МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет “Львівська політехніка”

ВКАЗІВНИКИ НА ФУНКЦІЇ. РЕКУРСИВНІ ФУНКЦІЇ.

ІНСТРУКЦІЯ
до лабораторної роботи № 7 з курсу
“Основи програмування”
для базового напрямку “Програмна інженерія”

Затверджено
На засіданні кафедри
програмного забезпечення
Протокол № від

ЛЬВІВ – 2018

1. МЕТА РОБОТИ

Мета роботи – поглиблене вивчення можливостей функцій в мові С з використанням механізмів рекурсії та вказівників.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Вказівник на функцію

Як і дані, функція зберігається у пам'яті, а отже, вона має адресу. Тому на функцію можна встановити вказівник. Робиться це за допомогою такої конструкції:

```
тип_результату(* ім'я_вказівника)(список_параметрів);
```

Саме розміщення вказівника у круглих дужках і означає, що він посилається на функцію. Без круглих дужок значення конструкції зовсім інше – оголошувалася б функція, що повертає вказівник на дані зазначеного типу. Порівняйте:

```
int (*p)(int) /* оголошується вказівник на функцію, що повертає результат типу int */
int *p(int) /* оголошується функція, яка повертає вказівник на ціле число */
```

Оскільки в С ім'я функції є вказівником на першу комірку області пам'яті, де зберігається функція, то для ініціалізації слід просто присвоїти вказівнику ім'я функції. Наприклад, нехай ми маємо такі прототипи функцій:

```
int f3 (int );
int f5 (int );
int f7 (int );
```

Тоді можна описати такий вказівник на функцію, яка повертає результат типу `int` та має один формальний параметр також типу `int`:

```
int (*fst) (int );
```

Після такого оголошення, оператор присвоєння `fst = f3;` присвоїть вказівнику `fst` адресу входу у функцію `f3`, оператор `fst = f5`; - адресу входу у функцію `f5` і, відповідно, `fst = f7;` - адресу входу у функцію `f7`. Після цього викликати кожну з функцій (`f3, f5, f7`) можна будь-яким оператором, записаним нижче (наприклад, для функції `f3`):

```
f3(a);          // звертання до функції, використовуючи її ім'я.
(*fst) (a);     // виклик функції через вказівник.
fst(a);         // виклик функції також через вказівник.
```

Останній варіант також правильний, тому що і `f3`, і `fst` - це адреси входу у функцію. Однак виклик `(*fst) (a)` явно показує, що використовується вказівник на функцію, а не викликається функція з ім'ям `fst.`

Вказівники на функцію широко застосовуються в програмуванні:

- багато бібліотечних функцій як аргумент одержують вказівник на функцію;
- використання вказівників на функцію як аргументів дозволяє розробляти універсальні функції, наприклад функції чисельного рішення рівнянь, чисельного інтегрування й диференціювання;
- масиви вказівників на функції використовуються для організації меню.

Як і звичайні змінні, вказівники на функції можуть об'єднуватися в масиви. Наприклад, визначити й проініціалізувати масив вказівників на функції можна в такий спосіб. Нехай є такі функції:

```
float     func1(float);
float     func2(float);
float     func3(float);
float     func4(float);
float     func5(float);
```

Тоді масив вказівників на функції можна описати так:

```
float (*fparray [5])(float) = { func1, func2, func3, func4, func5};
```

Доступ до елементів масиву *fparray* виконується, як до звичайних елементів масиву. Наприклад:

```
float x = 1;
printf("%f", fparray[0](x)); //   або printf("%f", (*fparray[0]) (x);
```

Вказівники на функції застосовуються у випадках, коли потрібно викликати різні функції залежно від певних умов.

```
/*****************************************************************/
/*                Використання вказівників на функцію             */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

double suma(double,double);
double dobutok(double,double);

void main()
{
     double (*p)(double,double);
     int n1, n2; char ch;

     printf("Enter number 1\n");
     scanf("%d", &n1);
     printf("Enter number 2\n");
     scanf("%d", &n2);
     printf("Sum or divide (s|another key)?\n");
     scanf("%c", &ch);
     if(ch=='s')
          p = &suma;
     else
          p = dobutok;
     printf("%lf", p(n1,n2));
     _getch();
}

double suma(double n1,double n2)
{
     return (n1+n2);
}

double dobutok(double n1,double n2)
{
     return (n1*n2);
}
```

