МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи
«Рекурсивні функції в програмах мовою С++»
з курсу «Програмування»
для студентів напряму 6.040302 – Інформатика
і курсу «Програмування та алгоритмічні мови»
для студентів напряму 6.040303 – Системний аналіз

Затверджено редакційно-видавничою
радою університету,
протокол № 2 від 06.12.12.

Харків
НТУ «ХПІ»
2014

Методичні вказівки до лабораторної роботи «Рекурсивні функції в програмах мовою С++» з курсу «Програмування» для студентів напряму 6.040302 – Інформатика і курсу «Програмування та алгоритмічні мови» для студентів напряму 6.040303 – Системний аналіз / Уклад. М. І. Безменов, О. М. Безменова. – Х. : НТУ «ХПІ», 2014. – 16 с.

Укладачі: М. І. Безменов,
О. М. Безменова

Рецензент І. П. Гамаюн

Кафедра системного аналізу і управління

ВСТУП

Рекурсивні функції є ефективним засобом реалізації циклічних алгоритмів.

Мета роботи – освоєння методики визначення та практичного застосування рекурсивних функцій у програмах, написаних мовою С++.

1. ТЕОРЕТИЧНІ ОСНОВИ

1.1. Рекурсивні функції

Кожна з функцій може викликати інші функції, у тому числі звертатися до самої себе. Функція, що звертається до самої себе, називається ***рекурсивною***. Рекурсія може бути і неявною, коли функція `first` викликає функцію `second`, а та в свою чергу викликає функцію `first`. У зв'язку з наявністю взаємних посилань функцій однієї на іншу виникає проблема, що зумовлена вимогою С++ до описів: спочатку описати, потім використовувати. При неявній рекурсії обов'язковим є випереджальний опис функції за допомогою прототипу.

Обов'язковим елементом в описі будь-якого рекурсивного процесу є деяке твердження (оператор), що визначає умову завершення рекурсії; іноді вона називається опорною умовою. У цій умові може бути задане яке-небудь фіксоване значення, що обов'язково буде досягнуте в ході рекурсивного обчислення, дозволяючи тим самим організувати своєчасне зупинення процесу. Крім того, повинен бути другий елемент – спосіб вираження одного кроку розв'язання за допомогою іншого, простішого. Кількість рівнів вкладеності не обмежена і може бути досить великою.

Кожен виклик рекурсивної функції повинен наближати випадок, що зупиняє рекурсивні виклики, інакше виконання функції ніколи не припиниться через нескінченний ланцюжок рекурсивних викликів. У дійсності, нескінченний ланцюжок рекурсивних викликів не може реалізуватися, оскільки відбувається аварійне завершення програми. Найпростішим способом гарантування виконання опорної умови є зменшення деякої додатної величини до досягнення деякого «малого» значення.

Особливістю реалізації рекурсивних функцій є те, що в програмі існує тільки один екземпляр коду цієї функції. На кожному рівні рекурсії здійснюється нове звернення до цього коду. Рис. 1.1 ілюструє виконання рекурсивного процесу з трьома рівнями рекурсивного виклику; правда, для простоти по-

яснення **вважається**, що на кожному рівні створюється новий екземпляр коду функції. Розглянемо послідовність дій що виконується в цьому випадку.

