

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет “Львівська політехніка”



**ОСОБЛИВОСТІ РОБОТИ З ФУНКЦІЯМИ В С.
ДИРЕКТИВИ ПРЕПРОЦЕСОРА.**

ІНСТРУКЦІЯ

до лабораторної роботи № 6 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 C та основ роботи з препроцесором.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1. Передача масивів у функції

У ролі аргументів функцій в C можуть використовуватися масиви. У цьому випадку достатньо вказати лише ім'я масиву без необхідності задавати квадратні дужки і розмірність масиву. Так, наприклад, якщо одновимірний масив `array` оголошений як

```
int array[25];
```

то передати його у функцію сортування можна так:

```
sort(array);
```

Прототип функції `sort` можна задати двома способами:

```
void sort(int []);      або      void sort(int *);
```

C автоматично передає масиви у функції використовуючи механізм виклику функції з передачею адрес. Це означає, що у функцію передається лише адреса масиву (тобто адреса першого елемента масиву), а не його копія (пригадуємо, що у мові C ім'я масиву без індексу є вказівником на перший елемент масиву). Тому функції, яким передається масив, можуть змінювати значення елементів масиву, тобто будь-яка модифікація масиву в функції буде автоматично змінювати масив у точці виклику цієї функції. У тілі самої функції доступ до елементів масиву може здійснюватися або через стандартний механізм індексації, або через механізм вказівника. Наступний приклад програми демонструє ці можливості.

```
/* *****  
/* передача масивів у функції та робота з ними всередині функції*/  
/* *****  
#include <stdio.h>  
#include <conio.h>  
  
void sort(int [], int);    // прототип функції з параметром типу масив  
void display(int *, int);  // прототип функції з параметром типу вказівник  
  
void main()  
{  
    const int n=10;  
    int array[n]={3,5,8,10,1,4,15,2,4,7};  
    printf("Array before sort: ");  
    display(array, n);    // виклик функції з аргументом типу масив  
    sort(array, n);       // ще один виклик функції з аргументом типу масив  
    printf("Array after sort: ");  
    display(array,n);  
    _getch();  
}  
  
void display(int *mas, int size)  
{    // доступ до елементів масиву здійснюється через механізм вказівників  
    for(int i=0; i<size; i++) printf("%d ", *(mas+i));  
    printf("\n");  
}  
  
void sort(int mas[], int size)  
{    // доступ до елементів масиву здійснюється через механізм індексації  
    int k=1, temp;  
    for(int i=0; (i<size)&&(k); i++) {  
        k=0;  
        for(int j=0; j<size-1; j++)  
            if(mas[j]>mas[j+1]) {  
                temp=mas[j]; mas[j]=mas[j+1];
```

```

        mas[j+1]=temp; k=1;
    }
}

```

Інколи виникають ситуації, коли потрібно заборонити модифікувати елементи масиву всередині функції. Оскільки масив передається за адресою, то контролювати самостійно всі зміни масиву в тілі функції може бути не простою задачею. Краще тут використати спеціальний специфікатор типу мови C, а саме специфікатор **const**. Коли параметр типу масив описується за допомогою *const*, то елементи такого масиву в тілі функції стають константами і будь-яка спроба їх модифікувати буде приводити до помилки (для прикладу спробуйте поміняти прототип та заголовок опису функції `sort` у попередній програмі на `void sort(const int [], int);` та `void sort(const int mas[], int size)`, відповідно).

Для передачі двовимірних масивів у функції необхідно явно задавати розмірність другого виміру масиву. Це пов'язано з тим, що компілятор використовує це значення розмірності для правильного визначення комірок пам'яті елементів багатовимірного масиву, тобто компілятору потрібна інформація про те скільки елементів знаходиться в одному рядку двовимірного масиву для того, щоб правильно розбити матрицю на окремі рядки. Наступний приклад програми демонструє можливості передачі двовимірних масивів у функції з їх подальшою обробкою.

```

/*****
/* передача 2D-масивів у функції та робота з ними всередині функції*/
*****/
#include <stdio.h>
#include <conio.h>

#define n 3
#define m 4

void sort(int [], int);
void display(int [][m], int);

void main()
{
    int matr[n][m]={ {3,5,8,10}, {1,4,15,2}, {4,7,2,9} };
    printf("Matrix before sort: \n");
    display(matr, n);
    for (int i=0; i<n; i++)
        sort(matr[i], m);
    printf("Matrix after sort: \n");
    display(matr,n);
    _getch();
}

void display(int mas[][m], int size)
{
    for(int i=0; i<size; i++) {
        for (int k=0; k<m; k++)
            printf("%d ", mas[i][k]);
        printf("\n");
    }
    printf("\n");
}

void sort(int mas[], int size)
{
    int k=1, temp;
    for(int i=0; (i<size)&&(k); i++) {
        k=0;
        for(int j=0; j<size-1; j++)
            if(mas[j]>mas[j+1]) {
                temp=mas[j];
                mas[j]=mas[j+1];
            }
    }
}

```

```

        mas[j+1]=temp;
        k=1;
    }
}

```

2.2. Функції зі змінною кількістю параметрів

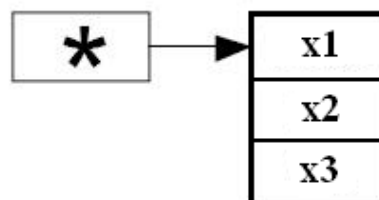
У С крім функцій з *фіксованим* числом параметрів є можливість також використовувати функції зі *змінною* кількістю параметрів, під якими розуміють функції, в які можна передавати дані, не описуючи їх усіх у прототипі й, відповідно, у заголовку опису функції. Класичним прикладом є успадковані з “чистої” С функції введення/виведення `scanf` та `printf`. Список формальних параметрів для функцій із змінним числом параметрів складається з постійних параметрів та змінної частини, яка задається за допомогою конструкції еліпсису, що представляє собою три крапки `...`, причому спочатку перераховуються постійні параметри. У загальному випадку формат оголошення функції зі змінною кількістю параметрів має такий вигляд:

`[тип_результату] ім'я_функції(параметр1, параметр2, ...);`

де поля `параметр1`, `параметр2` представляють постійну частину списку параметрів (хоча насправді тут може бути тільки один параметр або більше, ніж два параметри), а еліпсис `...` - змінну частину, яка означає довільну кількість параметрів.