У даній програмі оголошено дві функції – `suma` та `dobutok`. Обидві функції приймають по два параметри типу `double` та повертають результат типу `double`. У тілі функції `main()` оголошено вказівник `p` на функцію, яка приймає два параметри типу `double` та повертає значення типу `double`: `(double (*p)(double,double))`. Користувачеві пропонується ввести з клавіатури два числа, які будуть записані у змінні `n1`, `n2`, та вибрати дію над цими числами – додати чи помножити. Для додавання чисел користувачеві пропонується натиснути клавішу ‘s’, а для множення – довільну іншу клавішу. Якщо користувач натиснув ‘s’, вказівнику `p` присвоюється адреса функції `suma` оператором `p = &suma;` інакше у вказівник `p` записується ім’я функції `dobutok` (тобто вказівник на першу комірку адреси області пам’яті, де зберігається функція) за допомогою оператора `p = dobutok;` Обидва записи є коректними. Виклик вибраної функції, передача параметрів і вивід результатів на екран здійснюється так: `printf("%lf", p(n1,n2));` На рис. 1 показані результати роботи програми для дії додавання та множення.

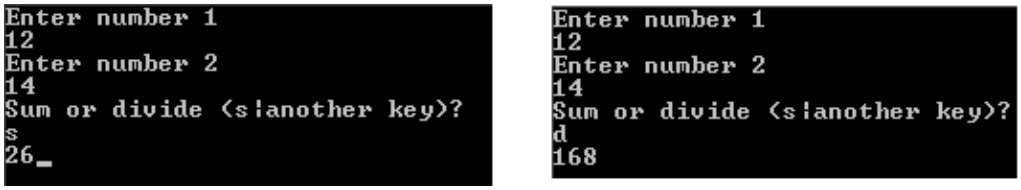

Рис. 1 Приклад роботи програми із застосуванням вказівника на функцію

2.2. Рекурсивні функції

В алгоритмічній мові С функції можуть бути рекурсивними. ***Рекурсивною*** називається така функція, яка звертається сама до себе (викликає саму себе). У рекурсивній функції обчислювальний процес повинен бути організований так, щоб на кожному кроці рекурсії задача поступово спрощувалася до тих пір поки для неї не стане можливим не рекурсивне рішення. Саме тому рекурсивна функція повинна повертати конкретний результат тільки для найпростішої частини задачі, тобто тоді коли можливо обчислити результат безпосередньо. В усіх інших випадках функція викликає саму себе, але з передачею нового значення аргументу, при якому нова задача є більш простою для розв'язання підзадачею вихідної задачі, причому алгоритм розв'язування цієї нової задачі є аналогічним алгоритму розв'язування вихідної задачі.

Створюючи рекурсивну функцію слід дотримуватися таких правил:

1) при кожному виклику такої функції в неї повинні передаватися нові дані;

2) на якомусь етапі повинен бути припинений подальший виклик даної функції;

3) після завершення кожного виклику рекурсивної функції в точку повернення повинен передаватися деякий результат для подальшого використання.

Рекурсія не є властивістю функції, це властивість опису функції. Про це свідчить такий простий приклад. Нехай потрібно знайти найбільше серед двох заданих чисел. Виявляється, що ця проста задача може бути розв'язана також і рекурсивним способом, як це показано у наступній програмі.

```
/*************************************/
/*функцію max можна зробити рекурсивною*/
/*************************************/
#include<stdio.h>
#include<conio.h>

int max(int, int);

void main()
{
    int a,b;
    printf("Enter a,b:");
    scanf("%d%d", &a, &b);
    printf("Max is %d", max(a,b));
    _getch();
}

int max(int a, int b)
{
    if (a<b) max(b,a);  // рекурсія в дії
    else return a;
}
```

У функцію `max` передаються значення двох цілих чисел. Функція повертає результат лише в тому випадку коли більше число задане першим аргументом. У протилежному випадку функція `max` звертається до самої себе, однак параметри `a` і `b` у виклику міняються місцями, тобто параметру `a` у функції передається більше значення (`b`), а параметру `b` - менше значення (`a`). Якщо ж спочатку умова `a < b` не виконується, то відразу ж буде мати місце не рекурсивний вихід.