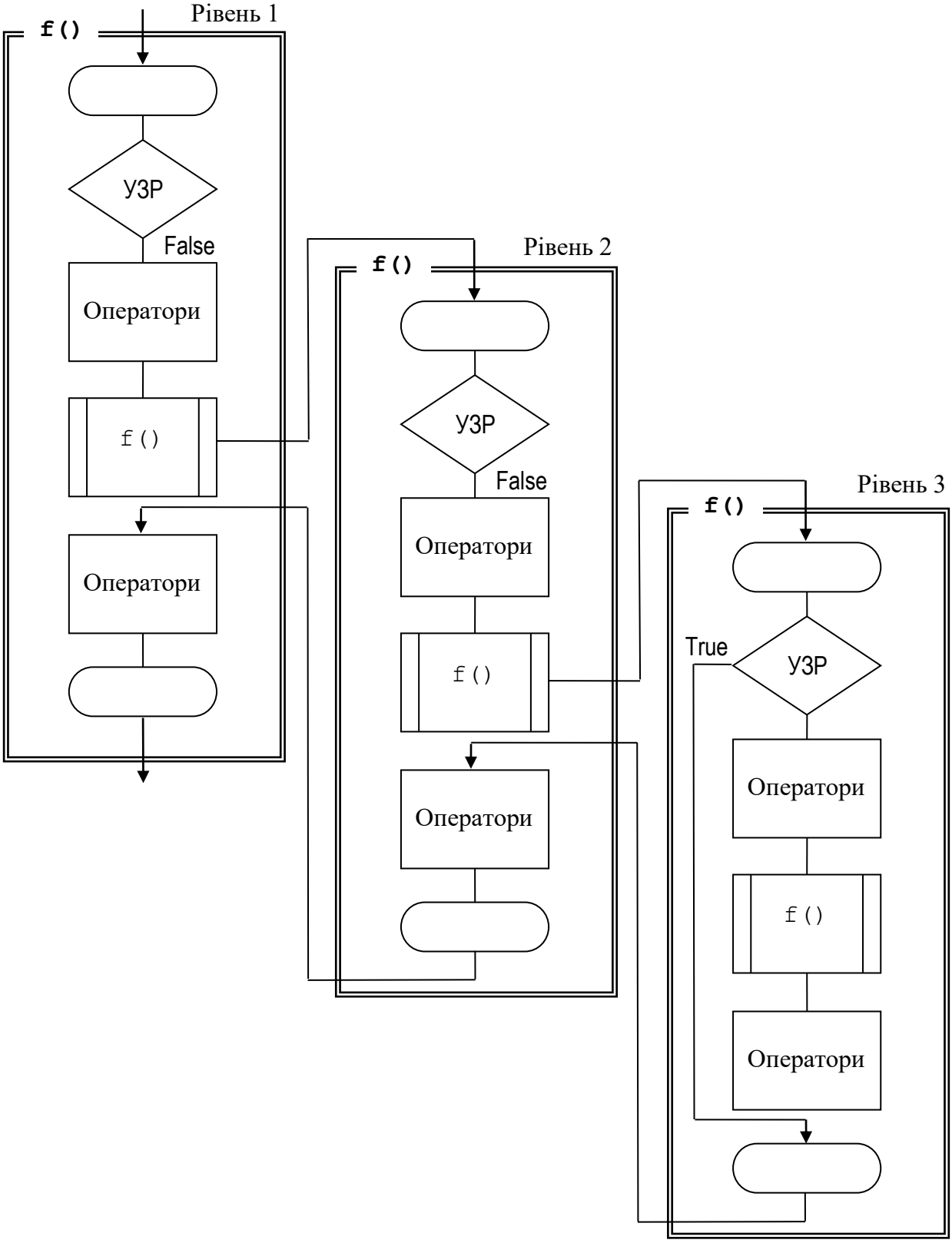

Рис. 1.1. Схема виконання рекурсивного процесу
(УЗР – умова завершення рекурсії)

Виклик функції забезпечує початок виконання її коду. Вважаючи що на першому рівні рекурсії опорна умова не виконується, маємо виконання операторів, які йдуть за опорною умовою, і новий виклик функції. На цьому виконання функції на першому рівні тимчасово переривається, і починається другий рівень рекурсії. Другий рівень рекурсії аналогічний першому. Він також тимчасово переривається з переходом до третього рівня рекурсії.

Вважаючи, що на третьому рівні рекурсії опорна умова виконана, маємо завершення третього рівня з поверненням на другий рівень у точку, де здійснилося тимчасове переривання його виконання. Далі завершуються обчислення на другому рівні рекурсії з наступним поверненням на перший рівень. Після завершення обчислень на першому рівні, отримуємо остаточний результат. Звичайно, у випадку виконання опорної умови також можуть виконуватися деякі оператори.