Для розуміння механізму роботи з функціями із змінним числом параметрів слід уважніше розглянути процес, який відбувається при виклику функції і передачі аргументів. Як відомо, передача аргументів у функції в С відбувається через спеціальну ділянку оперативної пам'яті, яка називається *стек*. У С/С++ фактичні параметри записуються в стек з кінця списку аргументів, причому ці дані поміщаються в стек відповідно до типу, який використовується при виклику функції. Оскільки принцип роботи зі стеком полягає в тому, що значення, яке було занесено до стеку останнім, буде використовуватися (витягуватися зі стеку) першим, то, наприклад, для такого виклику функції $f(x1, x2, x3)$, схематично вміст стеку можна зобразити так:

вершина стеку



Тепер стає зрозуміло, що маючи вказівник на початок (вершина стеку) і знаючи тип даних аргументів, ми можемо послідовно перебрати всі аргументи зі стеку (поки стек не стане порожнім), навіть не знаючи наперед їх кількість. Саме така організація передачі аргументів в С і робить можливим створення та використання функцій зі змінною кількістю параметрів.

Очевидно, що при передачі у функцію додаткових аргументів (тобто аргументів, що відповідають змінній частині списку формальних параметрів) відсутня інформація про їх тип даних на етапі компіляції програми. Тому контроль типів аргументів та параметрів, а також приведення аргументів до типу параметрів не відбувається. Уся відповідальність за правильну обробку типів таких аргументів покладається на програміста. **Якщо передати функції зі змінним числом параметрів параметри не того типу, обробка якого запрограмована, наслідки будуть непередбачувані і, скоріше за все, сумні, адже компілятор у цьому випадку не перевірятиме ні кількість, ні типи параметрів.** Більше того, на програміста також покладається задача забезпечення доступу до додаткових аргументів, адже їхні імена відсутні. Видобути аргументи зі стеку у цьому випадку можна лише через вказівники. Причому тут існує два способи задання довжини змінного списку параметрів: в одному з постійних аргументів вказувати їх кількість або ж додати у кінець списку аргумент з унікальним значенням, що служитиме ознакою кінця списку. Наприклад: `func1(5,x1,x2,x3,x4,x5);` - тут зазначене число аргументів - 5; `func2(x1,x2,x3,x4,0);` - тут зазначена ознака кінця списку - 0.

У стандарт мови C входять макроси для роботи зі списками параметрів змінної довжини. Ці макроси визначені у файлі `stdarg.h`. При їх використанні також доводиться вказувати у списку постійний параметр, оголосити та встановити на нього вказівник та переміщувати цей вказівник по списку. В кінці списку має бути `NULL`. Макроси мають наступний формат:

```
void va_start(va_list prm, останній_фіксований_параметр);
тип va_arg(va_list prm, тип);
void va_end(va_list prm);
```

Тип вказівника повинен бути `va_list`, наприклад: `va_list p`;

Макрос `va_start` встановлює вказівник типу `va_list` на останній фіксований параметр. Макрос `va_arg` переміщає вказівник на наступний параметр. Синтаксис наступний: `va_arg(вказівник_типу_va_list, тип_чергового_параметра)`. У момент написання програми програміст повинен знати тип кожного з параметрів. Макрос `va_end` встановлює вказівник в `NULL`.

```
/* **** */
/* Функція для обчислення добутку довільної кількості дійсних чисел */
/* **** */
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>

double multiply(char *, ...); // прототип функції зі змінним числом параметрів

int main(int argc, char * argv[])
{
    multiply("4.0*6.0*2.0 = ", 4.0, 6.0, 2.0, -1.0);
    multiply("2.0*3.0*5.0*0.5 = ", 2.0, 3.0, 5.0, 0.5, -1.0);
    multiply("This product is ", -1.0);
    _getch();
    return 0;
}

double multiply(char *str, ...)
{
    double product = 1, term;

    va_list arg_list; // оголошуємо вказівник типу va_list
    va_start(arg_list, str); // встановлюємо вказівник на початок змінної
                             // частини списку параметрів

    int k = 0;
    while((term = va_arg(arg_list, double))>0)
    {
        product *=term;
        k = 1;
    }
    if(!k) product = 0;
    printf("%s%lf\n", str, product);
    va_end(arg_list); // завершуємо обробку списку параметрів
    return product;
}
```

Функція `multiply` повертає добуток довільної кількості дійсних чисел, заданих другим та наступними параметрами функції. Ця функція викликається в тілі функції `main` тричі: перший раз з трьома параметрами, другий – з чотирма, а третій – з одним обов'язковим параметром, що є рядком символів, та одним параметром, що служить ознакою кінця списку. Поточне значення добутку зберігається у змінній `product`. Початкове значення цієї змінної встановлене в 1, а на кожній ітерації циклу воно множиться на значення параметра функції дійсного типу, оброблюваного у поточний момент. Для випадку, коли функції `multiply` не передано

параметрів дійсного типу, слід змінну `product` встановлювати в 0. Для цього застосовується змінна `k`, що служить ознакою – чи виконався цикл принаймні один раз (тобто, чи має функція принаймні один параметр дійсного типу).

