**ПРАКТИЧНА РОБОТА № 6.**
**Оператори циклу, директиви препроцесора**
**та форматований ввід-вивід у мові С**

**Мета роботи**

Ознайомитися з директивами препроцесора мови С, з операторами циклу і функціями вводу-виводу.

**Теоретичні відомості**

3.2.1.Директиви препроцесора

Препроцесор мови С використовується для обробки тексту програми до її компіляції. Препроцесор виконує макропідстановку, умовну компіляцію, під'єднання іменованих файлів. Директиви препроцесора починаються з символу "**#**".

За допомогою директиви препроцесора **#include** в програму на мові С можна включити текст будь-якого файлу. Директива **#include** має дві форми:
**#include <ім'я файлу>** – під'єднання стандартного файлу:

```
#include <stdio.h>
#include <math.h>
```

**#include "<ім'я файлу>"** – під'єднання зовнішнього не стандартного файлу:

```
#include "myfile.h".
```

Суфікс "**h**" використовується для файлів, які під'єднуються в заголовку програми. Це, так звані, заголовочні (*header*) файли.

Макровизначення

Директива **#define** ставить у відповідність ідентифікатору текстову стрічку, тобто проводить деяке визначення. Синтаксис оператора:

**#define <ідентифікатор> <стрічка заміни>**

Стрічка заміни може містити ідентифікатори, ключові слова, розділювачі. Директива **#define** може стояти у будь-якому місці програми і виконує такі функції:

1) Визначення констант:

```
#define NULL 0
#define TRUE 1
#define FALSE 0
```

2) Прості макровизначення:

```
#define begin {
#define end }
```

тоді замість фігурних дужок будуть використовуватись слова **begin** і **end**.

3) Параметризація макровизначень:

```
#define MAX(x,y) ((x)>(y)?(x):(y))
#define MIN(x,y) ((x)>(y)?(y):(x))
```

Директива **#undef** – відміняє дію **#define**. Наприклад:

```
#include <stdio.h>
#define TWO 2
#define FOUR TWO*TWO
#define PX printf("x equal %d.\n",x)
int main()
{
    int x = TWO;
    PX;
    x = FOUR;
    PX;
    return 0;
}
```

У результаті роботи цієї програми одержимо повідомлення:

```
x equal 2.
x equal 4.
```

За директивою **#define** препроцесор замінює кожне макровизначення на стрічку заміни, тобто:

**int x = TWO** перетвориться в **int x = 2**;

**PX** перетвориться в **printf("x equal %d.\n",x)**

**x = FOUR** перетвориться в **x = TWO*TWO** і далі в **x = 2*2**

і так далі.

Умовна компіляція

Умовна компіляція це вибіркова компіляція лише тих частин програми, які задовольняють певні умови. Для умовної компіляції використовуються такі директиви препроцесора: **#if**, **#else**, **#endif**, **#ifdef**, **#ifndef**.

Синтаксис директиви умовної компіляції:

```
#if
<текстові рядки для випадку "істина">
#else
<текстові рядки для випадку "не істина">
#endif
```

**if** – заголовок містить умови, на основі яких здійснюється перевірка. Управляючий рядок **if** – заголовок має 3 форми:

```
#if <вираз, що має постійне значення>
#ifdef <ідентифікатор>
#ifndef <ідентифікатор>
```

В першій формі вираз визначається значенням нуль або не нуль (“істина”, “не істина”. В другій формі значення “істина” відповідає умові, якщо ідентифікатор був визначений в директиві **#define**. В третій формі значення “істина” відповідає умові, якщо ідентифікатор або не був визначений в директивою **#define**, або був відмінений директивою **#undef**.

Для прикладу умовної компіляції приведемо такий фрагмент програми:

```
#ifndef MAX_STK
# define MAX_STK 128
#endif
```

Ідентифікатор **MAX_STK** має значення по замовчуванню, якщо не буде заданий користувачем.

