Macro Parameters

In the context of macro processing, especially in programming languages that support macro systems, various types of parameters can be used to customize the behavior of macros. Here are four common types:

1. **Positional Parameters:**
    - These are parameters that are identified by their position in the parameter list. The order in which parameters are passed matters.
    - Example: In a macro definition, you might have something like `MACRO(name, age, city)` where `name`, `age`, and `city` are positional parameters.
2. **Keyword Parameters:**
    - These are parameters identified by keywords or names rather than their position. This allows you to pass values in any order, and it is clear which value corresponds to which parameter.
    - Example: `MACRO(name: "John", age: 25, city: "New York")` where `name`, `age`, and `city` are keyword parameters.
3. **Mixed Parameters:**
    - This is a combination of positional and keyword parameters. Some parameters are identified by their position, and others are identified by keywords.
    - Example: `MACRO(name, age: 25, city)` where `name` and `city` are positional parameters, and `age` is a keyword parameter.
4. **Default Value:**
    - Default values are assigned to parameters in case the caller doesn't provide a value for that parameter. This is useful for making certain parameters optional.
    - Example: `MACRO(name, age = 25, city = "DefaultCity")` where `name` is required, but `age` and `city` have default values.

Scheduling algorithms

Here are the advantages and disadvantages of several common scheduling algorithms:

1. **First-Come-First-Serve (FCFS):**

    Advantages:

    - Simple and easy to understand.
    - No starvation, as processes are executed in the order they arrive.

    Disadvantages:

- Can lead to the "convoy effect" where short processes get stuck behind long processes.
- Poor turnaround time and waiting time, especially if long processes arrive first.

2. **Round Robin:**

   Advantages:

   - Fairness in terms of CPU allocation to processes.
   - Suitable for time-sharing systems where multiple users need access to the CPU.

   Disadvantages:

   - High turnaround time for processes with high burst times.
   - Performance can degrade if the time quantum is too large or too small.

3. **Shortest Job First (SJF):**

   Advantages:

   - Minimizes average waiting time and turnaround time.
   - Optimal for minimizing the total processing time.

   Disadvantages:

   - Requires knowledge of the burst time in advance, which may not be available.
   - Can lead to starvation for longer processes if shorter ones keep arriving.

4. **Shortest Remaining Time First (SRTF):**

   Advantages:

   - Provides improved response time and throughput compared to SJF.
   - Dynamic adaptation to changing burst times during execution.

   Disadvantages:

   - May lead to frequent preemptions, increasing context switch overhead.
   - Can result in starvation for longer processes.

5. **Priority Scheduling:**

   Advantages:

   - Allows prioritization of important or time-sensitive tasks.
   - Can be adapted for different system requirements.

   Disadvantages:

   - May lead to starvation of lower priority processes.
   - The potential for priority inversion, where a lower-priority process holds a resource needed by a higher-priority process.

THEORY:

CPU scheduling refers to a set of policies and mechanisms built into the operating systems

that govern the order in which the work to be done by a computer system is completed..

What is scheduler?

1. Scheduler in an OS module that selects the next job to be admitted into the system and the next process to run.

2. Primary objective of the scheduler is to optimize system performance in accordance with the criteria deemed by the system designers. In short, scheduler is that module of OS which schedules the programs in an efficient manner.

Types of schedulers

In general, there are three different types of schedulers which may co-exist in a complex operating system.

• Long term scheduler

• Medium term scheduler

• Short term scheduler.

In operating systems, particularly in the context of process management and scheduling, the terms "long-term scheduler," "medium-term scheduler," and "short-term scheduler" refer to different phases of the process lifecycle and scheduling. Each plays a specific role in managing processes within the system.