Рекурсія нерідко входить у визначення математичних функцій. Найтиповішим прикладом є обчислення факторіала за допомогою рекурсії. Розглянемо на цьому прикладі детальніше

механізм рекурсивних функцій. Програма мовою С, яка реалізує задачу обчислення факторіала від заданого числа за допомогою рекурсії може мати такий вигляд.

```
/*************************************/
/*  обчислення факторіала  рекурсивно  */
/*************************************/
#include<stdio.h>
#include<conio.h>

unsigned long factorial(unsigned long);

void main()
{
    for(int i=1; i<=15; i++)
        printf("%d != %d\n", i, factorial(i));
    _getch();
}

unsigned long factorial(unsigned long number)
{
 if(number==1)
   return 1;
 else
  return number*factorial(number-1);
}
```

Пояснимо тепер процес виконання цієї програми. Функція `main` викликає функцію `factorial`, яка має параметр `number`. Якщо виконується умова `number==1`, то повернеться значення 1, інакше відбувається рекурсивний виклик функції з передачею нового значення аргумента `number-1`. Цей рекурсивний процес буде тривати до виклику `factorial(1)`, який поверне значення 1 в останню точку виклику.

Порядок виконання кожного виклику функції `factorial` і кожного повернення через оператор `return` для обчислення 5! схематично зображено на рис2.

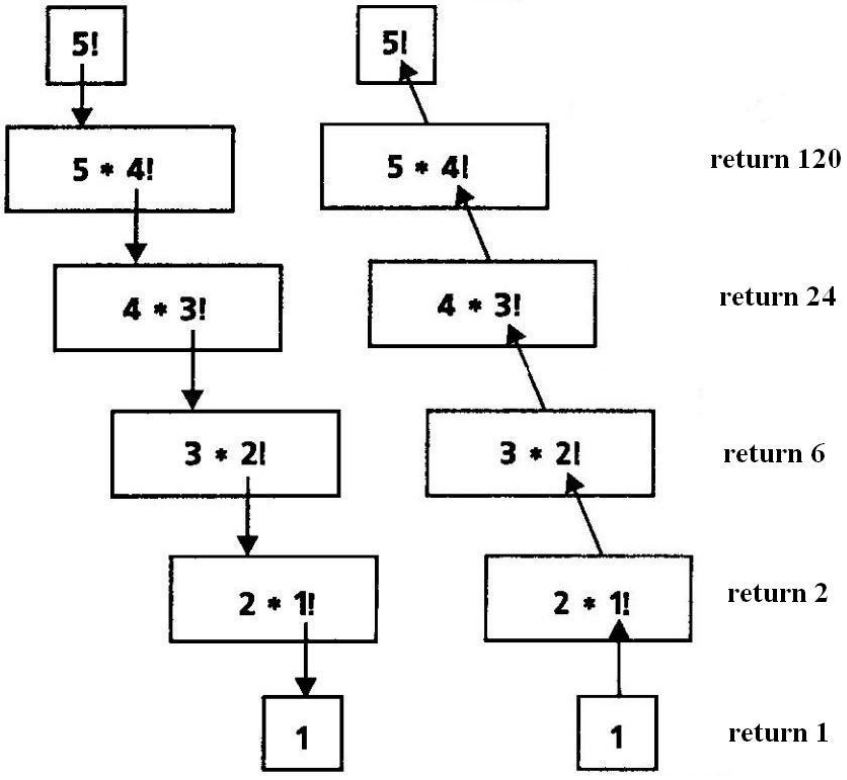

Рис.2. Порядок виконання рекурсивної функції обчислення факторіалу

З цього рисунку добре видно, що в процесі виконання рекурсивної функції спочатку відбувається послідовне звертання функції до самої себе до настання найпростішого випадку

обчислення цієї функції, коли стає можливим просте обчислення значення функції. Після цього моменту відбувається зворотній хід, тобто послідовне повернення результату кожного рекурсивного виклику в те місце функції, де він відбувся, до самого першого виклику (у цю точку виклику і буде повернено остаточний результат.

Іншим класичним прикладом застосування рекурсії є обчислення чисел, що належать послідовності Фібоначі. Текст програми мовою С, яка розв'язує цю задачу подано нижче.

```
/*************************************/
/*рекурсивне обчислення чисел Фібоначі */
/*************************************/
#include<stdio.h>
#include<conio.h>

unsigned long fibo(unsigned long);

void main()
{
    unsigned long n;

    printf("Enter n:");
    scanf("%lu", &n);
    printf("Fibo number: %d\n", fibo(n));
    _getch();
}

unsigned long fibo(unsigned long number)
{
 if(number==1)
     return 1;
 else if (number==0)
     return 0;
  else
     return fibo(number-1) + fibo(number-2);
}
```

Аналогічно до попереднього випадку, порядок роботи рекурсивного механізму виклику функції можна зобразити схематично (див. рис.3).

