sequence of steps is small
* To eliminate time overhead.

⇒ Macros:

We need # define compiler directive and it is used for text substitution.

Syntax: # define square(x) (x*x)
printf("%d", square(5)); expands to 5*5,

Advantage	disadvantage
It is fast and quick text substitution.	No type checking.

⇒ Inline functions

It is defined using inline keyword and is used to optimize function calls by the compiler

Syntax: inline int square (int x) { return x*x};
printf("%d", square (5); // Maybe optimized by the compiler

Advantage	disadvantage
Type safety, debuggable, no side effects	The compiler might ignore the inline suggestion

* Both aim to reduce function call overhead but differ in implementation.

Feature	Macro	Inline Function
Definition	Preprocessor directive (`#define`)	Function definition (`inline` keyword)
Type Safety	No type checking	Type-safe, function-like behavior
Compilation	Handled by preprocessor	Handled by the compiler
Debugging	Hard to debug (text substitution)	Easier to debug (like normal functions)
Side Effects	Possible due to multiple evaluations	No side effects, behaves like a function
Code Bloat	Can cause code duplication	Compiler optimizes inline code

Macros

Definition: Macros are preprocessor directives in C/C++ that define a set of code that can be reused. They are typically defined using the `#define` directive.

Advantages:
1. **Performance**: Since macros are expanded inline during preprocessing, there is no function call overhead.
2. **Code Reusability**: Allows you to write a piece of code once and use it multiple times.
3. **Conditional Compilation**: Macros can be used to include or exclude parts of code based on certain conditions.

Disadvantages:
1. **Debugging Difficulty**: Errors in macros can be hard to trace, as the code expanded from macros may not show the original macro definition.
2. **No Type Checking**: Macros do not check types, which can lead to unexpected behavior.
3. **Side Effects**: If a macro is defined with arguments that have side effects (like function calls), it can lead to unexpected results when the macro is expanded.

Inline Functions

Definition: Inline functions are functions that are defined with the `inline` keyword in C/C++. They suggest to the compiler to insert the function's code at each point the function is called, instead of performing a traditional call.

Advantages:
1. **Type Safety**: Inline functions provide type checking, helping to avoid errors that macros might introduce.
2. **Debugging Ease**: Errors in inline functions can be traced back to their definitions, making debugging easier.
3. **Scoped Variables**: Inline functions can have local variables, avoiding potential conflicts seen with macros.

Disadvantages:
1. **Code Bloat**: If an inline function is used many times, it can increase the size of the binary since the code is duplicated.
2. **Compiler Discretion**: The compiler may ignore the inline request and treat it like a regular function, especially if the function is complex or large.
3. **Overhead for Small Functions**: For very small functions, the overhead of making the call can sometimes outweigh the benefits of inlining.

Summary

Feature	Macros	Inline Functions
Definition	Preprocessor directives for code reuse.	Functions defined with the inline keyword.
Advantages	Performance, code reusability, conditional compilation.	Type safety, debugging ease, scoped variables.
Disadvantages	Debugging difficulty, no type checking, potential side effects.	Code bloat, compiler discretion, overhead for small functions.