МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ПОЛІТЕХНІЧНИЙ КОЛЕДЖ
ОДЕСЬКОГО НАЦІОНАЛЬНОГО ПОЛІТЕХНІЧНОГО УНІВЕРСИТЕТУ
ВІДДІЛЕННЯ КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Завдання для виконання
на лабораторних роботах
з дисципліни

«Алгоритми та структури даних»

Херсон — 2015

Зміст

Перелік ілюстрацій

Перелік таблиць

Listings

Лабораторна робота № 1
Рекурсія і рекурсивні алгоритми

Мета роботи

Вивчити поняття, види рекурсії і рекурсивну тріаду, навчитися розробляти рекурсивну тріаду при вирішенні завдань мовою C++.

При виконанні лабораторної роботи для кожного завдання потрібно написати програму на мові C++, яка отримує на вході числові дані, виконує їх обробку відповідно до вимог завдання і виводить результат на екран. Для обробки даних необхідно реалізувати рекурсивну функцію. Введення даних здійснюється з клавіатури з урахуванням вимог до вхідних даних, що містяться в постановці завдання (введення даних супроводжуйте діалогом). Обмеженнями на вхідні дані є допустимий діапазон значень використовуваних числових типів в мові C++.

1.1 Теоретичні відомості

Лістинг 1.1 – Програма обчислення факторіалу ітераційним способом

```
public class App {
    public static void main(String[] args) {
        System.out.println(factorial(5));
    }

    public static int factorial(int arg) {
        int result = 1;
        for (int k = 1; k <= arg; k++) {
            result *= k;
        }
        return result;
    }
}

>> 120
```

Лістинг 1.2 – Програма обчислення факторіалу рекурсивним способом

```
public class App {
    public static void main(String[] args) {
        System.out.println(factorial(5));
    }

    public static int factorial(int arg) {
        if (arg == 1) {
            return 1;
        } else {
            return arg * factorial(arg - 1);
        }
    }
}

>> 120
```

1.2 Вказівки до виконання роботи.

Кожне завдання необхідно вирішити вивченими рекурсивними методами вирішення завдань і методами обробки числових даних у мові C++. Перед реалізацією коду кожного завдання необхідно розробити рекурсивну тріаду відповідно до постановкою завдання: виконати параметризацію, виділити базу і оформити декомпозицію рекурсії. Програму для вирішен-

ня кожного завдання необхідно розробити методом процедурної абстракції, використовуючи рекурсивні функції. Етапи супроводити коментарями в коді.

Слід реалізувати кожне завдання у відповідності з наведеними етапами:

- ◇ вивчити словесну постановку задачі, виділивши при цьому всі види даних;
- ◇ сформулювати математичну постановку задачі;
- ◇ вибрати метод розв'язання задачі, якщо це необхідно;
- ◇ розробити графічну схему алгоритму;
- ◇ записати розроблений алгоритм на мові C ++;
- ◇ розробити контрольний тест до програми;
- ◇ налагодити програму;
- ◇ подати звіт по роботі.

1.3 Індивідуальне завдання

Завдання до лабораторної роботи

1. Створіть програму для обчислення n-го числі Фібоначчі.

Контрольні запитання

1. Чи можна випадок непрямої рекурсії звести до прямої рекурсії? Відповідь обґрунтуйте.
2. Чи може рекурсивна база містити кілька тривіальних випадків? Відповідь обґрунтуйте.
3. Чи є параметри, база і декомпозиція єдиними для конкретного завдання? Відповідь обґрунтуйте.
4. З якою метою в задачах відбувається перегляд або коригування обраних параметрів, виділеної бази або випадку декомпозиції?
5. Чи є рекурсія універсальним способом вирішення завдань? Відповідь обґрунтуйте.
6. Чому для оцінки трудомісткості рекурсивного алгоритму недостатньо одного методу підрахунку вершин рекурсивного дерева?

Лабораторна робота № 2
Основи OpenMP

Мета роботи

Ознайомитись зі специфікацією OpenMP. Навчитись розробляти та виконувати програми, що використовують OpenMP.

2.1 Теоретичні відомості

Вступ

Одним зі стандартів для програмування систем з загальною пам'яттю є інтерфейс (API) OpenMP(Open Multi-Processing). Цей стандарт реалізовано для мов програмування C, C++ та Fortran на великій кількості комп'ютерних архітектур, включаючи платформи Unix та Microsoft Windows.

