Problem Statement: Dynamic Product Management System
Objective:
Develop a **Product Management System** that dynamically manages product details, including runtime and compile-time array allocation. The system should allow users to **create, modify, and display product details**, determine the most expensive product, and update the store name for all products.

Requirements:

1. **Product Class Implementation:**
   - A class Product should store product details.
   - **Attributes:**
     - pid: Unique product ID (automatically assigned).
     - pname: Name of the product.
     - price: Price of the product.
     - storeName (static): Common store name for all products.
     - count (static): Tracks the number of products created.
2. **Functionalities to Implement:**
   - **Constructors:**
     - A **default constructor** initializes a product with default values.
     - A **parameterized constructor** initializes a product with a name and price.
   - **Accept (accept())**: Takes user input for product details.
   - **Display (display())**: Shows product details.
   - **Accessors (getPname(), getPrice())**: Retrieves product details.
   - **Mutators (setPname(), setPrice())**: Modifies product details.
   - **Static Method (setStoreName())**: Updates the store name for all products.

Execution Flow:

1. **User Input:**
   - The user provides the number of products (size).
   - The program dynamically allocates an array of Product objects.
   - The user enters product details one by one.
2. **Processing:**
   - The system stores and displays product details.
   - It determines the **most expensive product**.
   - It updates the **store name** for all products.
3. **Memory Management:**
   - After operations, the dynamically allocated array is deleted to prevent memory leaks.

Example Input & Output:
Enter the size: 3
Enter product details:
Product 1: Laptop, Price: 50000
Product 2: Mobile, Price: 20000
Product 3: TV, Price: 70000

```
Displaying all products:
1. Laptop - ₹50000 (Store: Default Store)
2. Mobile - ₹20000 (Store: Default Store)
3. TV - ₹70000 (Store: Default Store)

The max price is ₹70000
Updating store name to "XYZ Store"...
Displaying all products again:
1. Laptop - ₹50000 (Store: XYZ Store)
2. Mobile - ₹20000 (Store: XYZ Store)
3. TV - ₹70000 (Store: XYZ Store)
```

Key Concepts Demonstrated:

✅ **Classes and Objects**
✅ **Dynamic Memory Allocation (new & delete)**
✅ **Constructors (Default & Parameterized)**
✅ **Static Members (storeName, count)**
✅ **Encapsulation (Private attributes, Public methods)**
✅ **Mutators & Accessors**
✅ **Finding Maximum Value in an Array**
✅ **Runtime vs Compile-time Object Creation**

Steps

1. Create Product.h

```
#pragma once  // Ensures the file is included only once in compilation

#include <iostream>
#include <string>
using namespace std;

class Product {
private:
    int pid;            // Unique product ID
    string pname;        // Product name
    float price;        // Product price
    static string storeName; // Common store name for all products
    static int count;     // Tracks product count

public:
    // Constructors
    Product();                  // Default Constructor
    Product(string pname, float price); // Parameterized Constructor
    // Methods
    void accept();   // Accept user input
    void display();  // Display product details

    // Accessors (Getters)
    string getPname();
    float getPrice();

    // Mutators (Setters)
    void setPname(string pname);
    void setPrice(float price);

    // Static function to set store name
    static void setStoreName(string storeName);
};
```

2. Create ProductSrc.cpp

Define the Static Variables in ProductSrc.cpp
Since **static variables** belong to the class, they must be initialized outside the class definition.

```
#include "Product.h"

// Initialize static variables
string Product::storeName = "Default Store";
int Product::count = 0;
```

Explanation of Step 2:
✅ storeName is **shared across all products** (initially "Default Store").
✅ count keeps track of the **total number of Product objects** created.

Step 3: Implement Constructors in ProductSrc.cpp
1. Default Constructor

```
Product::Product() {
    pid = ++count;  // Auto-increment ID
    pname = "Unknown";
    price = 0.0;
}
```

🔷 **Explanation:**
✅ pid is **auto-incremented** using count.
✅ Default values: "Unknown" for pname, 0.0 for price.
2. Parameterized Constructor

```
Product::Product(string pname, float price) {
    pid = ++count;   // Assign unique ID
    this->pname = pname;
    this->price = price;
}
```

🔷 **Explanation:**

✅ Uses this-> pointer to **differentiate instance variables** from parameters.

✅ Assigns **user-specified values** to pname and price.

Step 4: Implement Accept and Display Functions in ProductSrc.cpp

1. Accept User Input

```
void Product::accept() {
    cout << "\nEnter Product Name: ";
    cin >> pname;
    cout << "Enter Product Price: ";
    cin >> price;
}
```

🔷 **Explanation:**

✅ Prompts the user to enter **product name** and **price**.

2. Display Product Details

```
void Product::display() {
    cout << "\nProduct ID: " << pid
         << "\nName: " << pname
         << "\nPrice: ₹" << price
         << "\nStore: " << storeName << endl;
}
```

🔷 **Explanation:**

✅ Displays **product ID, name, price, and store name**.

Step 5: Implement Accessors (Getters) and Mutators (Setters)

1. Getters

```
string Product::getPname() {
    return pname;
}

float Product::getPrice() {
    return price;
}
```

🔷 **Explanation:**

✅ Allows reading **product name and price**.

2. Setters

```
void Product::setPname(string pname) {
    this->pname = pname;
}
```

```
void Product::setPrice(float price) {
    this->price = price;
}
```

🔷 **Explanation:**

✅ Allows modifying **product name and price**.