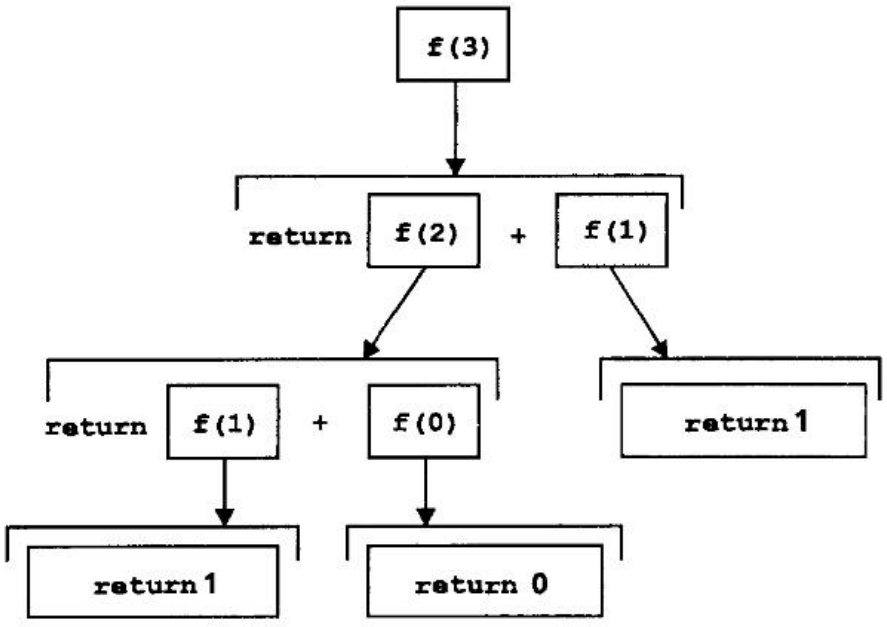

Рис.3. Порядок виконання рекурсивної функції обчислення чисел Фібоначі

Ще один характерний приклад застосування рекурсії - це швидке сортування елементів масиву (сортування Хоара). Основна ідея методу Хоара базується на принципі "розділяй і володій". Для заданого масиву вибирається елемент (який отримав назву "розділяючий

елемент"), що розбиває масив на дві частини, які потім сортуються незалежно один від одного. Очевидно, що це можна зробити лише в тому випадку, коли розділяючий елемент буде знаходитися на своєму місці, тобто всі елементи масиву зліва від нього будуть не перевищути його, а всі елементи справа від нього- навпаки не менші за розділяючий елемент (звісно коли мова йде про сортування у порядку зростання). Тоді та ж процедура може бути рекурсивно застосовуватися й до двох отриманих підмножин елементів масиву до тих пір поки в обох підмножинах залишається більше двох елементів. У протилежному випадку рекурсивний алгоритм швидкого сортування завершується.

Ефективність алгоритму швидкого сортування в значній мірі залежить від правильного вибору розділяючого елемента. Очевидно, що для найкращого результату, у ролі розділяючого елементу слід вибирати медіану масиву (тобто той елемент, який у відсортованому масиві буде знаходитися посередині масиву). Однак пошук медіани масиву на кожному кроці рекурсії приведе до значних обчислювальних затрат, і як наслідок, до суттєвої втрати в швидкодії самого алгоритму (і тоді той алгоритм важко буде називати швидким). Тому на практиці, у ролі розділяючого елементу вибирають медіану між першим, останнім та середнім елементами масиву, що сортується. Обчислювальні затрати на таку операцію будуть завжди постійними і виражатимуться не більше ніж трьома операціями порівняння. Іншим можливим підходом є вибір випадкового елемента масиву у ролі розділяючого елементу.