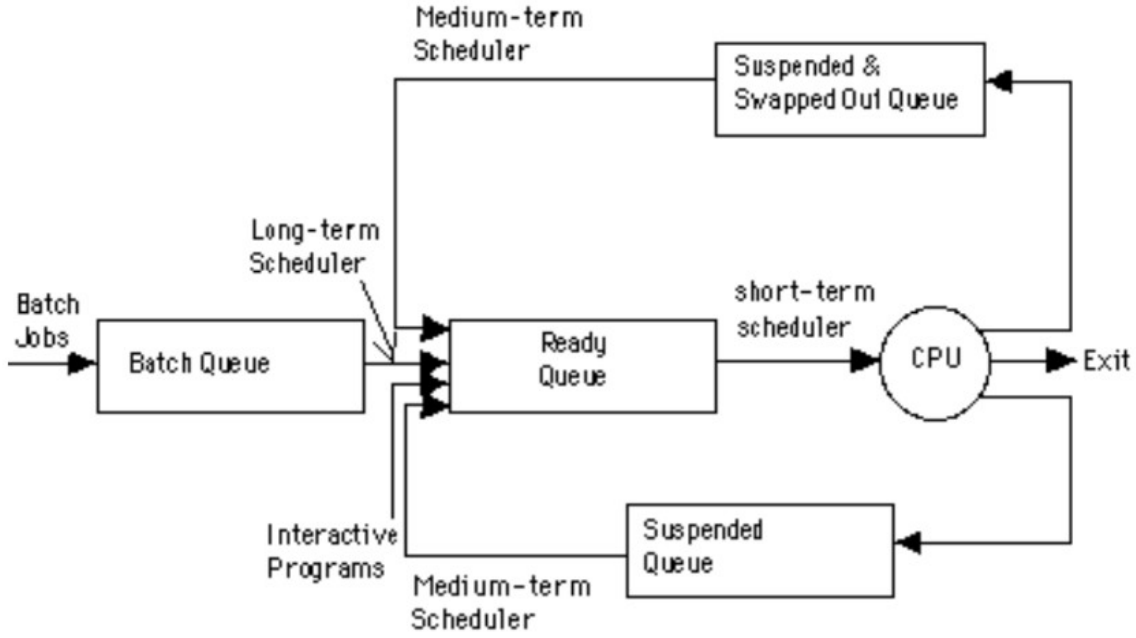
1. **Long-Term Scheduler (Job Scheduler):**
    - **Function:**
        - Selects processes from the job pool (waiting processes in the job queue) and loads them into the main memory.
        - Determines which processes should be brought into the ready queue for execution.
    - **Frequency:**
        - Executes infrequently (seconds or minutes).
    - **Objective:**
        - Maintain a good mix of processes in the system, balancing resource utilization and throughput.
    - **Key Considerations:**
        - Degree of multiprogramming.
        - System load.
        - Resource availability.
2. **Medium-Term Scheduler (Swapper):**

- **Function:**
    - Manages processes that are partially or fully in the main memory.
    - Decides which processes to swap in and out of the main memory.
    - Performs swapping (transfer of processes between main memory and secondary storage).
- **Frequency:**
    - Executes relatively less frequently than the short-term scheduler.
- **Objective:**
    - Improve system performance by optimizing the use of main memory.
- **Key Considerations:**
    - Page fault rate.
    - Memory congestion.
    - Overall system performance.

3. **Short-Term Scheduler (CPU Scheduler):**
    - **Function:**
        - Selects the next process from the ready queue for execution on the CPU.
        - Determines the order and duration of process execution.
        - Manages the CPU by deciding which process to execute next.
    - **Frequency:**
        - Executes very frequently (milliseconds or microseconds).
    - **Objective:**
        - Optimize CPU utilization, throughput, and response time.
    - **Key Considerations:**
        - Process priorities.
        - CPU burst times.
        - Scheduling algorithms (e.g., Round Robin, Shortest Job First, Priority).

Non-preemptive Scheduling: In non-preemptive mode, once if a process enters into running state, it continues to execute until it terminates or blocks itself to wait for Input/Output or by requesting some operating system service.

Preemptive Scheduling: In preemptive mode, currently running process may be interrupted and moved to the ready State by the operating system.



Page Scheduling

Page replacement algorithms, including FIFO, LRU, LIFO, and Optimal, are used in the context of virtual memory management in operating systems. These algorithms determine which page to replace when a page fault occurs (i.e., when the desired page is not in the main memory). The goal is to minimize the number of page faults and optimize the use of available physical memory. Let's look at the use of each algorithm:

1. **First In First Out (FIFO):**
    - **Use:**
        - Simple and easy to implement.
        - Used in scenarios where simplicity is prioritized over optimality.
    - **Advantages:**
        - Easy to understand and implement.
        - Low overhead.
    - **Disadvantages:**
        - May not perform well in terms of page fault rate, especially when access patterns don't match the order of page allocation.
2. **Least Recently Used (LRU):**
    - **Use:**
        - Used when the system aims to keep track of the least recently used pages to minimize the probability of future page faults.
    - **Advantages:**

        - Generally performs well in capturing temporal locality.
        - Can lead to better overall system performance compared to FIFO.
    - **Disadvantages:**
        - Implementation can be more complex and may require additional data structures like counters or linked lists.
        - Some implementations may have higher overhead.