Параметри обробляються за допомогою макросів, визначених у файлі `stdarg.h`, тому у програмі присутня директива `#include <stdarg.h>`. Єдиним обов'язковим параметром функції `multiply` є параметр `str` типу `char *`. Тому для перебору параметрів функції `multiply` створюємо вказівник `arg_list` спеціального типу `va_list` та спочатку встановлюємо його на параметр `str` (`va_start(arg_list, str);`). У циклі переміщаємося по параметрах за допомогою макросу `va_arg`: `term = va_arg(arg_list, double)` – кожен такий запис посуває вказівник на 1 позицію, причому враховується, що оброблювані параметри функції є типу `double`, а поточний параметр записується у заздалегідь оголошену змінну `term` типу `double`. Цикл продовжуватиметься, поки `term > 0`. При виклику функції `multiply` їй в якості останнього параметра передаємо значення `-1` або якесь інше від'ємне значення чи значення 0. Як тільки буде досягнуто останній параметр функції, умова циклу `while` стане хибною і тіло циклу більше не виконається.

Для обробки параметрів функції зі змінною кількістю параметрів можна обійтися і без стандартних засобів, лише з використанням вказівників, як це показано у наступному прикладі.

```
/******  
/*Функція обчислення середнього арифметичного довільної кількості дійсних чисел */  
/******  
#include <stdio.h>  
#include <conio.h>  
  
double f(double , ...); // прототип функції зі змінним числом параметрів  
  
int main()  
{  
    printf("Seredne aryfmetychne 2,4,3 = %lf\n", f(2.0,4.0,3.0,0.0));  
    printf("Seredne aryfmetychne 2,14,8,16 = %lf\n", f(2.0,14.0,8.0,16.0,0.0));  
    _getch();  
    return 0;  
}  
  
double f(double n, ...)  
{  
    double *p = &n; // вказівник на початок списку параметрів  
    double sum = 0;  
    int count = 0;  
  
    while (*p) // поки в списку є ненульовий елемент  
    {  
        sum+=(*p);  
        p++; // переходимо на наступний параметр зі списку  
        count++;  
    }  
    return ((sum)?sum/count:0);  
}
```

Робота функції `f` базується на тому, що її останній параметр обов'язково має бути 0. Встановлюємо вказівник `p` на перший параметр функції `n`. Поки значення, що знаходиться за адресою, на яку посилається вказівник, не дорівнює нулю (тобто, до досягнення останнього елемента) поточний оброблюваний параметр функції додаємо до суми `sum` та збільшуємо лічильник `count` на 1. Функція `f` повертає `sum/count` або ж 0, якщо сума рівна 0. При цьому останній параметр (0) є виключно службовим – він не застосовується при розрахунку середнього арифметичного, оскільки при досягненні цього параметра умова циклу `while` стає хибною і тіло циклу більше не виконується.

Довжину змінної частини списку аргументів можна також задавати за допомогою фіксованого параметру, що демонструє такий приклад.

```

/*****
/*Функція обчислення середнього арифметичного довільної кількості дійсних чисел */
/* Варіант 2 */
*****/
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>

double f(int , ...);

int main()
{
    printf("%lf\n", f(3,2.0,4.0,3.0));
    printf("%lf\n", f(4,2.0,14.0,8.0,16.0));
    printf("%lf\n", f(10,1.5,1.5,2.8,3.4,2.0,14.0,8.0,16.0,4.5,2.3));
    _getch();
    return 0;
}

double f(int n, ...)
{
    double sum=0, term;
    int count = 0;

    va_list p;
    va_start(p, n);
    for (int i=0;i<n;i++)
    {
        term = va_arg(p, double);
        printf("term #%d = %lf\t", (i+1), term);
        sum+=term;
        count++;
    }
    va_end(p);
    printf("\nAverage = ");
    return ((sum)?sum/count:0);
}

```

У даному випадку фіксований параметр `n` функції `f` містить число параметрів, які будуть фактично передані функції при виклику. Вказівник `p` типу `va_list` спочатку вказує на єдиний обов'язковий параметр `n` функції `f`. У циклі з лічильником `i` за допомогою макросу `va_arg` “перебираємо” усі параметри, записуючи поточний параметр у змінну `term`, причому враховуємо, що очікувані параметри мають тип `double`. Для наочності, щоб перевіряти кожен поточний параметр, виводимо його на екран. Змінну `term` додаємо у змінну `sum` та збільшуємо значення лічильника `count`. У тілі функції `main` функцію `f` викликаємо тричі – з 3, 4 та 10 параметрами дійсного типу. Результат показано на рис. 1:

```

term #1 = 2      term #2 = 4      term #3 = 3
Average = 3
term #1 = 2      term #2 = 14     term #3 = 8      term #4 = 16
Average = 10
term #1 = 1.5    term #2 = 1.5    term #3 = 2.8    term #4 = 3.4    term #5 = 2
term #6 = 14     term #7 = 8      term #8 = 16     term #9 = 4.5    term #10 = 2.3
Average = 5.6
-

```

Рис. 1 Виклик функції з різною кількістю параметрів

2.3 Аргументи функції main

Мова С передбачає можливість передачі аргументів головній функції, запущеній на виконання програми – функції `main()` – за допомогою використання командного рядка. Відомо, що функція `main()` служить диспетчером, що організовує взаємодію між усіма іншими функціями у програмі. Зазвичай ця функція застосовується без параметрів, однак може викликатися з рядом параметрів, які передаються з командного рядка. Список формальних параметрів функції `main()` є фіксованим, однак за його допомогою можна задавати список аргументів, кількість яких обмежується лише можливостями операційної системи. Аргументи функції `main()` повинні вказуватися у командному рядку одразу після імені виконуваного модуля (файла з розширенням `.exe`). У ролі аргументів функції `main()` доцільно передавати імена файлів, функцій, текст, що задає режим роботи програми, а також самі дані (числа)

С підтримує три параметри функції `main()`. Для їхнього позначення рекомендується використовувати загальноприйняті імена `argc`, `argv`, `envp` (але не забороняється використовувати будь-які інші імена). Заголовок функції `main()` при використанні командного рядка має вигляд:

```
int main(int argc, char *argv[], char *envp[])
```

Перший параметр `argc` є цілим числом, що задає кількість аргументів, вказаних у командному рядку. Ім'я програми вважається першим аргументом. Тому у випадку, якщо користувач не задав жодного аргументу, значення `argc` становитиме 1.