Директива **if** подібна до оператора **if** у мові С:

```
#if SYS == "IBM"
# include "ibm.h"
#endif
```

Якщо вираз **SYS=="IBM"** істина, то під’єднується файл “**ibm.h**”.

3.2.2.Оператори циклу у мові С

У мові С, як і в більшості інших існує три типи операторів циклу:

1) Оператор циклу з передумовою:

```
while(<вираз>) <оператор>;
```

2) Оператор циклу з постумовою:

```
do <оператор>;
while(<вираз>);
```

3) Оператор з параметрами:

```
for(<вираз 1>; <вираз 2>; <вираз 3>) <оператор>;
```

Наприклад, треба обчислити 5!. Фрагменти програм з операторами циклу будуть мати такий вигляд:

З оператором **while**:

```
int f = 1, n = 1;
while(n <= 5)
{
    f *= n;
    n++;
}
```

З оператором **do-while**:

```
int n = 1, f = 1;
do
{
   f *= n;
```

```
        n++;
    }
    while(n <= 5);
```

З оператором **for**:

```
    int f, n;
    for(f = 1, n = 1; n <=5 ; n++)
        f *= n;
```

Оператори циклу **while** виконуються до того часу поки виконується умова, тобто **n<=5**. Якщо умова не виконується, наприклад **n==8**, то оператори циклу **while** не виконуються ні разу.

Оператори циклу **do-while** також виконуються до того часу поки виконується умова. Але перевірка умови проводиться після першого виконання циклу, тобто якщо умова одразу не виконується, наприклад **n==8**, то оператори циклу **do-while** один раз будуть виконані.

Оператор циклу **for** можна подати в такому вигляді:

```
    for(<ініціалізація початкових значень>;
      <перевірка умови>;
      <зміна параметра>)
    <оператор>;
```

У прикладі надаємо початкові значення не тільки параметру циклу **n**, але і змінній **f**. Далі перевіряється умова виконання циклу **n<=5**, якщо умова виконується[1], то виконуються оператори циклу. Третій вираз це зміна параметра циклу. У нашому випадку **n=n+1** або **n++**. Оператор **for** має дуже гнучку структуру. Він може мати вкорочену форму, тобто:

```
    for(;n <= 5;)
    f *= n;
```

але тоді зміну **n** треба робити в тілі операторів циклу, а визначення початкового значення перед оператором **for**.

Допускається і така форма запису оператора **for**:

```
    int y = 1;
    for(int x = 1; y <= 15; y = 5 * x++)
        printf("%10d%10d\n", x, y);
```

---

[1] Часто в циклі **for** можна зустріти використання оператора *кома* “**,**”. Цей оператор, згідно зі стандарту є однією з *точок слідування* – спеціальних мість, наприклад, таких як оператор “**;**”, що гарантують виконання всіх побічних ефектів (оголошення, ініціалізація, виклик функцій, обчислення виразів і т.д.) у виразі ліворуч. Особивістю оператора “**,**” є те, що він відкидає результати виразу ліворуч і повертає у місце виклику результат виразу праворуч. Наприклад у конструкції **for(;a < 5, b > 10;){/*...*/}** обидва логічні вирази будуть обчислені, але до уваги буде прийматися тільки останній результат **b > 10**.

В результаті роботи цієї програми одержимо:

```
1          1
2          5
3         10
4         15
```

Тут перевіряється умова виходу по значенню **y**, а не **x**, а в виразі "зміна параметра" одночасно рахується значення **y** і **x** змінюється на 1. В мові С допускається вкладення циклів. Вкладеним називається цикл, що міститься всередині іншого циклу. Для ілюстрації приведемо програму, яка буде виводити на друк всі прості числа, що містяться між числом **2** і **num**:

```
#include <stdio.h>
int main()
{   int number, div, num, count = 0;
    printf("please input integer number > 2\n");
    scanf("%d", &num);
    printf("prime numbers between 2 and %d are:\n",
        num);
    for(number = 2; number <= num; number++)
    {   for(div = 2; number % div != 0; div++);
            if(div == number)
            {   printf("%5d", number);
                if(++count % 10 == 0)
                    printf("\n");
            }
    }
    return 0;
}
```