3. **Last In First Out (LIFO):**
    - **Use:**
        - Not commonly used in practical scenarios.
        - Historically used in some systems.
    - **Advantages:**
        - Simplicity in implementation.
    - **Disadvantages:**
        - Poor performance in capturing temporal locality.
        - May lead to higher page fault rates compared to other algorithms.
4. **Optimal Algorithm:**
    - **Use:**
        - Used as a reference point to evaluate the optimality of other algorithms.
        - Not practically implementable because it requires knowledge of future page accesses.
    - **Advantages:**
        - Serves as a theoretical benchmark for other algorithms.
        - Helps in analyzing the performance of other algorithms.
    - **Disadvantages:**
        - Not implementable in a real-world scenario due to the requirement of future knowledge.

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called frames and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

The main visible advantage of this scheme is that programs can be larger than physical memory.

Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. Virtual memory is commonly implemented by demand paging.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

1. Paging:

Paging is a memory management scheme used in computer operating systems to efficiently manage and allocate physical memory. In paging, physical memory is divided into fixed-size blocks called **frames**, and logical memory is divided into fixed-size blocks called **pages**. The size of a page is the same as the size of a frame.

- **Key Concepts:**
    - Pages are the basic units of allocation and represent logical memory.
    - Frames are the basic units of allocation in physical memory.
    - Both pages and frames are of the same size.

- **Address Translation:**
    - The logical address space is divided into pages.
    - The physical address space is divided into frames.
    - A page table is used for address translation. The page table maps logical pages to physical frames.

2. Page Fault, Demand Paging, Swap In, Swap Out:

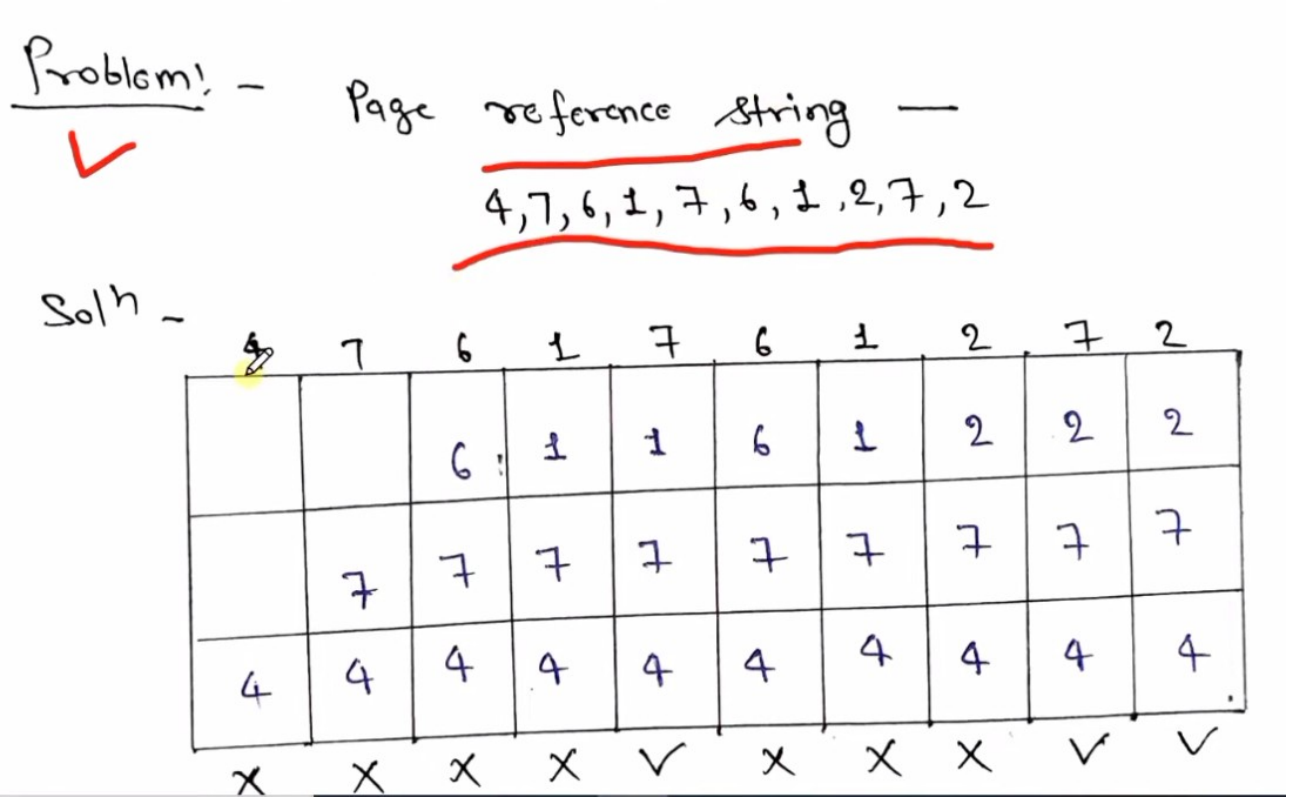
- **Page Fault:**
    - A **page fault** occurs when a program tries to access a page that is not currently in main memory (RAM).
    - The operating system must then bring the required page into memory, either from the disk or swap space.
- **Demand Paging:**
    - **Demand paging** is a strategy where pages are loaded into memory only when they are demanded by the program during execution.
    - It helps reduce the initial loading time and increases the degree of multiprogramming.
- **Swap In and Swap Out:**
    - **Swap In:** When a page fault occurs, and a required page is not in main memory, the operating system brings the page from the disk to main memory. This process is called **swap in**.
    - **Swap Out:** When there is not enough space in main memory, the operating system may decide to move a page from main memory to the disk to make room for a new page. This process is called **swap out**.