Незважаючи на простоту та наочність коду рекурсивних функцій, вони можуть вимагати дуже великих обсягів пам'яті для свого виконання. Це пояснюється тим, що, як і у випадку звичайних функцій, локальні змінні створюються в програмному стеку на кожному рівні рекурсивного виклику, що може привести до ситуації, коли обсяг стека просто вичерпається, оскільки знищення локальних змінних здійснюється тільки при виході з функції.

2. ПРИКЛАДИ ПРОГРАМ

Приклад 1. Дано ціле число n, $0 \le n \le 12$. Обчислити $n!$ за допомогою функції, яка реалізує такий метод обчислення факторіала:

$$0! = 1;\ k! = k \cdot (k-1)!,\ k = 1, 2, 3, \ldots$$

Розв'язок.

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

int factorialR(int k);                    // Прототип функції
                                  // Визначення головної функції
int main()
{
    int n;
    do
    {
        cout << "n = ";
        cin >> n;
    } while (n < 0 || n > 12);
```

```
    cout << n << "! == " << factorialR(n) << '\n';
    system("pause");
    return 0;
}

int factorialR(int k)
{
    if (k == 0) return 1;          //Якщо опорна умова виконана
    return k * factorialR (k - 1);                    //Якщо ні
}
```

У наведеному тексті функція `factorialR()` після першого її виклику в головній функції багаторазово звертається сама до себе зі значенням параметра `k`, яке зменшується від заданого при першому виклику початкового значення `n` до `0`. При досягненні параметром значення `0` виконується опорна умова і здійснюється вихід з функції. При цьому відбувається повернення в точку виклику функції на попередньому рівні з продовженням обчислення значення виразу `k * factorialR(k - 1)` і наступним виходом з функції з поверненням на попередній рівень.

Приклад 2. Визначити рекурсивну функцію множення цілих чисел через додавання: $x \times 0 = 0$, $x \times y = x + x(y-1)$ при $y > 0$.

Дано два цілих числа. Знайти їх добуток за допомогою розробленої функції.

Розв'язок.

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

int Multiplication(int x, int y);          // Прототип функції
                                  // Визначення головної функції
int main()
{
    int x, y;
    cout << "x = ";
    cin >> x;
    cout << "y = ";
    cin >> y;
    cout << x << " * " << y << " == " <<
                              Multiplication(x, y) << '\n';
    system("pause");
    return 0;
}
```

```
int Multiplication(int x, int y)
{
    if (y < 0)
    {
        y = -y;
        x = -x;
    }
    if (y == 0)                                  //Опорна умова
        return 0;                            // Завершення рекурсії
    return x + Multiplication(x, y - 1);  //Рекурсивний виклик
}
```

Приклад 3. Визначити і використати рекурсивну функцію підсумовування елементів масиву, що містить *n* дійсних чисел ($n \le 20$).

Розв'язок.

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

double SumRecurs(double *a, int Count);    // Прототип функції

int main()                          // Визначення головної функції
{
    int n;
    double a[20];
    do
    {
        cout << "n = ";
        cin >> n;
    } while (n < 1 || n > 20);
    for (int i = 0; i < n; i++)
    {
        cout << "a[" << i << "] = ";
        cin >> a[i];
    }
    cout << "Sum == " << SumRecurs(a, n) << '\n';
    system("pause");
    return 0;
}

double SumRecurs(double *a, int Count)   // Визначення функції
{
                // Якщо в масиві один елемент, то він і є сумою
    if (Count == 1)                              // Опорна умова
        return a[0];                         // Завершення рекурсії
```

```
            // Інакше до останнього елементу додаємо суму решти
    //елементів, яка обчислюється рекурсивним викликом функції
    return a[Count - 1] + SumRecurs(a, Count - 1);
}
```

Приклад 4. Дано ціле число *n*. Отримати нове число, десятковий запис якого співпадає з десятковим записом розвернутого на 180° числа *n* зі збереженням знаку. Визначити рекурсивну функцію «розвороту» цілого числа.

Розв'язок.