Якщо ввести ціле число 100, то в результаті роботи програми одержимо прості числа в діапазоні від 2 до 100:

```
please input integer number > 2
100
prime numbers between 2 and 100 are:
    2    3    5    7   11   13   17   19   23   29
   31   37   41   43   47   53   59   61   67   71
   73   79   83   89   97
```

### 3.2.3. Управляючі оператори **break**, **continue** і **goto**

Оператор **break** здійснює негайний вихід з операторів циклу або оператора **swich**. Управління передається наступному оператору після оператора з якого здійснювався вихід. Якщо оператор **break** стоїть всередині

вкладеного циклу, то вихід здійснюється тільки із внутрішньої структури, тобто тільки з того циклу в якому є оператор **break**.

Оператор **continue** – передає управління на кінець тіла циклу, всередині якого він знаходиться. Тобто пропускає частину ітерації, яку виконує і переходить до наступної ітерації. Наприклад, треба знайти суму додатних чисел, що вводяться з клавіатури:

```
#include <stdio.h>
int main()
{
    int n, s = 0, x;
    printf("please input the number of elements:\n");
    scanf("%d",&n);
    printf("please input %d integer numbers:\n", n);
    for(int i = 0; i < n; i++)
    {
        scanf("%d",&x);
        if(x < 0) continue;
        s += x;
    }
    printf("the sum of positive numbers is %d\n", s);
    return 0;
}
```

Результати:

```
please input the number of elements:
5
please input 5 integer numbers:
1 2 -3 -4 5
the sum of positive numbers is 8
```

Оператор **goto**:

```
goto <мітка>
```

де: “мітка” – це мітка оператора на який здійснюється перехід. Міткою може бути будь-який ідентифікатор, після якого стоїть символ двокрапка “**:**”.

Мова С володіє такими засобами, що використовувати оператор **goto** немає потреби. Єдиний випадок коли можна використовувати оператор **goto** це вихід із внутрішнього, вбудованого циклу у випадку знаходження помилки:

```
for(i = 0; i < n; i++)
     for(j = 0; j < m; j++)
          if(a[i] == b[j]) goto err;
...
err: printf("Error!\n");
```

### 3.2.4.Специфіка використання операторів **break** і **continue**.

Оператор **break**, який стоїть в тілі циклу, негайно припиняє виконання циклу й передає керування на рівень вище, а точніше, на наступний оператор, що стоїть після даного циклу, який містить **break**. Тому для припинення виконання багаторівневого циклу по "ініціативі" на найглибшому рівні доводиться виконувати не один, а декілька операторів **break**. Наприклад треба знайти суму перших додатних чисел, що вводяться з клавіатури:

```
#include <stdio.h>
int main()
{
    int n, s = 0, x;
    printf("please input positive numbers:\n");
    while(1)
    {   scanf("%d",&x);
        if(x < 0) break;
        s += x;
    }
    printf("the sum of positive numbers is %d\n", s);
    return 0;
}
```

Результати:

```
  please input positive numbers:
  1 2 3 4 5 0 -2
  the sum of positive numbers is 15
```

Ще один приклад використання оператора **continue**:

```
#include <stdio.h>
int main()
{
    char ch;
    while((ch = getchar())!= '\n')
    {   if(ch == '*') continue;
        putchar(ch);
    }
    return 0;
}
```

У даному прикладі у випадку вводу символу ***** керування програми передається знову на початок циклу. При цьому оператор **putchar(ch)** ігнорується. Якщо в цьому випадку використовувати оператор **break** замість **continue**, то ввід символу ***** аналогічний вводу символу переводу рядка, тобто виходу із циклу.

Функція **getchar()** читає символ з клавіатури, а функція **putchar()** – виводить його на екран.