Параметр `argv` є вказівником на масив символічних вказівників, які посилаються на аргументи командного рядка. Слід мати на увазі, що всі аргументи командного рядка є рядками, а тому при потребі їх слід перетворювати у числа за допомогою відповідних функцій. Самі аргументи в командному рядку повинні розділятися пробілами або знаками табуляції, коми, крапки та інші символи не розглядаються як розділювачі. Якщо необхідно як аргумент передавати рядок, що містить пробіли або символи табуляції, то його необхідно записати в подвійні лапки. Останнім елементом масиву вказівників є нульовий вказівник (NULL).

Параметр `env` є масивом вказівників на рядки оточення операційної системи. Його реалізація залежить від конкретного середовища. Отримати вказівники на рядки оточення можна за допомогою функції `getenv()`, а задати їх – за допомогою функції `putenv()`. Обидві ці функції визначені у заголовному файлі `stdlib.h`.

```
/* **** Застосування аргументів функції main **** */
#include <stdio.h>
#include <conio.h>

int solvelin(double, double);

int main(int argc, char* argv[])
{
    printf("There are %d arguments\n", argc);
    if ( ! (argc==4) )
    {
        printf("Error - not enough or too many arguments\n");
        _getch();
        return 1;
    }
    solvelin(atof(argv[2]), atof(argv[3]));
    _getch();
    return 0;
}

int solvelin(double var1, double var2)
{
```

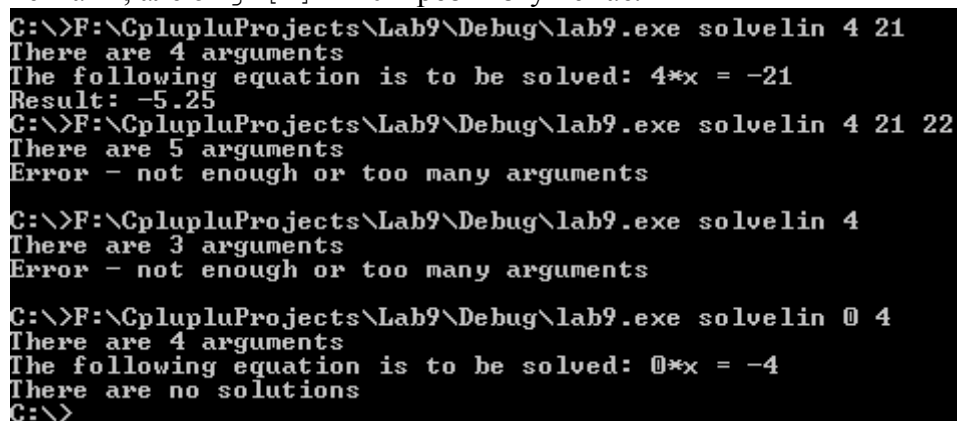


```

printf("The equation to be solved is:\t%lf*x = %lf\n", var1, -var2);
if(var1) printf("Result: %lf", -var2/var1);
else printf("There are no solutions");
return 0;
}

```

У даній програмі в функції `main` виводиться на екран кількість аргументів, введених з командного рядка. Якщо кількість аргументів не рівна 4, то виводиться повідомлення про те, що кількість аргументів або недостатня, або надмірна. У цьому випадку робота функції `main` припиняється оператором `return`. Інакше викликається користувацька функція `solvelin`, що приймає два параметри `var1` і `var2` типу `double` і служить для розв'язання лінійного рівняння $var1 \cdot x = -var2$, за умови, що $var1 \neq 0$ (інакше виводиться повідомлення про те, що розв'язків немає). Функції `solvelin` передаються параметри `atof(argv[2])` та `atof(argv[3])`. Передбачається, що користувач з командного рядка вводить назву виконуваного модуля `lab9.exe` (це буде значення `argv[0]`), назву функції `solvelin` (це буде `argv[1]`) та представлення двох дійсних чисел (вони і визначатимуться значеннями `argv[2]` та `argv[3]`). Оскільки всі задані з командного рядка аргументи є рядками символів, то необхідне перетворення типів – `argv[2]` та `argv[3]` приводяться до дійсного типу за допомогою функції `atof`. На рис. 2 показано вікно програми `cmd.exe` (запущеної через `Start=>Run`), де викликається виконуваний модуль `lab9.exe` з різними аргументами. При цьому вказується повний шлях до файлу `lab9.exe`. Як видно з рис. 2, спочатку `lab9.exe` викликається з "правильною" кількістю аргументів – 4. Результатом є виклик функції `solvelin`. Далі `lab9.exe` викликається спочатку з недостатньою, а тоді з надмірною кількістю аргументів, про що видається відповідне повідомлення. Зрештою, `lab9.exe` викликається з чотирма аргументами, але `argv[2] = 0` – розв'язку немає.



```

C:\>F:\CplupluProjects\Lab9\Debug\lab9.exe solvelin 4 21
There are 4 arguments
The following equation is to be solved: 4*x = -21
Result: -5.25
C:\>F:\CplupluProjects\Lab9\Debug\lab9.exe solvelin 4 21 22
There are 5 arguments
Error - not enough or too many arguments

C:\>F:\CplupluProjects\Lab9\Debug\lab9.exe solvelin 4
There are 3 arguments
Error - not enough or too many arguments

C:\>F:\CplupluProjects\Lab9\Debug\lab9.exe solvelin 0 4
There are 4 arguments
The following equation is to be solved: 0*x = -4
There are no solutions
C:\>

```

Рис. 2 Виклик функції `main` з командного рядка з різними аргументами

Одним з критеріїв успішності виконання програми для оцінки операційною системою є код повернення програми (код помилки). Це ціле число, яке вказує на наявність певної проблеми або її відсутність при виконанні програми. Якщо функція `main()` описується з типом повернення `int`, то значення при операторі `return` в цій функції передаватиметься операційній системі як результат виконання програми (0 – успішний). Якщо ж функція `main()` описана з типом повернення `void`, то компілятор генерує код, який завжди вказуватиме операційній системі на успішне виконання програми.