Реалізація вище описаного варіанту алгоритму швидкого сортування елементів масиву у порядку зростання наведена нижче у вигляді двох функції: функції обміну двох елементів масиву `swap` і рекурсивної функції `qs_sort(int array[], long start, long end)`, яка сортує підмножину масиву задану індексами `start` та `end`.

```
void swap(int array[], long pos1, long pos2)
{
 long tmp;
 tmp=array[pos1];
 array[pos1]=array[pos2];
 array[pos2]=tmp;
}

void qs_sort(int array[], long start, long end)
{
 long head=start, tail=end-1, tmp;
 long diff=end-start;
 long pe_index;

// якщо залишилося менше двох елементів – кінець рекурсії
 if (diff<1) return;
 if (diff==1)
  if (array[start]>array[end]) {
    swap(array, start, end);
    return;
  }

// пошук індексу розділяючого елементу pe_index
 long m=(start+end)/2;
 if (array[start]<=array[m]) {
   if (array[m]<=array[end]) pe_index=m;
   else if (array[end]<=array[start]) pe_index=start;
   else pe_index=end;
 }
 else {
   if (array[start]<=array[end]) pe_index=start;
   else if(array[end]<=array[m]) pe_index=m;
   else pe_index=end;
 }

 long pe=array[pe_index]; // сам розділяючий елемент
 swap(array,pe_index,end);
```

```
  while (1) {
    while(array[head]<pe)
      ++head;
    while(array[tail]>pe && tail>start)
      --tail;

    if (head>=tail) break;
    swap(array,head++, tail--);
  }

  swap(array,head,end);
  long mid=head;

  qs_sort(array, start, mid-1); // рекурсивний виклик для 1-оï підмножини
  qs_sort(array, mid+1, end);   // рекурсивний виклик для 2-оï підмножини
}
```

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке вказівник на функцію? Як його описати?
2. Як здійснюється виклик функції через вказівник? Наведіть приклад.
3. Яким чином відбувається об'єднання вказівників на функцію у масиви?
4. Які функції називаються рекурсивними?
5. Як правильно створити рекурсивну функцію?
6. Опишіть дію рекурсивної функції на прикладі обчислення x^n при $n = 4$.
7. Які переваги рекурсивного опису функції над не рекурсивним описом? І навпаки.

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичним матеріалом викладеним вище в даній інструкції і виконати приклади програм.
2. Одержати індивідуальне завдання з Додатку 1.
3. Скласти програму на мові С у відповідності з розробленим алгоритмом.
4. Виконати обчислення по програмі. Навести характеристики продуктивності комп'ютера на якому було виконано програму.
5. Одержати індивідуальне завдання з Додатку 2.
6. Розробити алгоритм розв'язання індивідуального завдання і подати його у вигляді блок-схеми.
7. Скласти програму на мові С у відповідності з розробленим алгоритмом.
8. Виконати обчислення по програмі при різних значеннях точності і порівняти отримані результати.
9. Одержати індивідуальне завдання з Додатку 3.
10. Розробити алгоритм розв'язання індивідуального завдання і подати його у вигляді блок-схеми.
11. Скласти програму на мові С у відповідності з розробленим алгоритмом.
12. Виконати обчислення по програмі при різних значеннях точності і порівняти отримані результати.Підготувати та здати звіт про виконання лабораторної роботи.

5. СПИСОК ЛІТЕРАТУРИ

1. Керниган Б., Ритчи Д. Язык программирования С. - М. - Финансы и статистика. - 1992. – 272 с.
2. Уэйт М., Прата С., Мартин Д. Язык С. Руководство для начинающих. - М. - Мир. - 1988. – 512 с.
3. К. Джамса. Учимся программировать на языке С++. М.: Мир, 1997. – 320 с
4. Герберт Шилдт. Полный справочник по С++. М. – С.-П.-К., Вильямс. – 2003. – 800 с.
5. Демидович Е. М. Основы алгоритмизации и программирования. Язык Си. (Учебное пособие). – Санкт-Петербург: "БХВ Петербург". – 2006. – 439 с.

6. ІНДИВІДУАЛЬНІ ЗАВДАННЯ

ДОДАТОК 1

Використовуючи вищенаведені функції `swap` та `qs_sort`, які реалізують алгоритм швидкого сортування масиву, написати програму мовою С для порівняння ефективності алгоритмів сортування масивів великих обсягів (наприклад, 100000 елементів). Програма повинна також реалізувати один з класичних алгоритмів сортування масиву згідно з варіантом індивідуального завдання. У програмі використати два одинакових масиви, які заповнити випадковими числами, здійснити перевірку впорядкованості елементів масиву, перевірку ідентичності масивів до і після сортування, а також за допомогою стандартної функції `time`, оцінити час виконання реалізованих алгоритмів сортування.