### 3.2.5.Форматований ввід-вивід

Функції **printf()** і **scanf()** виконують форматований ввід-вивід на консоль, інакше кажучи, вони можуть зчитувати й записувати дані в заданому форматі, Функція **printf()** виводить дані на консоль. Функція **scanf()**, навпаки, зчитує дані з клавіатури. Обидві функції можуть оперувати будь- якими вбудованими типами даних, включаючи символи, рядки та числа.

Функція **printf()** – *print formatted*

Прототип функції **printf()** виглядає таким чином:

```
int printf (const char  *<керуюча_стрічка>,     ...);
```

Функція повертає кількість записаних нею символів, а у випадку помилки – від'ємне число. Параметр **<керуюча_стрічка>** складається з елементів двох видів. По-перше, він містить символи, які виводяться на екран. По-друге, у нього входять специфікатори формату, що починаються зі знака відсотка, за яким слідує код формату. Кількість аргументів повинна співпадати з кількістю специфікаторів формату, вони попарно зрівнюються зліва направо. Наприклад:

```
printf("I love %c%s", 'C', "11!");
```

виведе на екран рядок:

```
I love C11!
```

Функція **printf()** допускає широкий вибір специфікаторів формату, зокрема:

| Код | Формат |
|---|---|
| **%c** | Символ |
| **%d, %i** | Десяткове ціле число зі знаком |
| **%e, %E** | Науковий формат |
| **%f** | Десяткове число із плаваючою комою |
| **%g, %G** | Залежно від того, який формат коротший, застосовується або **%e**, або **%f** |
| **%a, %A** | Шістнадцяткове число з плаваючою комою |
| **%o** | Вісімкове число без знаку |
| **%s** | Рядок символів |
| **%u** | Десяткове ціле число без знаку |
| **%x, %X** | Шістнадцяткове число без знаку (малі літери) |
| **%p** | Вказівник |
| **%n** | Вказівник на цілочисельну змінну. Специфікатор викликає присвоювання цій цілочисельній змінній кількість символів, виведених перед ним |
| **%%** | Знак **%** |

Вивід символів

Для виводу окремих символів використовується специфікатор **%c**. У результаті відповідний аргумент без змін буде виведений на екран. Для виводу рядків застосовується специфікатор **%s**.

Вивід чисел

Для виводу десяткових цілих чисел зі знаком застосовуються специфікатори **%d** або **%i**. Ці специфікатори еквівалентні. Одночасна підтримка обох специфікаторів обумовлена історичними причинами. Для виводу цілого числа без знака варто застосовувати специфікатор **%u**. Специфікатор формату **%f** дає змогу виводити на екран числа із плаваючою комою. Специфікатори **%e** та **%E** вказують функції **printf()**, що на екран виводиться аргумент типу **float** у науковому форматі. Числа, представлені в науковому форматі, виглядають так:

```
x.dddddE+/-yy
```

Якщо буква **E** повинна бути виведена як велика, варто використовувати специфікатор **%E**, а якщо як мала – **%e**.

Функція **printf()** може сама вибирати подання числа за допомогою специфікатора **%f** або **%e**, якщо замість них вказати специфікатори **%g** або **%G**. У цьому випадку функція сама визначить, який вид числа коротший. Специфікатор **%G** дає змогу вивести букву **E** як велику, а **%g** – як малу. Наступна програма демонструє ефект застосування специфікатора **%g**:

```
#include <stdio.h>
int main()
{
    double f;
    for(f = 1.0; f < 1.0e+10; f *= 10)
        printf("%g ", f);
    return 0;
}
```

У результаті на екрані з'являться такі числа:

```
1 10 100 1000 10000 100000 1e+06 1e+07 1e+08 1e+09
```

Вивід адрес

Якщо на екран необхідно вивести адресу, варто застосовувати специфікатор **%p**. Цей специфікатор формату змушує функцію **printf()** виводити на екран адресу, формат якої сумісний з типом адресації, прийнятої в комп'ютері. Наступна програма виводить на екран адресу змінної **sample**:

```
#include <stdio.h>
int sample;
int main()
```

```
{
    printf("%p", &sample);
    return 0;
}
```