2.4. Препроцесор. Основні директиви препроцесора та компілятора.

Першим етапом підготовки програми при перетворенні компілятором вихідного тексту у виконуваний код є обробка її *препроцесором* (або *макропроцесором*). Препроцесор обробляє текст програми і повертає його нову редакцію. У більшості систем програмування *препроцесор* сполучений з *транслятором*, і для програміста його робота прозора. Хоча при потребі препроцесор можна викликати окремо без компіляції тексту. Препроцесор виконує лише обробку тексту. Це означає, що він "не розуміє" операторів мови програмування, "не знає" про змінні, константи програми, а розрізняє тільки константи та змінні і виконує тільки оператори

макромови – вирази у програмі, що адресовані макропроцесору. При цьому у проміжному тексті програми (результат роботи препроцесора) останні будуть відсутні і для подальших етапів обробки "не видні". Так, якщо препроцесор замінив у програмі деякий текст А на текст В, то транслятор уже бачить тільки текст В, і не знає, був цей текст написаний програмістом "своєю рукою" чи підставлений препроцесором.

Директиви (команди) препроцесора зазвичай використовуються для полегшення внесення змін до початкової програми, а також для полегшення її компіляції для різних середовищ виконання.

Всі директиви препроцесора в мові С починаються із символу #. Він повинен бути першим непорожнім символом в рядку (символи пробілу і табуляції ігноруються). Для зручності читання коду директиви препроцесора прийнято починати писати з першої колонки рядка.

Нижче наведено основні команди препроцесора та їх короткий опис.

#include	#ifndef	#endif
#define	#if	#error
#undef	#else	#pragma
#ifdef	#elif	

Директива #include має наступний формат:

```
#include "filename"
```

або

```
#include <filename>
```

Використовується для вставлення усього вмісту деякого файлу з ім'ям (шляхом) filename у місце виклику директиви. Якщо ім'я файлу взяте в подвійні лапки, то пошук файлу відбувається в локальних папках проекту (задаються в параметрах компілятора, за замовчуванням – папка, в якій розміщений опрацьовуваний файл). Якщо ж ім'я файлу взяте в кутові дужки, препроцесор шукатиме його в папці для стандартної бібліотеки (залежить від налаштувань компілятора). Директива #include, як правило, використовується для підключення заголовкових файлів стандартної бібліотеки (таких як stdio.h, stdlib.h) та користувацьких заголовних файлів для великих програм, які складаються з декількох (багатьох) файлів, що компілюються разом в одну програму.

Директива #define має наступний формат:

```
#define IDENTIFIER text
```

або

```
#define IDENTIFIER(params) text
```

Використовується для заміни часто вживаних в тексті програми констант, ключових слів, операторів або виразів деякими ідентифікаторами. В останніх двох випадках такі ідентифікатори називаються макровизначеннями або просто макросами. Макроси можуть приймати параметри. Формат макроса проілюстровано другим. Директива #define заміняє всі наступні входження ідентифікатора (в ілюстрації – IDENTIFIER) на вказаний текст (в ілюстрації – text). Текстом може бути будь-який фрагмент програми на мові С або пустий фрагмент (відсутність тексту як такого). Для задання багаторядкового тексту заміни можна використати символ \ вкінці кожного рядка, окрім останнього.

Директива #undef має наступний формат:

```
#undef IDENTIFIER
```

Використовується для скасування дії директиви #define для вказаного ідентифікатора. Після такого рядка всі наступні входження IDENTIFIER не будуть замінюватися.

Препроцесор С містить декілька вбудованих макроімен (ідентифікаторів), серед яких варто виділити такі:

- __LINE__ – номер рядка у вихідному файлі (заміняється номером, де розміщений),
- __FILE__ – ім'я опрацьовуваного файлу,
- __DATE__ – дата початку опрацювання файлу препроцесором,
- __TIME__ – час початку процесу опрацювання.

Вбудовані макроімена не можна оголошувати в #define і скасовувати у #undef.

Директиви умовної компіляції

Директиви умовної компіляції використовуються для поділу тексту програми на частини, одні з яких будуть компілюватися, а інші пропускатимуться компілятором залежно від деяких умов.

Формат:

```
#if CONST_EXPRESSION
#ifdef IDENTIFIER
#ifndef IDENTIFIER
```

Для `#if` обчислюється константний вираз `CONST_EXPRESSION`, і якщо результат обчислень – істина (ненульовий), то компілюються всі рядки програми аж до директиви `#else`, `#elif` або `#endif`. Для `#ifdef` та `#ifndef` компілюється відповідна частина коду, якщо визначено (не визначено) ідентифікатор `IDENTIFIER`.

У константному виразі директиви `#if` може бути використана операція `defined` для перевірки, чи був визначений зазначений ідентифікатор. Обернену умову перевіряє операція `!` `defined`. Ранні версії препроцесора не мають операції `defined`, а працюють тільки з `#ifdef` та `#ifndef`. Ці директиви були збережені для сумісності.

Вказані вище директиви можуть мати розділ "інакше", який визначається в одній з двох форм:

```
#else
#elif CONST_EXPRESSION2
```

Для `#else` відповідна послідовність коду компілюється в тому випадку, якщо одна з вимог, вказана в `#if` не виконується.

`#elif` аналогічна `#else` за винятком того, що має виконуватися умова, яка задається константним виразом `CONST_EXPRESSION2`.

Вказані вище директиви мають закінчуватися директивою `#endif`.

Директива `#error` зупиняє роботу компілятора і видає повідомлення про помилку.

Директива `#line` константа "ім'я файлу" змушує компілятор вважати, що константа задає номер наступного рядка вихідного файлу, і поточний вхідний файл іменується ідентифікатором. Якщо ідентифікатор відсутній, то ім'я файлу не змінюється.