3. Page Replacement Algorithms:

- **Optimal Page Replacement:**
    - Replaces the page that will not be used for the longest period in the future.
    - **Example:** Consider the sequence of page references: 1 2 3 4 1 2 5 1 2 3 4 5. Optimal replacement would replace the page that will be referenced farthest in the future. It requires knowledge of future page references, making it impractical but serves as a benchmark.
- **FIFO (First-In-First-Out):**
    - Replaces the oldest page in memory (the page that has been in memory the longest).
    - **Example:** If the pages in memory are referenced in the order 1 2 3 4 1 2 5, and a new page 6 is referenced, FIFO would replace page 1.

- **LRU (Least Recently Used):**
  - Replaces the page that has not been used for the longest period.
  - **Example:** If the pages in memory are referenced in the order 1 2 3 4 1 2 5, and a new page 6 is referenced, LRU would replace the page that was least recently used, which is 3.
- **Clock (or Second-Chance) Algorithm:**
  - Similar to FIFO but uses a clock-like mechanism.
  - A "hand" points to the oldest page, and when a page needs to be replaced, it scans through the pages in a circular manner until it finds a page with a reference bit of 0.
  - **Example:** If the pages in memory are referenced in the order 1 2 3 4 1 2 5, and a new page 6 is referenced, the Clock algorithm would replace the page that is pointed to when the clock hand stops.

Last in first out



Memory Management

The primary role of the memory management system is to satisfy requests for memory allocation. Sometimes this is implicit, as when a new process is created. At other times, processes explicitly request memory. Either way, the system must locate enough unallocated memory and assign it to the process.

Contiguous and non-contiguous memory allocation are two fundamental approaches to managing memory in a computer system.

1. **Contiguous Memory Allocation:**
    - In contiguous memory allocation, a process is allocated a single block of contiguous memory (a contiguous chunk of consecutive memory locations).
    - The entire process is loaded into this continuous block of memory.
    - Advantages:
        - Simple and easy to implement.
        - Efficient access to memory since it is contiguous.
    - Disadvantages:
        - Fragmentation can occur, leading to wasted memory.
        - Limited flexibility in handling varying memory requirements.