Можливий результат:

```
  0x600a24
```

Специфікатор **%n**

Специфікатор формату **%n** відрізняється від всіх інших. Він змушує функцію **printf()** записувати у відповідну змінну кількість символів, уже виведених на екран. Специфікатору **%n** повинен відповідати цілочисельний вказівник. Після завершення функції **printf()** цей вказівник буде посилатися на змінну, у якій утримується кількість символів, виведених до специфікатора **%n**. Наприклад:

```
#include <stdio.h>
int main()
{
    int count;
    printf("It is %n test\n", &count);
    printf("%d", count);
    return 0;
}
```

Ця програма виведе на екран рядок "**It is  test**" і число **6**. Специфікатор **%n** зазвичай використовується для динамічного форматування.

Модифікатори формату

Багато специфікаторів формату мають свої модифікатори, які трохи змінюють їхній зміст. Наприклад, з їхньою допомогою можна змінювати мінімальну ширину поля, кількість цифр після десяткової коми, а також виконувати вирівнювання по лівому краю. Модифікатор формату вказується між символом відсотка і кодом формату. Кожен модифікатор формату може доповнюватися відповідно до прототипу:

```
%[прапорець][ширина][.точність][розширення]специфікатор
```

Прапорець

**–** вирівнювання ліворуч (за замовчуванням праворуч);

**+** примусово виводить знак числа, навіть якщо воно додатнє;

(пробіл) якщо число додатнє, виведе пробіл, якщо від'ємне – знак мінус;

**#** якщо використовується з **%o** та **%x** то примусово виводить перед числом **0** або **0x** відповідно. Якщо з **%a**, **%e**, **%f** чи **%g**, то примусово виводит символ ".", навіть якщо число ціле;

**0** доповнює число ліворуч ведучими нулями.

Модифікатор мінімальної ширини поля

Ціле число, розміщене між символом відсотка і кодом формату, задає мінімальну ширину поля. Якщо рядок виводу коротший, ніж потрібно, він доповнюється пробілами, якщо довший – рядок все рівно виводиться повністю. Наприклад:

```
#include <stdio.h>
int main()
{   float item = 10.12304;
    printf("%f\n", item);
    printf("%12f\n", item);
    printf("%012f\n", item);//leading zeros using flag
    return 0;
}
```

Ця програма виводить на екран наступні числа:

```
  10.123040
     10.123040
  00010.123040
```

Модифікатор мінімальної ширини поля найчастіше використовується для форматування таблиць. Програма, наведена нижче, створює таблицю квадратів і кубів чисел.

```
#include <stdio.h>
int main()
{   int i;
    for(i = 1; i <= 8; i++)
        printf("%8d %8d %8d\n", i, i*i, i*i*i);
    return 0;
}
```

У результаті на екран буде виведена наступна таблиця:

```
 1        1        1
 2        4        8
 3        9       27
 4       16       64
 5       25      125
 6       36      216
 7       49      343
```

Модифікатор точності

Модифікатор точності вказується після модифікатора ширини поля (якщо він є). Цей модифікатор складається із крапки, за якої слідує ціле число. Точний зміст модифікатора залежить від типу даних, до яких він застосовується.

Якщо модифікатор точності застосовується до чисел із плаваючою комою з

форматами **%f**, **%e** або **%E**, він означає кількість десяткових цифр після крапки. Наприклад, специфікатор формату **%10.4f** означає, що на екран буде виведене число, яке складається з дев'яти символів, чотири з яких розташовані після крапки, тобто, чотири цифри до крапки, сама крапка і до п'яти цифр перед крапкою.

Якщо модифікатор застосовується до специфікаторів формату **%g** або **%G**, він задає кількість значущих цифр.

Якщо модифікатор використовується для виводу рядків, він задає максимальну довжину поля. Наприклад, специфікатор **%5.7s** означає, що на екран буде виведений рядок, який складається як мінімум з п'яти символів, довжина якого не перевищує семи символів. Якщо рядок виявиться довшим, останні символи будуть відкинуті.