Директива `#pragma` – це директива препроцесора, яка реалізує можливості компілятора. Ці особливості можуть бути пов'язані з типом компілятора. Різні типи компіляторів можуть підтримувати різні директиви. Загальний вигляд директиви:

```
#pragma команда компілятора
```

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Як одновимірний масив передати у функцію?
2. Чому при передачі двовимірного масиву у ролі аргумента функції потрібно вказати значення другої розмірності масиву?
3. Як заборонити модифікацію елементів масиву в тілі функції?
4. Що розуміється під терміном “функції зі змінною кількістю параметрів”?
5. Що таке конструкція еліпсису? Для чого вона призначена?
6. Опишіть відомі Вам способи задання довжини змінної частини списку формальних параметрів функції.
7. Яким чином організована передача аргументів у функції в мові C?
8. Чи можна в C створювати функції, список параметрів яких містить лише змінну частину? Як Ви думаєте, чому?
9. Які макроси мови C використовуються для роботи зі списками параметрів змінної довжини?
10. Опишіть синтаксис, призначення та приклад використання макросу `va_arg`?

11. Скільки параметрів може мати функція `main()`? Як передати аргументи головній функції програми на мові C?
12. Як обробляються в функції `main()` параметри командного рядка?
13. Що таке препроцесор?
14. Поясніть призначення макросів.
15. Що таке іменована константа?
16. В чому полягає суть умовної компіляції?

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. Розробити алгоритм розв'язання індивідуального завдання і подати його у вигляді блок-схеми.
4. Скласти програму на мові C у відповідності з розробленим алгоритмом.
5. Виконати обчислення по програмі.
6. Одержати індивідуальне завдання з Додатку 2.
7. Розробити алгоритм розв'язання індивідуального завдання і подати його у вигляді блок-схеми.
8. Скласти програму на мові C у відповідності з розробленим алгоритмом.
9. Виконати обчислення по програмі при різних значеннях точності і порівняти отримані результати.
10. Одержати індивідуальне завдання з Додатку 3.
11. Скласти програму на мові C для реалізації індивідуального завдання.
12. Підготувати та здати звіт про виконання лабораторної роботи.

5. СПИСОК ЛІТЕРАТУРИ

1. Керниган Б., Ритчи Д. Язык программирования C. - М. - Финансы и статистика. - 1992. - 272 с.
2. Уэйт М., Прата С., Мартин Д. Язык C. Руководство для начинающих. - М. - Мир. - 1988. - 512 с.
3. К. Джамса. Учимся программировать на языке C. М.: Мир, 1997. - 320 с
4. Герберт Шилдт. Полный справочник по C. 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.

1. 6. ІНДИВІДУАЛЬНІ ЗАВДАННЯ

ДОДАТОК 1

Написати програму для обробки даних, організованих у масив, згідно завдання наведеного варіанту.

1. Написати функцію для обчислення суми елементів квадратної матриці, що розташовані вище побічної діагоналі. З її допомогою знайти мінімальне значення такої суми в k матрицях.
2. Написати функцію для обміну рядків двовимірного масиву з її допомогою відсортувати масив по елементах третього стовпчика.
3. Написати функцію для знаходження визначника квадратної матриці порядку 3. З її допомогою знайти серед заданих матриць таку, визначник якої найбільший.
4. Розробити функцію, яка вилучає з речення слово, задане своїм порядковим номером (якщо таке слово є в реченні). На основі розробленої функції вилучити друге та шосте слово з введеного речення.

5. Написати функцію для додавання двох матриць. З її допомогою додати вихідну матрицю і транспоновану.
6. Задано три матриці A,B,C цілих чисел розмірності $n \times m$. Для кожної матриці знайти середнє арифметичне додатніх елементів і вивести на друк ту матрицю, для якої це значення буде найбільшим.
7. Написати функцію визначення скалярного добутку двох векторів. З її допомогою визначити чи є введена матриця ортонормована, тобто така, що скалярний добуток кожної пари різних рядків дорівнює 0, а скалярний добуток рядка самої на себе дорівнює 1.
8. Задано A,B,C – квадратні дійсні матриці n -ого порядку. Вивести на друк ту з них, норма якої найбільша. За норму матриці взяти найбільший по абсолютній величині елемент.
9. Написати функцію обміну стовпчика і рядка квадратної матриці. З її допомогою поміняти місцями ті рядки і стовпчики, перші елементи яких збігаються.
10. Написати функцію транспонування квадратної матриці. З її допомогою визначити чи є задана матриця симетричною. (Матриця називається симетричною, якщо транспонована матриця дорівнює вихідній).
11. Написати функцію, для пошуку максимального елемента в зазначеному рядку двовимірного масиву. Зсунути в двовимірному масиві всі рядки циклічно вправо на кількість елементів рівну максимальному елементу в цьому рядку.
12. Задано три матриці A,B,C цілих чисел розмірності $n \times m$. Обчислити:

$$\frac{\|A\| + \|B\| + \|C\|}{\|A+B+C\|}$$
, де $\|\cdot\|$ - норма матриці, яка рівна сумі максимальних елементів кожного рядка матриці.
13. Розробити функцію, яка визначає найдовше слово у заданому символьному рядку і повертає довжину цього слова. На основі розробленої функції визначити і надрукувати найдовше слово серед усіх слів групи введених речень.
14. Написати функцію для обміну стовпців матриці. З її допомогою відсортувати матрицю по елементах другого рядка.
15. Задано дві матриці A,B цілих чисел розмірності $n \times m$. Поміняти місцями максимальні та мінімальні елементи цих матриць. Для вводу матриць використати функцію вводу двовимірного масиву.
16. Написати функцію для обчислення суми елементів квадратної матриці, що розташовані нижче головної діагоналі. З її допомогою знайти максимальне значення такої суми в n заданих матрицях.
17. Написати функцію $sum(x, y, z)$, яка присвоює вектору z суму векторів x та y і за її допомогою обчислити $d = a + b + c$, де a, b, c - це одновимірні масиви дійсних чисел розмірності n , для введення елементів яких також написати окрему функцію.
18. Написати функцію, яка замінює у заданому символьному рядку всі слова, що містять вказану літеру на відповідну кількість символів '*'. Використовуючи розроблену функцію, "закодувати" всі слова з введених речень, в яких зустрічається задана користувачем літера.
19. Написати функцію, яка формує з заданого речення нове з інверсним порядком слів. На основі розробленої функції надрукувати введені речення зі зворотним порядком слів.
20. Написати функцію, що перевіряє чи є нульові елементи в зазначеному стовпці двовимірного масиву. Видалити з масиву всі стовпці з нульовими елементами, а видалений стовпець заповнюється 0 і переноситься в кінець масиву.
21. Написати функцію, що перевіряє чи сума елементів головної діагоналі двовимірного масиву більша за суму елементів побічної діагоналі. На основі цієї функції перевірити 5 заданих матриць.
22. Написати функцію, яка перевіряє чи матриця є розрідженою. З її допомогою визначити кількість розріджених матриць серед введених. (Матриця вважається розрідженою, якщо кількість нульових елементів перевищує 2/3 кількості елементів).

23. Написати функцію, яка обчислює суму елементів під головною діагоналлю. З її допомогою знайти серед введених матриць такі, сума елементів яких під головною діагоналлю рівна 0.
24. Написати функцію визначення скалярного добутку двох векторів. З її допомогою визначити чи є введена матриця ортогональною, тобто така, що скалярний добуток кожної пари різних рядків дорівнює 0, а скалярний добуток рядка самої на себе відмінний від нуля.
25. Написати функцію, яка визначає, чи входить до складу заданого символьного рядка вказане користувачем ключове слово. На основі розробленої функції визначити, в якому з речень найбільше разів зустрічається заданий ключовий набір символів.

ДОДАТОК 2

1. У функцію зі змінним числом параметрів надходять рядки, кінець списку – вказівник NULL. Порахувати кількість переданих рядків та вивести їх на екран. (оскільки аргументами є рядки, то в стек записуються адреси рядків).
2. У функцію зі змінним числом параметрів надходять додатні цілі числа, кількість яких задана першим параметром. Знайти і вивести найбільший спільний дільник цих чисел.
3. У функцію зі змінним числом параметрів надходять рядки, кінець списку - вказівник NULL. Розбити кожний рядок на слова та вивести їх на екран. (оскільки аргументами є рядки, то в стек записуються адреси рядків).
4. У функцію зі змінним числом параметрів надходять символи, кінець списку – нуль-символ '\0'. Знайти і вивести символ, який найчастіше зустрічається.
5. У функцію зі змінним числом параметрів надходять додатні цілі числа, кінець списку – значення -1. Порахувати, скільки разів зустрічається кожна цифра у заданому числі.
6. У функцію зі змінним числом параметрів надходять цілі числа, кількість яких задана першим параметром. Знайти і вивести кількість додатніх чисел переданих у функцію.
7. У функцію зі змінним числом параметрів надходять слова, кінець списку - вказівник NULL. Знайти і вивести найдовше слово.
8. У функцію зі змінним числом параметрів надходять дійсні числа, кількість яких задана першим параметром. Знайти і вивести найбільше з них.
9. У функцію зі змінним числом параметрів надходять додатні цілі числа, кінець списку – значення -1. Знайти і вивести те число, сума цифр якого рівна кількості переданих чисел.
10. У функцію зі змінним числом параметрів надходять слова, кінець списку - вказівник NULL. Знайти і вивести всі слова зі списку, які складаються з 5 літер.
11. У функцію зі змінним числом параметрів надходять дійсні числа, кількість яких задана першим параметром. Знайти і вивести кількість додатніх та від'ємних чисел.
12. У функцію зі змінним числом параметрів надходять послідовності чисел Фібоначі, кінець списку – значення -1. Знайти і вивести перше натуральне число, яке більше за останнє число в цій послідовності.
13. У функцію зі змінним числом параметрів надходять додатні цілі числа, кінець списку – значення -1. Знайти і вивести всі прості числа, якщо вони відсутні, то вивести відповідне повідомлення.
14. У функцію зі змінним числом параметрів надходять слова, кінець списку - вказівник NULL. Знайти і вивести всі слова, в яких зустрічається подвоєння літер, якщо таких слів немає, то вивести відповідне повідомлення.
15. У функцію зі змінним числом параметрів надходять рядки, кінець списку - вказівник NULL. Порахувати кількість слів у кожному з переданих рядків (оскільки аргументами є рядки, то в стек записуються адреси рядків).
16. У функцію зі змінним числом параметрів надходять цілі числа, кількість яких задана першим параметром. Для всіх додатніх чисел вивести їх двійковий код.
17. У функцію зі змінним числом параметрів надходять символи, кінець списку – нуль-символ '\0'. Знайти і вивести кількість голосних літер.

18. У функцію зі змінним числом параметрів надходять додатні цілі числа, кінець списку – значення -1. Знайти і вивести всі числа, для яких у двійковому коді кількість нулів менша за кількість одиниць.
19. У функцію зі змінним числом параметрів надходять дійсні числа, кількість яких задана першим параметром. Знайти і вивести ті з них, в яких перша цифра дробової частини рівна останній цифрі цілої частини.
20. У функцію зі змінним числом параметрів надходять додатні цілі числа, кінець списку – значення -1. Перевірити чи утворюють вони послідовність чисел Фібоначі.
21. У функцію зі змінним числом параметрів надходять цілі числа, кінець списку – значення -1. Перевірити чи утворюють вони зростаючу послідовність.
22. У функцію зі змінним числом параметрів надходять слова, кінець списку - вказівник NULL. Знайти і вивести всі слова, які читаються однаково спочатку в кінець і з кінця на початок.
23. У функцію зі змінним числом параметрів надходять символи, кінець списку – нуль-символ '\0'. Знайти і вивести символ, який зустрічається найрідше.
24. У функцію зі змінним числом параметрів надходять цілі числа, кінець списку – значення -1. Знайти і надрукувати всі парні числа, які не містять цифри 7. Знайти їх кількість або вивести повідомлення про їх відсутність.
25. У функцію зі змінним числом параметрів надходять цілі числа, кількість яких задана першим параметром. Знайти і вивести кількість додатніх парних чисел переданих у функцію.