```
#include <iostream>
using namespace std;
int ReverseValue(int n, int *p = new int); // Прототип функції
int main()                          // Визначення головної функції
{
    int n;
    cin >> n;
    cout << ReverseValue(n) << '\n';
    system ("pause");
    return 0;
}
int ReverseValue(int n, int *p)          // Визначення функції
{
    if (n / 10 == 0)                            // Опорна умова
    {
        *p = 10;
        return n;                         // Завершення рекурсії
    }
    else
    {
        int k;         // Нижче здійснюється рекурсивний виклик
        return (k = ReverseValue(n / 10, p) + n % 10 * *p,
                                                *p *= 10, k);
    }
}
```

Функція `ReverseValue()` має два параметри: `n` – число, що «розвертається», `p` – вказівник, що вказує на комірку пам'яті для збереження і нарощування цілого степеня числа 10. Другий параметр є параметром за умовчанням; тому до функції можна звертатися як з одним параметром, так і з двома. Значенням за умовчанням є адреса комірки динамічної пам'яті, відведення якої здійснюється при першому виклику функції `ReverseValue()`. У тілі функції

здійснюється багаторазове звернення її до самої себе з першим фактичним параметром `n / 10`, що відповідає «відсіканню» цифри в запису числа `n` (самої крайньої праворуч). Оператор **`return`** повертає досить незвичне значення (`k = ReverseValue(n / 10, p) + n % 10 * *p, *p *= 10, k)`, в якому використовується операція «кома». При цьому в змінну `k` записується розвернута на 180° частина числа, до якої ліворуч «дописується» цифра (остача від ділення), яка перед підсумовуванням помножується на степінь числа 10 (а саме, `*p`), після чого значення `*p` помножується на 10 (`*p *= 10`). Значенням же виразу в дужках є значення змінної, яка записано після другої коми. Початковим значення змінної `*p` є число 10, записуване при виконанні опорної умови.

Особливістю алгоритму розв'язання наведеної вище задачі є те що в процесі виконання алгоритму повинні змінюватися дві величини – оброблюване число і степінь десяти, на значення якого помножується остача від ділення. При цьому оброблюване число повинне починати зменшуватися з першого рівня рекурсивних викликів, а початкове значення степеня (за яке береться число 10) може бути задане тільки на останньому рівні перед завершенням рекурсивного процесу.

Наведений вище код рекурсивної функції базується на використанні вказівника в якості її другого параметру. Можна запропонувати рекурсивну функцію, яка не використовує вказівники:

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

int ReverseValue(int n);                    // Прототип функції

int main()                          // Визначення головної функції
{
    int n;
    cin >> n;
    cout << ReverseValue(n) << '\n';
    system ("pause");
    return 0;
}

int ReverseValue(int n)                     // Визначення функції
{
    static int p;
    if (n / 10 == 0)                            // Опорна умова
```

```
    {
        p = 10;
        return n;                          // Завершення рекурсії
    }
    else
    {
        int k;          // Нижче здійснюється рекурсивний виклик
        return (k = ReverseValue(n / 10) + n % 10 * p,
                                            p *= 10, k);
    }
}
```

Ця версія рекурсивної функції `ReverseValue()` ґрунтується на використанні статичної локальної змінної (цілочислова змінна `p`), яка оголошується всередині її коду. Особливістю статичних локальних змінних є те що вони розподіляються в пам'яті один раз при першому зверненні до функції і зберігають своє значення при виході з функції і повторному вході в неї. Функція побудована так, що змінна `p` отримує своє початкове значення 10 тільки по завершенні останнього рівня рекурсії (при виконанні опорної умови), яке збільшується від рівня до рівня при зворотному проходженні.

Приклад 5. Дано дійсне число x та непарне натуральне число n. Обчислити

$$\cfrac{x}{1+\cfrac{x}{3+\cfrac{x}{5+\cfrac{x}{\ddots\,(n-2)+\cfrac{x}{n}}}}}.$$

Розв'язок.