OpenMP API складається з набору директив компілятора pragma (прагм), функцій та змінних середовища, що впливають на поведінку паралельної програми під час її виконання. Коли прагми OpenMP використовуються в програмі, вони дають вказівки компілятору створити виконуваний модуль, який буде виконуватись паралельно з використанням декількох потоків.

В OpenMP застосовується модель паралельного виконання, що отримала назву «розгалудження-злиття». Така програма починається як один потік виконання, який називають начальним потоком. Коли потік зустрічає паралельну конструкцію, він створює нову групу потоків, що складається з цього потока та позитивного числа допоміжних потоків, і стає головним в новій групі. Всі члени нової групи виконують код в межах паралельної конструкції. В кінці такої конструкції є неявний бар'єр. Після її виконання код програми продовжує лише головний потік.

Прагми OpenMP

Для активації підтримки прагм OpenMP в компіляторі необхідно застосовувати додаткові параметри або флаги компіляції. Для Visual C++ таким параметром є /openmp, для gcc — `-fopenmp`. Також у Visual C++ увімкнути підтримку OpenMP можна за допомогою відповідного параметру в діалозі налаштування проекту: вибрати Configuration Properties, C/C++, Language і змінити значення параметру «OpenMP Support» на yes. Для використання функцій OpenMP необхідно підключити до проекту файл `vcomp.lib` (або `vcompd.lib` в режимі відлагодження).

Прагми OpenMP починаються зі слів `#pragma omp` і мають наступний формат:

`#pragma omp <директива> [список параметрів]`

OpenMP підтримує наступні директиви: atomic, barrier, critical, flush, for, master, ordered, parallel, parallel for, section, sections та single, що визначають або механізми розділення коду, або конструції синхронізації. Необов'язкові параметри директиви уточнюють її поведінку. Найбільш уживана директива — parallel, яка має наступний синтаксис: `#pragma omp parallel [список параметрів]`. Структурований блок Вона інформує компілятор, що структурований блок (складений оператор) має виконуватись паралельно в кількох потоках. Як правило, кількість потоків дорівнює кількості процесорів в системі. В якості прикладу розглянемо варіант класичної програми «Hello, world!» (див. 2.1).

Лістинг 2.1 – Програма Hello OpenMP!

```
#include <stdio.h>
int   main ( )
{
  #pragma omp   parallel
  {
    printf ( "␣Hello␣,␣␣OpenMP!\n" ) ;
```

```
  }   /* #pragma omp  parallel   */
  return   0;
}   /*   int  main( )   */
```

На двопроцесорній системі результат роботи цієї програми може бути наступним:

```
Hello, OpenMP!
Hello, OpenMP!
```

Функції та змінні середовища OpenMP

OpenMP пердбачає також набір функцій, що дозволяють:

- ◇ під час виконання програми отримувати та встановлювати різноманітні параметри, що визначають її поведінку, наприклад, кількість потоків, можливість вкладеного паралелізму.
- ◇ застосовувати синхронізацію на основі замків (locks). Прототипи функцій знаходяться в файлі omp.h. Розглянемо набір функцій OpenMP, що використовуються найчастіше.

Функція `omp_get_thread_num()` повертає номер потоку, в якому була викликана. Головний потік паралельного блоку має номер 0. Функція має наступний прототип:
`int omp_get_thread_num(void)`
Функція `omp_set_num_threads()` має наступний прототип:
`void omp_set_num_threads(int num_threads)`
і використовується для встановлення кількості потоків, що будуть виконуватись в наступному паралельному блоці. Для визначення поточної кількості паралельних потоків застосовується функція `omp_get_num_threads()` з наступним прототипом:
`int omp_get_num_threads(void)` В лістингу 2.2 наведено текст програми, що демонструє використання розглянутих функцій.

Лістинг 2.2 – Функції OpenMP

```
#include <stdio.h>
#include <omp.h>
int   main ()
{
  omp_set_num_threads (5) ;
  #pragma omp   parallel
  {
    printf ( "␣Hello␣,␣␣OpenMP!␣(␣thread␣␣num=%d)␣\n",
    omp_get_thread_num () ) ;
  }   /* #pragma omp  parallel   */
  return   0;
}  /*   int  main()   */
```

В стандарті OpenMP визначено ряд змінних середовища операційної системи, які контролюють поведінку OpenMP-програм. Зокрема, змінна `OMP_NUM_THREADS` визначає максимальну кількість потоків, що будуть виконуватись в паралельній програмі. Наприклад, в командному рядку Windows необхідно виконати наступну команду:
`set OMP_NUM_THREADS=4`
Таким чином, за допомогою функції `omp_set_num_threads()` або змінної середовища `OMP_NUM_THREADS` можна встановити довільну кількість потоків, що будуть створюватись під час виконання OpenMP-програм.