3. Static Method to Set Store Name

```
void Product::setStoreName(string storeName) {
    Product::storeName = storeName; // Updates static storeName
}
```

🔷 **Explanation:**

✅ Updates **store name for all products**.

Step 6: create client.cpp

```
#include"Product.h"

int main()
{
	//run time array
	Product* parr;
	int size;

	cout << "\n enter the size:";
	cin >> size;  //3

	parr = new Product[size];

	for (int i = 0; i < size; i++)
	{
		parr[i].accept();
	}
	for (int i = 0; i < size; i++)
	{
		parr[i].display();
	}

	float maxprice = parr[0].getPrice();
	for (int i = 1; i < size; i++)
	{
		if (maxprice < parr[i].getPrice())
		{
			maxprice = parr[i].getPrice();
		}
	}

	cout << "\n the max price is " << maxprice;

	Product::setStoreName("Xyz store");
	for (int i = 0; i < size; i++)
	{
		parr[i].display();
	}
```

```
        delete[] parr;


        //compile time array
        /*Product parr[3] = {Product("LG tv",89000),
              Product("Android tv",45000),
              Product("Sony Tv",78000)};

        for (int i = 0; i < 3; i++)
        {
              parr[i].display();
        }*/

}
```

Explanation of main() Function (Step-by-Step)

The main() function orchestrates the entire **Product Management System** by:

- **Dynamically allocating memory** for an array of Product objects.
- **Accepting and displaying product details**.
- **Finding the most expensive product**.
- **Updating the store name for all products**.

---

1 Declare Variables

```
Product* parr;
int size;
```

- parr → A **pointer** to a dynamically allocated array of Product objects.
- size → Stores **user-defined number of products**.

---

2 Get User Input for Number of Products

```
cout << "\nEnter the number of products: ";
cin >> size;
```

- Prompts the user to **enter the number of products** to manage.
- Stores the value in size.

---

3 Dynamically Allocate Memory for Products

```
parr = new Product[size];
```

- Uses new to allocate **dynamic memory** for an array of size Product objects.
- This allows **runtime creation** of objects instead of using a fixed-size array.

---

4 Accept Product Details

```
for (int i = 0; i < size; i++) {
   parr[i].accept();
}
```

- Loops through the dynamically allocated product array.
- Calls the accept() method on each product to **get input from the user**.

---

5 Display Product Details

```
for (int i = 0; i < size; i++) {
   parr[i].display();
}
```

- Loops through the product array and **displays each product's details** using display().

---

6 Find the Most Expensive Product

```
float maxPrice = parr[0].getPrice();  // Assume first product has the highest price

for (int i = 1; i < size; i++) {
   if (maxPrice < parr[i].getPrice()) {
      maxPrice = parr[i].getPrice();
   }
}

cout << "\nThe most expensive product costs ₹" << maxPrice;
```

Step-by-step Execution:

1. **Initialize maxPrice** with the price of the **first product**.
2. **Loop through the array** to check if any product has a **higher price** than maxPrice.
3. If a product has a **higher price**, **update maxPrice**.
4. Finally, print the most expensive product's price.

---

7 Update Store Name for All Products

```
Product::setStoreName("XYZ Store");
```

- Calls the **static method** setStoreName() to update the **store name for all products**.
- Since storeName is static, it affects **all objects of Product class**.

---

8 Display Products Again with Updated Store Name

```
cout << "\n\nAfter changing store name:";
for (int i = 0; i < size; i++) {
   parr[i].display();
}
```

- Loops through the product array **again** and **displays updated details**.
- Now, the store name will be "XYZ Store" instead of "Default Store".

---

9 Deallocate Memory

```
delete[] parr;
```

- **Frees the dynamically allocated memory** to prevent memory leaks.
- Since we used new for dynamic allocation, we must use delete[] to release the memory.

💡 Summary of Execution Flow in main()

Step	Operation
1	Declare variables (parr, size)
2	Get the number of products from the user
3	Allocate dynamic memory for size products
4	Accept user input for each product
5	Display all product details
6	Find and display the most expensive product
7	Update the store name using a static method
8	Display products again with the updated store name
9	Deallocate dynamically allocated memory

🛠 Example Execution

Input:

```
Enter the number of products: 3

Enter Product Name: Laptop
Enter Product Price: 50000

Enter Product Name: Mobile
Enter Product Price: 20000

Enter Product Name: TV
Enter Product Price: 70000
```

Output:

```
Product ID: 1
Name: Laptop
Price: ₹50000
Store: Default Store

Product ID: 2
Name: Mobile
Price: ₹20000
Store: Default Store

Product ID: 3
Name: TV
Price: ₹70000
Store: Default Store

The most expensive product costs ₹70000

After changing store name:

Product ID: 1
```

```
Name: Laptop
Price: ₹50000
Store: XYZ Store

Product ID: 2
Name: Mobile
Price: ₹20000
Store: XYZ Store

Product ID: 3
Name: TV
Price: ₹70000
Store: XYZ Store
```

🚀 **Key Takeaways**

✅ **Dynamic Memory Allocation:** Used new and delete[] for runtime object creation.
✅ **Encapsulation:** Used **private attributes** and **public methods**.
✅ **Static Members:** storeName shared across all objects.
✅ **Efficient Searching:** Loop used to find the most expensive product.
✅ **Proper Memory Management:** Memory was freed after use to prevent leaks.