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

double Value(double x, int n, int k = 1);  // Прототип функції

int main()                          // Визначення головної функції
{
    double x;
    int n;
    cout << "x = ";
```

```
    cin >> x;
    do
    {
        cout << "n = ";
        cin >> n;
    } while (!(n > 0 && n % 2 == 1));
    cout << "Result = " << Value(x, n) << '\n';
    system ("pause");
    return 0;
}

double Value(double x, int n, int k)      // Визначення функції
{
     if (k == n)                                  // Опорна умова
        return x / n;                       // Завершення рекурсії
                          // Нижче здійснюється рекурсивний виклик
     return x / (n - 2 + Value(x, n, k + 2));
}
```

Код функції ґрунтується на співвідношенні:

$$f(x,n,k)=\begin{cases} x/n & \text{при } k=n; \\ \dfrac{x}{i-2+f(x,n,i)}, i=1,3,\ldots k & \text{при } k<n. \end{cases}$$

Третій параметр функції `Value` є параметром зі значенням за умовчанням. Цей параметр змінюється від 1 (саме це значення задається за умовчанням) до n з кроком 2 при переході з одного рівня рекурсії до іншого.

Приклад 6. Дано дійсні числа a, b, c, d_1, d_2 ($d_1 \le d_2$) і близьке до 0 додатне число ε. Відомо, що на інтервалі $[d_1, d_2]$ рівняння $y = a\sin(bx + c)$ має корінь і цей корінь єдиний. Знайти корінь рівняння методом поділу відрізка навпіл. Вважати, що будь-яка точка інтервалу $[d_1, d_2]$ є коренем, якщо довжина інтервалу менше ε. Розробити рекурсивну функцію пошуку вказаним методом кореня рівняння однієї змінної з трьома коефіцієнтами.

Розв'язок.

```
#include <iostream>
#include <conio.h>

using namespace std;

double Root(double d1, double d2, double eps,
            double a, double b, double c);          // Прототип
```