    Subtypes of Contiguous Memory Allocation:

    Contiguous memory allocation can be done in two ways

    Fixed Partitioning − In fixed partitioning, the memory is divided into fixed-size partitions, and each partition is assigned to a process. This technique is easy to implement but can result in wasted memory if a process does not fit perfectly into a partition.

    Variable Partitioning − In variable partitioning, the memory is divided into variable size partitions, and each partition is assigned to a process. This technique is more efficient as it allows the allocation of only the required memory to the process, but it requires more overhead to keep track of the available memory.

    In a Variable Partition Allocation we use four strategies to allocate memory:

    First Fit, Best Fit, Worst Fit, Next Fit

2. **Non-contiguous Memory Allocation:**
    - In non-contiguous memory allocation, a process is allocated multiple blocks of memory that may not be physically adjacent.
    - Memory is allocated as needed, and different parts of a process can be located in different areas of memory.
    - Advantages:
        - More flexible in handling varying memory requirements.

        - Minimizes fragmentation issues.
    - Disadvantages:
        - Increased overhead due to the need for a memory map to track allocated memory blocks.
        - Slightly slower access to memory due to non-contiguous nature.

Subtypes of Non-contiguous Memory Allocation:

- **Paging:**
    - Memory is divided into fixed-size blocks called pages.
    - Processes are divided into fixed-size blocks called frames.
    - Allows for efficient use of memory, but may lead to internal fragmentation.
- **Segmentation:**
    - Memory is divided into variable-sized segments.
    - Each segment represents a logical division of the process (e.g., code segment, data segment).
    - Provides flexibility but may lead to external fragmentation.

1. Fragmentation in Memory Allocation:

Fragmentation refers to the phenomenon where memory becomes divided into small, non-contiguous blocks, making it challenging to use available memory efficiently. There are two main types of fragmentation:

- **Internal Fragmentation:**
    - Occurs when memory is allocated in fixed-size blocks, and the allocated block is larger than the actual data, resulting in wasted space within the block.
    - Common in fixed-size partitioning and some paging systems.
- **External Fragmentation:**
    - Occurs when free memory is scattered throughout the system, but there is not enough contiguous space to satisfy a memory request.
    - More common in dynamic allocation systems.

2. Effect of Allocation Method on Memory Utilization:

Choice of Allocation Method Affects Memory Utilization:

- The allocation method influences how efficiently the available memory is used.
- Different methods have varying levels of overhead, fragmentation, and adaptability to varying memory requirements.

3. Techniques to Address Fragmentation:

Operating Systems Use Several Techniques to Address Fragmentation:

- **Compaction:**
    - Rearranges memory to place all free memory together in one block, reducing external fragmentation.
- **Paging and Segmentation:**
    - Divide memory into fixed-size blocks (paging) or variable-sized segments (segmentation), reducing both internal and external fragmentation.
- **Buddy System:**
    - Allocates memory in powers of two and merges adjacent free blocks to combat fragmentation.
- **Memory Defragmentation Tools:**
    - Automatically reorganize memory to reduce fragmentation.

4. Fixed Size Memory Partitioning:

Advantages:

- **Simplicity:** Easy to implement.
- **Predictability:** Predictable allocation and deallocation.
- **Prevention of External Fragmentation:** Allocation in fixed-size blocks helps prevent external fragmentation.

Disadvantages:

- **Internal Fragmentation:** Can lead to internal fragmentation, especially if processes are not an exact fit for the partition size.
- **Inflexibility:** Inflexible for varying memory requirements of different processes.
- **Underutilization:** May lead to underutilization of memory due to partition size constraints.

Note: The choice of memory allocation method depends on the specific characteristics and requirements of the system, and there is no one-size-fits-all solution. Different methods may be suitable for different scenarios.

DLL

1. Working Mechanism of DLL:

DLL (Dynamic Link Library):

- DLL is a file containing compiled code that multiple programs can use simultaneously. It allows sharing of functions and resources among multiple applications.