2.2 Індивідуальне завдання

Завдання до лабораторної роботи

1. Виконати програми, наведені в лістингах 2.2 та 2.1.

2. Змінити програму з лістінгу 2.2 таким чином, щоб кількість потоків вказувалась користувачем програми.

3. Розробити програму, в якій парні потоки друкують «Hello, OpenMP!», а непарні — ім'я та призвіще студента.

Контрольні запитання

1. Для програмування яких обчислювальних систем — з розподіленною чи загальною пам'яттю — використовується стандарт OpenMP?

2. Назвіть складові частини стандарту OpenMP.

Лабораторна робота № 3
Розпаралелювання коду в OpenMP

Мета роботи

Ознайомитись з засобами OpenMP для розпаралелювання коду. Навчитись розробляти та виконувати програми, що використовують OpenMP.

3.1 Теоретичні відомості

Директива for

Однією з директив, що застосовується досить часто, є директива for, яка належить до директив розподілення роботи (work-sharing directive). Ця директива інформує компілятор, що при виконанні циклу for в паралельному блоці ітерації циклу мають бути розподілені між потоками групи. Розглянемо роботу директиви for на прикладі програми, текст якої наведено в лістінгу 3.1.

Лістинг 3.1 – Застосування директиви for

```
#include <stdio.h>
#include <omp.h>
int main()
  {
  int i;
  #pragma omp parallel
    {
    #pragma omp  for
    for (i = 0;   i  <  5;   i++)
    printf ( "thread␣N%d␣i=%d\n", omp_get_thread_num () , i ) ;
    }  /* #pragma omp  parallel   */
  return 0;
}
```

Результат роботи програми наведено нижче:

```
thread N1 i=3
thread N1 i=4
thread N0 i=0
thread N0 i=1
thread N0 i=2
```

Порядок появи рядків на екрані може бути довільним і змінюватись при кожному запуску програми. Якщо з тексту програми вилучити директиву `#pragma omp for`, то кожний потік буде виконувати повний цикл for.

У зв'язку з тим, що розпаралелювання коду найчастіше виконується саме в циклічних конструкціях, OpenMP має скорочений варіант запису комбінації директив `#pragma omp parallel` і `#pragma omp for`: `#pragma omp parallel for for(i = 0; i < 5; ++i)`

Слід підкреслити, що розпаралелюватись можуть лише такі циклі, в яких ітерації не мають залежностей одна від одної. Якщо цикл не має залежностей, компілятор має змогу виконати цикл в любому порядку, навіть паралельно. Також цикли, які OpenMP може розпаралелювати, мають відповідати наступному формату:

- `for(цілий тип i = інваріант циклу1;`
- `i {<,>,=,!=,<=,>=} інваріант циклу2;`
- `i {+,-}= інваріант циклу3)`

Ці вимоги введені для того, щоб OpenMP міг визначити кількість ітерацій циклу. При розробці паралельних програм необхідно враховувати, які змінні є спільними (shared), а які приватними (private). Спільні змінні доступні всім потокам в групі, тому зміна значеннь таких змінних в одному потоці стає видимою в інших потоках. Що стосується приватних зміних, то кожний потік має окремі їх копії, тому їх зміна в одному потоці не відображається на інших потоках. За замовчуванням всі змінні в паралельному блоці є спільними. Лише в трьох випадках змінні є приватними. По-перше, приватними є індекси паралельних циклів. По-друге, приватними є локальні змінні паралельних блоків. По-третє, приватними стають змінні, що вказані в параметрах private, firstprivate, lastprivate и reduction прагми for.

Параметр private вказує, що для кожного потоку має бути створена локальна копія кожної змінної, що вказана в списку. Приватні копії будут ініціалізуватись значенням за замовчанням, при можливості буде застосовуватись конструктор за замовчанням.

Параметр firstprivate виконує аналогічні дії, однак перед виконанням паралельного блоку він копіює значення приватної змінної в кожний потік, застосовуючи при необхідності конструктор копій.