```
double f(double x, double a, double b, double c); // Прототип
int main()
{
    double a, b, c;
    double d1, d2, epsilon;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
    cin >> c;
    cout << "d1 = ";
    cin >> d1;
    cout << "d2 = ";
    cin >> d2;
    cout << "epsilon = ";
    cin >> epsilon;
    double x = d1;
    cout << "x = " << Root(d1, d2, epsilon, a, b, c) <<'\n';
    getch();
    return 0;
}
                                // Визначення рекурсивної функції
double Root(double d1, double d2, double eps,
            double a, double b, double c)
{
    if (fabs(d1 - d2) < eps)                    // Опорна умова
        return d1;                        // Завершення рекурсії
                        // Підготовка до рекурсивного звернення
    double x = (d1 + d2) / 2;
    if (f(d1, a, b, c) * f(x, a, b, c) <= 0)
        d2 = x;
    else
        d1 = x;
    return Root(d1, d2, eps, a, b, c);   // Рекурсивний виклик
}
                                  // Визначення звичайної функції
double f(double x, double a, double b, double c)
{
        // Замість функції a * sin(b * x + c) можна взяти іншу
    return a * sin(b * x + c);
}
```

3. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

За час, відведений для виконання лабораторної роботи (2 академічні години), студент повинен:

1. Розробити алгоритм розв'язання задачі, запропонованої для програмування.
2. Здійснити проектування форми для функціонування розроблюваної програми.
3. Здійснити програмну реалізацію розробленого алгоритму.
4. Здійснити налаштування програми, виправивши синтаксичні та логічні помилки.
5. Підібрати тестові дані для перевірки програми, включаючи виняткові випадки.
6. Оформити звіт до лабораторної роботи.
7. Відповісти на контрольні запитання.
8. Здати викладачу працездатну програму з демонстрацією її роботи на декількох варіантах вихідних даних.

4. ВАРІАНТИ ЗАДАЧ

1. Дано цілі числа n, m ($n > 0, m \geq 0$). Знайти їх найбільший спільний дільник (НСД). Для розв'язання задачі визначити й використати рекурсивну функцію обчислення НСД, засновану на співвідношенні: НСД$(n,\ 0) = n$; НСД$(n,\ m)$ = = НСД$(m,\ r)$, де r – остача від ділення n на $m (m \neq 0)$.
2. Числа Фібоначчі u_0, u_1, u_2, … визначаються за таким правилом: $u_0 = 0$, $u_1 = 1$, $u_n = u_{n-1} + u_{n-2}$ ($n = 2, 3, ...$). Розробити рекурсивну функцію обчислення u_n для даного невід'ємного цілого n, що базується на безпосередньому використанні співвідношення $u_n = u_{n-1} + u_{n-2}$.
3. Дано невід'ємні цілі числа n та m. Написати рекурсивну функцію для обчислення функції Аккермана:

$$A(n,m) = \begin{cases} m+1, & \text{якщо} \quad n = 0, \\ A(n-1,1), & \text{якщо} \quad n \neq 0,\ m = 0, \\ A(n-1, A(n, m-1)), & \text{якщо} \quad n > 0,\ m > 0. \end{cases}$$

4. Скласти рекурсивну функцію, що обчислює суму двох цілих невід'ємних чисел шляхом багаторазового додавання числа 1. Наприклад, 6 + 10 = (6 + + 1) + (10 – 1) = (7 + 1) + (9 – 1) = ...

5. Описати функцію для обчислення C_n^m з використанням рекурсії:

$$C(0,n) = C(n,n) = 1,$$
$$C(m,n) = C(m,n-1) + C(m-1,n-1) \text{ при } 0 < m < n.$$

Дано цілі числа n і m ($n > 0$, $0 \le m \le n$). За допомогою описаної функції знайти значення C_n^m.

6. Дано натуральні числа m, n_1, n_2, …, n_m ($m > 1$). Обчислити найбільший спільний дільник (НСД) чисел n_1, n_2, …, n_m: НСД$(n_1, n_2, \ldots, n_m)$. Відомо, що для будь-якого натурального $k > 1$ виконується таке співвідношення:

$$\text{НСД}(n_1, n_2, \ldots, n_k) = \text{НСД}(\text{НСД}(n_1, n_2, \ldots, n_{k-1}), n_k).$$

Визначити рекурсивну функцію обчислення найбільшого спільного дільника елементів масиву натуральних чисел.

7. Дано натуральне число n і дійсне число x ($x \ne 0$). Обчислити значення ланцюгового дробу, розробивши для цього рекурсивну функцію:

$$\cfrac{x}{x + \cfrac{2}{x + \cfrac{4}{x + \cfrac{8}{\ddots \; x + \cfrac{n}{x}}}}}.$$

8. Описати функцію відшукання мінімального елементу масиву з n дійсних чисел, де n – натуральне число. У тілі функції не повинно бути явної організації циклу. Скористатися цією функцією для пошуку мінімуму в заданому одновимірному масиві.
9. Описати рекурсивну функцію сортування за зростанням масиву з n цілих чисел. Ідея методу така: помістити найменший елемент на першу позицію, після чого відсортувати частину масиву, що залишилася, за допомогою рекурсивного виклику.
10. Описати рекурсивну функцію з двома аргументами, яка без використання операторів циклу перевіряє, чи є одновимірний цілочисловий масив симетричним.
11. Дано квадратну матрицю $\boldsymbol{A}$ розміру $n \times n$, де n – натуральне число. Обчислити її визначник. Використати рекурсивну функцію обчислення визначника розкладанням за першим рядком.
12. Описати рекурсивну функцію з трьома аргументами (одновимірний дійсний масив і два індекси-обмежники), що не використовує оператори циклу.

Функція повинна розвертати на 180º частину одновимірного дійсного масиву від першого індексу-обмежника до другого включно. Протестувати цю функцію для таких випадків:

а) перший індекс-обмежник менший за другий;
б) перший індекс-обмежник більший від другого;
в) індекси-обмежники рівні.

5. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке рекурсія?
2. Що повинно бути у функції для правильної організації рекурсії?
3. Що таке неявна рекурсія?
4. Як треба описувати функції для неявної рекурсії?
5. Чи може бути нескінченним рекурсивний процес?
6. Що можна сказати про обсяги пам'яті, які необхідні для роботи рекурсивних функцій?
7. Перелічіть переваги та недоліки звичайних і рекурсивних функцій.
8. Маємо таку функцію:

```
int f(int n)
{
    if (n = 0) return 0;
    return n + f(n - 1);
}
```

Що вона повинна обчислювати? Чи є синтаксичні помилки в її коді? Виходячи із з'ясованого Вами призначення функції, зробіть висновки про наявність логічних помилок в її коді. Зробіть пояснення і виправлення помилок у разі їх наявності.

9. Маємо таку функцію:

```
int f(double *a, int n)
{
    if (n == 1) *a;
    return *a * f(a, n - 1);
}
```

Що вона повинна обчислювати? Чи є синтаксичні помилки в її коді? Виходячи із з'ясованого Вами призначення функції, зробіть висновки про наявність логічних помилок в її коді. Зробіть пояснення і виправлення помилок у разі їх наявності.

10. Маємо таку функцію:

```
int f(double *a, int n)
{
    return n > 1 ? *a * f(a, n - 1) : *a;
}
```

Що вона повинна обчислювати? Чи є синтаксичні помилки в її коді? Виходячи із з'ясованого Вами призначення функції, зробіть висновки про наявність логічних помилок в її коді. Зробіть пояснення і виправлення помилок у разі їх наявності.

11. Маємо таку функцію:

```
int f(double *p, double *q, double *a, int n)
{
    return n == 1 ? *p = *a, *q = *a, 0 :
        f(p, q, a, n - 1), *p < *(a + n - 1) ?
        *p = *(a + n - 1) : 0, *q > *(a + n - 1) ?
        *q = *(a + n - 1) : 0, 0;
}
```

Що вона повинна обчислювати? Чи є синтаксичні помилки в її коді? Виходячи із з'ясованого Вами призначення функції, зробіть висновки про наявність логічних помилок в її коді. Зробіть пояснення і виправлення помилок у разі їх наявності.

СПИСОК ЛІТЕРАТУРИ

1. Страуструп, Б. Язык программирования Си++ : Второе издание / Б. Страуструп. – К. : ДиаСофт, 1993. – Ч. 1. – 264 с. ; Ч. 2. – 296 с.
2. Керниган, Б. Язык программирования Си / Б. Керниган, Д. Ритчи. – М. : Финансы и статистика, 1992. – 272 с.
3. Либерти, Джесс. Освой самостоятельно С++ за 21 день : учеб. пособ. / Джесс Либерти. – М. : Вильямс, 2001. – 816 с.
4. Подбельский, В. В. Программирование на языке Си / В. В. Подбельский, С. С. Фомин. – М. : Финансы и статистика, 1999. – 600 с.
5. Подбельский, В. В. Язык Си++ / В. В. Подбельский. – М. : Финансы и статистика, 1999. – 560 с.
6. Савитч, Уолтер. Язык С++. Курс объектно-ориентированного программирования / Уолтер Савитч. – М. : Вильямс, 2001. – 704 с.

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи
«Рекурсивні функції в програмах мовою С++»
з курсу «Програмування» для студентів напряму 6.040302 – Інформатика
і курсу «Програмування та алгоритмічні мови»
для студентів напряму 6.040303 – Системний аналіз

Укладачі:
БЕЗМЕНОВ Микола Іванович,
БЕЗМЕНОВА Ольга Миколаївна

Відповідальний за випуск *О. С. Куценко*
Роботу до видання рекомендував *О. В. Горілий*
В авторській редакції

План 2013 р., поз. 9.

Підп. до друку 30.04.2013 р. Формат 60×84 1/16. Папір офсетний.
Riso-друк. Гарнітура Таймс. Ум. друк. арк. 0,9. Наклад 50 пр.
Зам. № 153. Ціна договірна.

Видавець і виготовлювач
Видавничий центр НТУ «ХПІ»,
вул. Фрунзе, 21, Харків, 61002.

Свідоцтво суб’єкта видавничої справи ДК № 3657 від 27.12.2009 р.