Якщо модифікатор точності застосовується до цілих типів, він задає мінімальну кількість цифр, з яких повинне складатися число. Якщо число складається з меншої кількості цифр, воно доповнюється ведучими нулями. Наприклад:

```
#include <stdio.h>
int main()
{
    printf("%.4f\n", 123.1234567);
    printf("%.8d\n", 1000);
    printf("%8.10s\n", "It is simple test.");
    return 0;
}
```

Ця програма виводить на екран наступні результати:

```
123.1235
00001000
It is simp
```

Модифікатор розширення

| розширення | специфікатор % | | |
|---|---|---|---|
| | **d i** | **u o x** | **f e g a** |
| (немає) | **int** | **unsigned int** | **double** |
| **hh** | **signed char** | **unsigned char** | |
| **h** | **short** | **unsigned short** | |
| **l** | **long** | **unsigned long** | |
| **ll** | **long long** | **unsigned long long** | |
| **L** | | | **long double** |

Функція **scanf()** – *scan formatted*

Функція є процедурою вводу. Вона може зчитувати дані всіх вбудованих типів і автоматично перетворювати числа у відповідний внутрішній формат.

Дана функція є повною протилежністю до функції **printf()**. Прототип функції **scanf()** має такий вигляд:

```
int scanf (const char  *<керуюча_стрічка>,  ...);
```

Функція повертає кількість змінних, яким вона успішно присвоїла значення. Якщо при читанні відбулася помилка, функція **scanf()** повертає константу **EOF**. Параметр **<керуюча_стрічка>** визначає порядок зчитування значень і присвоювання їх змінним, зазначеним у списку аргументів.

Керуюча стрічка складається із символів, розділених на три категорії:

- специфікатори формату;
- розділювачі;
- символи, що не є розділювачами.

Ввід чисел

Для вводу цілого числа використовуються специфікатори **%d** або **%i**. Специфікатор **%i** підтримує ввід вісімкових (починаються з **0**) та шістнадцяткових (починаються з **0x**) чисел. Для вводу числа із плаваючою комою, представленого в стандартному або науковому форматі, застосовуються специфікатори **%e**, **%f** або **%g**, а також **%a**, для вводу шістнадцяткових чисел з плаваючою комою. Використовуючи специфікатори **%o** або **%x**, можна вводити цілі числа, представлені у вісімковому або шістнадцятковому форматі відповідно. Наприклад:

```
#include <stdio.h>
int main()
{
    int i, j;
    scanf("%o%x", &i, &j);
    printf("%o %x", i, j);
    return 0;
}
```

Функція **scanf()** припиняє вводити числа, виявивши перший нечисловий символ.

Ввід цілих чисел без знаку

Для вводу цілих чисел без знаку застосовується модифікатор **%u**. Наприклад, фрагмент:

```
unsigned num;
scanf("%u", &num);
```

вводить ціле число без знаку і присвоює його змінній **num**.

Ввід окремих символів

Як вказувалося раніше, окремі символи можна вводити за допомогою функції **getchar()** або похідних від її функцій. Функцію **scanf()** також

можна застосовувати для цієї мети, використовуючи специфікатор **%c**. Однак, як і функція **getchar()**, функція **scanf()** використовує буферизований ввід, тому в інтерактивних програмах її застосовувати не слід.

Незважаючи на те що пробіли, знаки табуляції й символи переходу на новий рядок використовуються як розділювачі при читанні даних будь-яких типів, при вводі окремих символів вони зчитуються нарівні з усіма. Наприклад, якщо потік вводу містить рядок "**x y**", то фрагмент коду:

```
scanf("%c%c%c", &a, &b, &c);
```

присвоїть символ "**x**" змінній **a**, "пробіл" – змінній **b** і символ "**y**" – змінній **c**.

