1. Function overloading and overriding

```
#include <iostream>
#include <string>

// Base class  function overloading
class Base {
public:
   void process(int num) {
      std::cout << "Base: Square of " << num << " is " << (num * num) << std::endl;
   }

   void process(double num) {
      std::cout << "Base: Square of " << num << " is " << (num * num) << std::endl;
   }

   void process(const std::string &str) {
      std::cout << "Base: Concatenation of string \"" << str << "\" is \"" << (str + str) << "\"" <<
std::endl;
   }

   virtual void process() {
      std::cout << "Base: Default process method" << std::endl;
   }
};

// Derived class overriding
class Derived : public Base {
public:
```

```
    void process() override {
        std::cout << "Derived: Overridden process method" << std::endl;
    }
};

int main() {
    int choice;
    std::cout << "Choose an option:\n";
    std::cout << "1. Use Base class (Function Overloading)\n";
    std::cout << "2. Use Derived class (Function Overriding)\n";
    std::cout << "Enter your choice (1/2): ";
    std::cin >> choice;

    if (choice == 1) {
        Base baseObj;
        int type;
        std::cout << "Select the type of input:\n";
        std::cout << "1. Integer\n";
        std::cout << "2. Double\n";
        std::cout << "3. String\n";
        std::cout << "Enter your choice (1/2/3): ";
        std::cin >> type;

        switch (type) {
            case 1: {
                int intInput;
                std::cout << "Enter an integer: ";
```

```
            std::cin >> intInput;
            baseObj.process(intInput);
            break;
        }
        case 2: {
            double doubleInput;
            std::cout << "Enter a double: ";
            std::cin >> doubleInput;
            baseObj.process(doubleInput);
            break;
        }
        case 3: {
            std::string strInput;
            std::cout << "Enter a string: ";
            std::cin.ignore();
            std::getline(std::cin, strInput);
            baseObj.process(strInput);
            break;
        }
        default:
            std::cout << "Invalid choice!" << std::endl;
    }
} else if (choice == 2) {
    Derived derivedObj;
    derivedObj.process();
} else {
    std::cout << "Invalid choice!" << std::endl;
```

```
    }

    return 0;
}
```

2. Array start with index 3

```
#include <iostream>

int main() {

    const int startIndex = 3;
    const int endIndex = 9;
    const int size = endIndex - startIndex + 1;



    int array[size];
    for (int i = 0; i < size; ++i) {
        array[i] = 8 + i; // Initialize each element to start from 8
    }



    std::cout << "Index\tValue" << std::endl;
    for (int i = 0; i < size; ++i) {
        std::cout << (startIndex + i) << "\t" << array[i] << std::endl;
    }

    return 0;
```

```
}
```

3. Array take the input from less of the size of array

```
#include <iostream>

int main() {

   const int size = 8;



   int array[size] = {0};



   std::cout << "Enter values  0 to 4 of the array:" << std::endl;
   for (int i = 0; i <= 4; ++i) {
      std::cout << "Value for index " << i << ": ";
      std::cin >> array[i];
   }



   std::cout << "\nIndex\tValue\tReference" << std::endl;
   for (int i = 0; i < size; ++i) {
      std::cout << i << "\t" << array[i] << "\t" << &array[i] << std::endl;
   }

   return 0;
}
```