Working Mechanism:

1. **Loading:**
    - DLLs are loaded into memory when a program starts or when explicitly called.
    - Dynamic Link Library (DLL) files contain code and data that can be used by multiple programs.
2. **Linking:**
    - Linking to a DLL is dynamic, meaning that the linking process is done during runtime.
    - Programs using a DLL are linked to it during execution, rather than at compile time.
3. **Shared Code and Data:**
    - Multiple programs can use the same DLL, allowing for code and data sharing.
    - Reduces duplication of code and conserves memory.
4. **Run-time Binding:**
    - DLLs allow run-time binding, enabling programs to call functions and use resources dynamically.
5. **Flexibility:**
    - Updates or modifications to a DLL do not require recompilation of programs using it.
    - Enhances flexibility and maintainability.

2. DLL (Dynamic Link Library):

DLL Definition:

- A **Dynamic Link Library (DLL)** is a file containing compiled code and data that can be used by multiple programs at the same time.

Usage and Advantages:

- **Usage:**
    - Shared libraries of functions and resources.
    - Modular programming and code reuse.
    - Efficient memory utilization.

    - Simplifies updates and maintenance.
- **Advantages:**
    - **Code Reusability:** Multiple programs can use the same DLL, reducing redundancy.
    - **Modular Programming:** Breaks down complex programs into manageable modules.
    - **Memory Conservation:** Shared DLLs reduce memory consumption.
    - **Dynamic Linking:** Allows updating DLLs without recompiling dependent programs.
    - **Enhanced Maintainability:** Changes to a DLL don't affect programs unless explicitly updated.

3. Types of Linking:

There are two main types of linking:

1. **Static Linking:**
    - Linking is done at compile time.
    - The entire code and libraries are combined into a single executable.
    - No external dependencies during runtime.
2. **Dynamic Linking:**
    - Linking is done at runtime.
    - Code from external libraries (DLLs) is linked during program execution.
    - Allows for the use of shared libraries, reducing redundancy.

4. Importing and Exporting Functions in a Module:

- **Importing Functions:**
    - Programs using a DLL import functions dynamically at runtime.
    - Declarations of functions used from the DLL are specified in the program.
- **Exporting Functions:**
    - In the DLL, functions to be used by external programs are explicitly marked for export.
    - Exported functions are listed in the DLL's export table.
- **Example (C++):**

  cppCopy code

  ```
  // Exported function in DLL
  ```

```
__declspec(dllexport) int Add(int a, int b) {
    return a + b;
}

// Importing function in another program
__declspec(dllimport) int Add(int a, int b);
```

5. Resource DLL:

- **Resource DLL:**
  - A **Resource DLL** is a DLL that contains resources such as images, icons, strings, etc., to be shared among multiple applications.
  - It allows centralizing resources for consistency and easier maintenance.
  - Programs dynamically load the resource DLL to access shared resources.
- **Advantages:**
  - Efficient resource management.
  - Centralized updates and modifications.
  - Simplifies localization and globalization efforts.

Resource DLLs are particularly useful in scenarios where multiple applications need access to a common set of resources, promoting code and resource reuse.

```
#pragma once
#ifdef MATHLIBRARY_EXPORTS
#define MATHLIBRARY_API __declspec(dllexport)
#else
#define MATHLIBRARY_API __declspec(dllimport)
#endif
namespace MathLibrary
{
// This class is exported from the MathLibrary.dll
class Functions
{
public:
// Returns a + b
static MATHLIBRARY_API double Add(double a, double b);
// Returns a * b
static MATHLIBRARY_API double Multiply(double a, double b);
// Returns a + (a * b)
static MATHLIBRARY_API double AddMultiply(double a, double b);
};
}
```

```
// MathLibrary.cpp : Defines the exported functions for the DLL application.
#include "pch.h"
#include "MathLibrary.h"
namespace MathLibrary
{
double Functions::Add(double a, double b)
{ return a + b; }
double Functions::Multiply(double a, double b)
{ return a * b; }
double Functions::AddMultiply(double a, double b)
{ return a + (a * b); }
}
```

```
// MathClient.cpp : Defines the entry point for the console application.
#include "pch.h"
#include <iostream>
#include "MathLibrary.h"

using namespace std;
int main()
{
double a = 7.4;
int b = 99;
cout << "a + b = " <<
MathLibrary::Functions::Add(a, b) << endl;
cout << "a * b = " <<
MathLibrary::Functions::Multiply(a, b) << endl;
cout << "a + (a * b) = " <<
MathLibrary::Functions::AddMultiply(a, b) << endl;
return 0;
}
```