Практическая работа №1

Тема: Введение в основы C/C++ и структуры данных

Цель работы:
Ознакомление с базовыми элементами программирования на языке C/C++, включая работу с массивами, указателями и динамическими структурами данных (списки, стеки, очереди). Получение навыков работы с основными операциями над структурами данных и подготовки к параллельной обработке данных.

Теоретическая часть

Указатели и работа с памятью:
Указатели — это переменные, которые хранят адреса других переменных, что позволяет программе управлять памятью напрямую. Они особенно важны для создания **динамических структур данных**, таких как списки, стеки и очереди, где размеры данных могут меняться в процессе выполнения программы.
Пример использования указателей для динамического выделения памяти:
int arr = new int[N]; // выделение памяти для массива из N элементов*
**arr = 10; // изменение значения первого элемента массива через указатель*
delete[] arr; // освобождение памяти

- Указатели также необходимы для работы с динамической памятью, позволяя создавать структуры данных, которые могут адаптироваться к изменяющимся объемам данных. Это особенно важно для эффективного управления ресурсами при высоких нагрузках.

2. Основные структуры данных

Эти структуры данных являются основой для организации данных, используемых в параллельных и высокопроизводительных приложениях.

- **Массивы**:
  Массивы — это статические структуры данных, где элементы хранятся в последовательных блоках памяти. Они позволяют эффективно обращаться к элементам по индексу, но их размер фиксирован при создании.
- **Односвязные списки**:
  Односвязные списки состоят из узлов, каждый из которых содержит значение и указатель на следующий элемент. Они удобны для реализации структур, где данные добавляются и удаляются часто, но требуют больше памяти на хранение ссылок.
- **Стек и очередь**:
  Стек (LIFO) и очередь (FIFO) — это линейные структуры данных с определёнными правилами добавления и удаления элементов. Их

принципы работы позволяют эффективно реализовывать управление задачами и данными.

3. Введение в параллельное программирование с OpenMP

В заданиях практической работы вы будете использовать OpenMP для параллельной обработки данных, чтобы ускорить выполнение задач, таких как нахождение минимума и максимума в массиве.

- **OpenMP** — это библиотека для добавления многопоточности в C и C++. Она позволяет добавлять параллелизм в код с минимальными изменениями. Для этого используются директивы, которые позволяют автоматически распределять работу между несколькими потоками.
- **Основные директивы OpenMP**:
    a. **#pragma omp parallel** — создаёт параллельную секцию, где код выполняется одновременно несколькими потоками.
    b. **#pragma omp for** — распределяет итерации цикла между потоками для их одновременного выполнения.
    c. **reduction** — позволяет безопасно суммировать или находить максимум и минимум, используя несколько потоков.

Пример использования OpenMP для параллельного нахождения минимума и максимума:
#cpp

```cpp
#include <omp.h>
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> arr = {5, 8, 3, 9, 2, 7};
    int min_val = arr[0];
    int max_val = arr[0];

    #pragma omp parallel for reduction(min:min_val) reduction(max:max_val)
    for (int i = 0; i < arr.size(); i++) {
        min_val = std::min(min_val, arr[i]);
        max_val = std::max(max_val, arr[i]);
    }

    std::cout << "Минимум: " << min_val << ", Максимум: " << max_val << std::endl;
    return 0;
}
```

- **Как работает reduction**:
  Директива reduction создаёт локальные копии переменных min_val и max_val для каждого потока, которые затем объединяются, когда все потоки завершат выполнение, чтобы получить окончательные значения.

Задание

Часть 1: Работа с массивами

1. Создайте программу на C++, которая выполняет следующие операции:
    a. Создаёт массив из N элементов, заполняет его случайными числами от 1 до 100.
    b. Находит максимальный и минимальный элементы массива.
    c. Выводит массив, минимальное и максимальное значения.
2. **Параллелизация с использованием OpenMP**
   Добавьте параллельное выполнение с помощью OpenMP:
    a. Используйте директиву #pragma omp parallel for для нахождения максимума и минимума в массиве.
    b. Сравните время выполнения параллельной и последовательной реализации, используя библиотеку <chrono> для измерения времени.

Часть 2: Работа со структурами данных

1. Реализуйте следующие структуры данных:
    a. Односвязный список: реализуйте функции для добавления, удаления и поиска элемента.
    b. Стек: реализуйте функции push, pop, isEmpty.
    c. Очередь: реализуйте функции для добавления в конец, удаления из начала и проверки на пустоту.
2. **Параллельная работа с динамическими структурами данных**
   Добавьте параллельное выполнение некоторых операций с использованием OpenMP, например:
    a. Одновременное добавление нескольких элементов в очередь или список.
    b. Сравните, как производительность меняется с увеличением числа элементов и параллельного добавления данных.

Часть 3: Динамическая память и указатели

1. Реализуйте программу, которая создаёт динамический массив с помощью указателей и заполняет его случайными числами.
2. Напишите функцию для поиска среднего значения элементов массива.
3. **Параллельный подсчёт среднего значения**
   Используйте OpenMP для параллельного подсчёта суммы элементов и вычисления среднего значения.
    a. Добавьте директиву #pragma omp parallel for reduction(+:sum) для параллельного суммирования элементов массива.
4. Освободите память после завершения работы с массивом.

Контрольные вопросы

1. В чём основные отличия между массивами и динамическими структурами данных?
2. Что такое указатель, и как он используется в языке C++?
3. Объясните принцип работы стека и очереди.

4. Каковы преимущества и недостатки односвязных списков по сравнению с массивами?
5. Как правильно освобождать память в языке C++ после работы с динамическими структурами?
6. Почему важно понимать работу с указателями и динамической памятью для параллельного программирования?
7. Как использовать reduction в OpenMP для нахождения суммы, минимума или максимума в массиве?
8. Как влияет параллельное программирование на производительность при работе с большими массивами?