1. Сортування в порядку спадання “бульбашковим” методом без додаткової перевірки чи масив вже відсортований.
2. Сортування в порядку зростання методом вибору з пошуком мінімального елемента.
3. Сортування в порядку зростання “бульбашковим” методом з додатковою перевіркою чи масив вже відсортований.
4. Сортування в порядку спадання методом вставки.
5. Сортування в порядку зростання “бульбашковим” методом без додаткової перевірки чи масив вже відсортований.
6. Сортування в порядку зростання методом вибору з пошуком максимального елемента.
7. Сортування в порядку спадання “бульбашковим” методом з додатковою перевіркою чи масив вже відсортований.
8. Сортування в порядку спадання методом вибору з пошуком мінімального елемента.
9. Сортування в порядку зростання методом вставки.
10. Сортування в порядку спадання методом вибору з пошуком максимального елемента.
11. Сортування в порядку спадання “бульбашковим” методом без додаткової перевірки чи масив вже відсортований.
12. Сортування в порядку зростання методом вибору з пошуком мінімального елемента.
13. Сортування в порядку зростання “бульбашковим” методом з додатковою перевіркою чи масив вже відсортований.
14. Сортування в порядку спадання методом вставки.
15. Сортування в порядку зростання “бульбашковим” методом без додаткової перевірки чи масив вже відсортований.
16. Сортування в порядку зростання методом вибору з пошуком максимального елемента.
17. Сортування в порядку спадання “бульбашковим” методом з додатковою перевіркою чи масив вже відсортований.
18. Сортування в порядку спадання методом вибору з пошуком мінімального елемента.
19. Сортування в порядку зростання методом вставки.
20. Сортування в порядку спадання методом вибору з пошуком максимального елемента.
21. Сортування в порядку спадання “бульбашковим” методом без додаткової перевірки чи масив вже відсортований.
22. Сортування в порядку зростання методом вибору з пошуком мінімального елемента.
23. Сортування в порядку зростання “бульбашковим” методом з додатковою перевіркою чи масив вже відсортований.
24. Сортування в порядку спадання методом вставки.
25. Сортування в порядку зростання “бульбашковим” методом без додаткової перевірки чи масив вже відсортований.

ДОДАТОК 2

Написати мовою С три функції, щоб протабулювати, задану згідно варіанту, функцію на проміжку [*a* , *b*] з кроком *h*, використавши:

а) для першої функції оператор циклу for;

б) для другої – оператор циклу while;

в) для третьої – – оператор циклу do…while.

Вибір способу табулювання реалізувати через вказівник на відповідну функцію.

1. $f=\sin^2(x/2), a=0, b=\pi/2$;
2. $f=x^{15}\sqrt{1+3x^8}, a=0, b=1$;
3. $f=\cos^2(4x), a=-\pi/2, b=0$;
4. $f=\arcsin\sqrt{\frac{x}{1+x}}, a=0, b=3$;
5. $f=\frac{1}{\sqrt[3]{(x+1)^2}}, a=0, b=7$;
6. $f=\sin x \sin 2x \sin 3x, a=0, b=\pi/2$;
7. $f=(1+\sin 2x)^2, a=0, b=\pi/4$;
8. $f=\frac{1}{\cos^2(x/3)}, a=0, b=\pi$;
9. $f=e^x \cos^2 x, a=0, b=\pi$;
10. $f=sh^4 x, a=0, b=\ln 2$;
11. $f=x\sqrt[3]{1-x}, a=1, b=9$;
12. $f=\frac{1}{x\sqrt{x^2-1}}, a=-3, b=-2$;
13. $f=\sqrt{1-\cos 2x}, a=0, b=100\pi$;
14. $f=\sqrt[3]{(2x-1)}, a=0, b=4.5$;
15. $f=x^2 \cos x, a=0, b=2\pi$;
16. $f=\sqrt{1+2x}, a=0, b=4$;
17. $f=x e^{-x}, a=0, b=\ln 2$;
18. $f=x \sin x, a=0, b=\pi$;
19. $f=x\arctan(x), a=0, b=\sqrt{3}$;
20. $f=x^2\sqrt{4-x^2}, a=0, b=2$;
21. $f=x/(x^2+3), a=1, b=3$;
22. $f=x/\sqrt{(3x+2)}, a=0, b=2$;
23. $f=(3x^2+x-2)/(3x+1), a=0, b=1$;
24. $f=\cos^4 x, a=0, b=\pi$;
25. $f=\frac{\arcsin\sqrt{x}}{\sqrt{x(1-x)}}, a=0, b=1$;
26. $f=\frac{x\sin x}{1+\cos^2 x}, a=0, b=\pi$;
27. $f=(x\ln x)^2, a=1, b=e$.