Параметр lastprivate теж подібний до параметра private, однак після виконання останньої ітерації циклу або конструкції паралельного блоку значення змінних, що вказані в списку цього параметра, привласнюються змінним основного потоку. При цьому може застосовуватись оператор привласнювання копій. Параметр reduction в якості аргументів приймає змінну й оператор. Оператори, що підтримуються reduction, наведені в табл. 3.1, а змінна має бути скалярною (наприклад, int або float). Змінна параметру reduction ініціалізується значенням, що вказане в табл. 3.1. Після завершення паралельного блоку вказаний оператор застосовується до кожної приватної копії і початкового значенні змінної.

Табл. 3.1. Оператори параметру reduction

Оператор reduction	Ініціалізаційне значення змінної
+	0
*	1
-	0
&	0 (усі біти в змінній встановлені)
|	0
^	0
&&	0
||	0

В лістінгу 3.2 наведено програму, що використовує розглянуті параметри.

Лістинг 3.2 – Параметри прагми for

```
#include <stdio . h>
#include <omp.h>
int   main ()
{
int i, j;
int array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
int sum;
#pragma omp   parallel
{
  #pragma omp for firstprivate(j)   \
  lastprivate(i)   \
  reduction (+: sum)
  for (i = 0; i<sizeof (array)/ sizeof (array[0]); i++)
  {
    long   private = i;
    sum += array[i];
  }   /*  for(i = 0;   i  <  5;   i++) */
  }  /* #pragma omp  parallel   */
printf ("%d\n", sum);
return 0;
}
```

В цьому лістінгу змінна i є приватною тому, що вона є змінною циклу for, а також тому, що вказана в параметрі lastprivate. Змінна j примусово зроблена приватною за допомогою firstprivate. Змінна private також є приватною через те, що вона оголошена в паралельному блоці. В кожному потоці екземпляр приватної змінної sum неявно ініціалізується нулем.

Директива sections

Як правило, OpenMP застосовується для розпаралелювання циклів. Однак, в OpenMP є також засоби для підтримки паралелізму на рівні функцій. Цей механізм називається секціями OpenMP (OpenMP sections). Для створення паралельного блоку секцій застосовується директива `#pragma omp sections`, або, аналогічно директиві `#pragma omp parallel for`, її скорочений варіант `#pragma omp parallel sections`. Секції в блоці створюються директивой `#pragma omp section`. Кожній секції ставиться у відповідність один потік, і всі секції виконуються паралельно. Фрагмент програми, що демонструє розпаралелювання на рівні функцій наведено в лістінгу 3.3.

Лістинг 3.3 – Директива omp section

```
void quickSort(int L, int R)
{
int i;
if (R > L)
  {
  i = partition (L,  R);
  #pragma omp parallel sections
    {
    #pragma omp section
    quickSort(L,   i -  1);
    #pragma omp  section
    quickSort( i  + 1, R);
    }   /* #pragma omp   parallel   sections   */
  }   /*  if   (R>  l)   */
}   /*   void   quickSort(int   l ,   int   r)   */
```

3.2 Індивідуальне завдання

Завдання до лабораторної роботи

1. Виконати програми, наведені в лістінгах 3.2 та 3.3.
2. Розробити програму для обчислення добутку елементів масиву.
3. Написати програму, що складається з двох потоків, з яких один обчислює суму, а інший добуток елементів масиву. Для створення потоків використовувати механізм секцій.

Контрольні запитання

1. Якими директивами виконується розпаралелювання циклів в OpenMP?
2. Які змінні в паралельних блоках є спільними, а які — приватними?
3. Яким чином діє параметр `private`?
4. Яким чином діє параметр `firstprivate`?
5. Яким чином діє параметр `lastprivate`?
6. Яким чином діє параметр `reduction`?
7. Які оператори можуть застосовуватись з параметром `reduction`?
8. Яким чином виконується розпаралелювання коду на рівні `функцій`?

Додаток А
Правила оформлення звіту

А.1 Титульна сторінка лабораторної роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Звіт до лабораторної роботи №123
з дисципліни «Web-програмування»

Тема: **«Основи мережі Internet»**

Виконав
ст.групи хПР1 Пупкін А.А.

Перевірив
ст.викладач Іванов Б.Б.

Херсон 2012

A.2 Приклади блок-схем

Правила виконання блок-схем задані наступними документами:

- ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения
- ГОСТ 19.002-80. Схемы алгоритмов и программ. Правила выполнения
- ГОСТ 19.003-80. Схемы алгоритмов и программ. Обозначения условные графические