ДОДАТОК 3

1. Використовуючи той самий ідентифікатор оголосити числовий і символьний масиви. Для числового масиву визначити кількість повторень кожного числа. Для символьного масиву знайти позицію і значення символу за заданим кодом символу. Використовувати умовну компіляцію.
2. Здійснити перевірку введених з клавіатури чисел на парність і позитивність. Оформити у вигляді макросів визначення парності, позитивності, вивід діагностичних повідомлень. За допомогою умовної компіляції розділити процеси визначення парності і позитивності чисел.
3. Використовуючи макроси `__FILE__`, `__LINE__`, вивести інформаційні повідомлення. Сформувати довільне повідомлення про програмну помилку.
4. Використовуючи той самий ідентифікатор оголосити цифровий і символьний масиви. Для цифрового масиву знайти значення максимального елемента. Для символьного масиву знайти кількість слів (слова розділяються пробілами). Використовувати умовну компіляцію.
5. Задати макрос обчислення n-ого члена геометричної прогресії за введеними користувачем першим членом і знаменником.
6. Задати макрос обчислення периметра еліпса за відомими радіусами a та b.
7. Використовуючи умовну компіляцію, обчислити значення змінної. Функції `max`, `min` і піднесення до степені виконати у вигляді макросів.

$$w = \begin{cases} \max(x^2 + y + z, xyz), & 0 < a < 5, \\ \min(x^2 + (y + z)^2, xy), & a \leq 0, a \geq 5. \end{cases}$$

8. Використовуючи умовну компіляцію, обчислити значення змінної. Функції `max`, `min`, одержання модуля і піднесення до степені виконати у вигляді макросів.

$$w = \begin{cases} \max(|x| + |y| + |z|, x^3), & 0 < a < 3, \\ \min(2x + y, (2y - z)^2), & a > 3. \end{cases}$$

9. Продемонструвати особливості застосування опції `comment` директиви `#pragma`.
10. Використовуючи умовну компіляцію, обчислити значення змінної. Функції `max`, `min`, одержання модуля і піднесення до степені виконати у вигляді макросів.

$$w = \begin{cases} \min(x, z) * \max(x + z, x * z), & 5 < a < 10, \\ \max^2(|x - z|, z), & a \leq 5, z \geq 10. \end{cases}$$

11. Використовуючи умовну компіляцію, обчислити значення змінної. Функції \max , \min , одержання модуля, піднесення до степені, одержання кореня виконати у вигляді макросів.

$$w = \begin{cases} \max((x - y)^2, |y - x|), & a = 1, \\ \min(x + y^2, \sqrt{x + y}), & a \neq 1. \end{cases}$$

12. Використовуючи той самий ідентифікатор, оголосити числовий і символьний масиви. Для числового масиву знайти значення мінімального елемента. Для символьного масиву визначити символ, що найчастіше зустрічається. Використовувати умовну компіляцію.
13. Використовуючи умовну компіляцію, обчислити значення змінної. Функції \max , \min , піднесення до степені, одержання кореня виконати у вигляді макросів. Умову вибору задати самостійно.

$$w = \begin{cases} \sqrt{x - y + \max((x - z)^2, x - y)} \\ \max(xy, xz) + \min(x - y, y - z) \end{cases}$$

14. Використовуючи умовну компіляцію, обчислити значення змінної. Умову вибору задати самостійно.

$$w = \begin{cases} ((x \wedge y) \wedge (x \vee z)) \wedge (!x \oplus y) \\ ((x \vee y) \wedge (z \vee y)) \wedge (z \vee x) \end{cases}$$

15. Продемонструвати особливості використання директиви `#line` в обох режимах.
16. Використовуючи умовну компіляцію, обчислити значення змінної. Функції \max , \min , взяття модуля, піднесення до степені виконати у вигляді макросів. Умову вибору задати самостійно.

$$w = \begin{cases} \max(x^3, (x + y)^2) * \min(x - y, y - z), \\ \min(|x - z|, |x + z|). \end{cases}$$

17. Використовуючи умовну компіляцію, обчислити значення змінної. Функції \max , \min , піднесення до степені виконати у вигляді макросів. Умову вибору задати самостійно.

$$w = \begin{cases} \max^2(x, y, z) - 2 * \min(x, y, z), \\ \sin^2(x) + \max(x, y, z) / \min(x, y, z). \end{cases}$$

18. Продемонструвати особливості застосування опцій `startup` та `exit` директиви `#pragma`.
19. Розв'язати квадратне рівняння, використовуючи макроси для задання формули дискримінанту, коренів рівняння і виведення відповіді на екран.
20. Продемонструвати особливості використання директиви `#error`.
21. Задати макрос обчислення площі кола за введеним користувачем радіусом. Дослідити макрос при використанні постфіксних і префіксних операцій.
22. Задано два натуральних числа n, m ($n, m \leq 100$), якщо умова виконується, знайти середнє арифметичне чисел, інакше зупинити компіляцію за допомогою директиви `#error`.
23. Продемонструвати роботу операції `##`.
24. Знайти найменше натуральне число Q таке, що добуток його цифр дорівнює заданому числу N . Добуток обчислювати за допомогою макросів з параметрами.
25. Задані перший і другий елементи арифметичної прогресії. Знайти n -й елемент прогресії. При обчисленнях використовувати макроси з передачею параметрів.