ДОДАТОК 3

1. Використовуючи рекурсію, ввести групу даних і вивести їх у зворотному порядку, не оголошуючи масиву.
2. Використовуючи рекурсію, для кожного елемента з набору заданих додатніх цілих чисел вирахувати n! Масив не оголошувати.
3. Не оголошуючи масиву довгих цілих чисел, ввести групу даних. Вивести їх у зворотному порядку разом з їх порядковими номерами.
4. Обрахувати значення дробу

$$\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\ldots}}}}$$

використовуючи рекурсію. Значення n задає кількість членів (ступенів).

5. Порахувати ланцюговий дріб вигляду

$a_1/(b_1+a_2/(b_2+a_3/(b_3+a_4/(b_4+\ldots+a_{n-1}/(b_{n-1}+a_n/b_n)\ldots))))$., де a_n , b_n - задані цілі числа.

6. Ввести стрічку (число). Використовуючи рекурсивну функцію, отримати з введених символів все можливі перестановки знаків.
7. Ввести арифметичний вираз із дужками. Використовуючи рекурсивну функцію, перевірити правильність розстановки дужок.
8. Не оголошуючи масиву, ввести групу даних і вивести їх другу половину у зворотному порядку.
9. Обрахувати значення дробу

$$\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\ldots}}}},$$

використовуючи рекурсію. Значення n задає кількість членів (ступенів).

10. Не оголошуючи масиву чисел з плаваючою крапкою, ввести групу даних. Вивести їх разом з їх порядковими номерами.
11. Використовуючи рекурсію, для кожного елемента з набору цілих чисел вирахувати залишок від ділення на 3. Масив не оголошувати.
12. Знайти найбільший елемент заданого одновимірного масиву, використовуючи рекурсивну функцію.
13. Порахувати ланцюговий дріб вигляду

$1/(b_1+1/(b_2+1/(b_3+1/(b_4+\ldots+1/(b_{n-1}+1/b_n)\ldots))))$,де b_n - цілі числа.

14. Використовуючи рекурсію, для заданої послідовності додатніх цілих чисел знайти суму елементів. Масив не оголошувати.
15. Написати рекурсивну функцію переведення десяткового числа у вісімкове і використовуючи її знайти і вивести на друк вісімкове представлення перших 100 простих чисел.
16. Обрахувати значення дробу

$$\cfrac{2}{2+\cfrac{2}{2+\cfrac{2}{2+\cfrac{2}{2+\ldots}}}},$$

використовуючи рекурсію. Значення n задає кількість членів (ступенів).

17. Вивести у зворотньому порядку заданий рядок символів, використовуючи рекурсію.

18. Знайти найменший елемент кожного рядка у заданому двовимірному масиві, використовуючи рекурсивну функцію.
19. Використовуючи рекурсію, знайти всі парні числа з заданого набору цілих чисел. Масив не оголошувати.
20. Не оголошуючи масиву чисел з плаваючою крапкою, ввести 100 чисел. Вивести всі числа, порядкові номера яких є простими числами.
21. Не оголошуючи масиву, ввести групу даних, вивести їхню загальну кількість, порядкові номери і значення чисел у зворотному порядку. Ознака кінця вводу чисел - 0.
22. Написати рекурсивну функцію переведення десяткового числа у двійкове і використовуючи її знайти і вивести на друк двійкові коди перших 100 чисел Фібоначі.
23. Написати рекурсивну функцію знаходження мінімального елементу заданого одновимірного масиву і використовуючи її знайти масив, мінімальний елемент якого є максимальним (серед трьох заданих масивів).
24. Задано натуральні числа n, m. Обчислити значення біноміального коефіцієнта C_n^m використовуючи рекурсію за правилом: $C_n^0 = C_n^n = 1, \quad C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$.