Ввід рядків

Функцію **scanf()** можна застосовувати для вводу рядків із вхідного потоку. Для цього використовується специфікатор **%s**. Він змушує функцію **scanf()** зчитувати символи, поки не виявиться розділювач. Символи, пораховані із вхідного потоку, записуються в масив, на який посилається відповідний аргумент, а в кінець цього масиву записується нульовий байт. Функція **scanf()** вважає розділювачем "пробіл", символ переходу на новий рядок "**\n**", символ табуляції "**\t**", символ вертикальної табуляції "**\v**", а також символ розриве сторінки "**\f**". Отже, функцію **scanf()** не можна просто застосувати для вводу рядка "**It is test**", оскільки ввід припиниться на першому ж пробілі. Наприклад:

```
#include <stdio.h>
int main()
{   char str[80];
    printf("Please input a string:\n");
    scanf("%s", str);
    printf("This is your string: %s", str);
    return 0;
}
```

Результати:

```
Please input a string:
It is test
This is your string: It
```

Модифікатори формату

Як і у функції **printf()**, для **scanf()** кожен модифікатор формату може доповнюватися відповідно до прототипу:

```
%[*][ширина][розширення]специфікатор
```

***** вказує на те, що інформація має бути прочитана, але не повинна записуватися в змінні, що розташовані після керуючої стрічки;

**ширина** максимальна кількість символів, що повинні бути прочитані;

**розширення**:

| розширення | специфікатор % d i | u o x | f e g a |
|---|---|---|---|
| (немає) | int* | unsigned int* | float* |
| hh | signed char* | unsigned char* | |
| h | short* | unsigned short* | |
| l | long* | unsigned long* | double* |
| ll | long long* | unsigned long long* | |
| L | | | long double* |

Крім того, для функції **scanf()** визначено специфікатор **[<символи>]** та **[^<символи>]**, що позначають, які символи повинні читатися, чи які повинні розглядатися як термінатори при читанні відповідно. Тому, існує можливість обійти незручність функції **scanf()** для вводу рстрічок з розділювачами явно вказуючи які вхідні символи слід розгядати як кінець вводу, за допомогою конструкції **%[^<символ 1><символ 2>...]**. Наприклад: **scanf( "%[^\n]", str);** – зчитування продовжуватиметься, доки не зустрінеться символ "**\n**". Детальнішу інформацію про модифікатори функцій **printf()** та **scanf()** можна знайти у відповідній документації. Також часто використовуються аналоги цих функцій, що працюють не з стандартним вводом-виводом, а наприклад з файлами **fprintf()** та **fscanf()**, чи стрічками **sprintf()** та **sscanf()**.

3.2.6.Функції і перемикання вводу-виводу.

У мові С, наслідуючи концепцію операційної системи UNIX, що розглядає будь-яку перефирію як деякий файл (концепція – "*все довкола файл*"), стандартним вводу та виводу відповідають файли з зарезервованими назвами **stdin** та **stdout**, що за замовчуванням відповідають клавіатурі та екрану консолі. Відповідно, функції **scanf()** та **printf()** працюють з цими файлами. Крім описаних функцій в стандартній бібліотеці вводу-виводу **<stdio.h>** існують і інші. Зокрема функції вводу і виводу одного символа: **getchar()** та **putchar()**. Ці функції використовуються для вводу-виводу текстів. Функція **getchar()** одержує один символ з клавіатури і записує його в стандартний файл вводу **stdin**. Функція **putchar(char a)** пересилає один символ з пам'яті машини в стандартний файл виводу **stdout** (тобто на екран). Найпростіша програма, яка відображає роботу цих функцій має вигляд:

```
#include <stdio.h>
int main()
{   char ch;
    ch = getchar();
```

```
        putchar(ch);
        return 0;
    }
```

Параметром функції **putchar()** є ім'я змінної, що виводиться на друк. Функція **getchar()** параметрів не має.

Щоб відмітити де закінчується один файл і починається інший вводиться таке поняття, як "*ознака кінця файлу*" – **EOF** (*End-of-File*)[1]. З використанням символьної константи **EOF** програма копіювання із стандартного файлу вводу **stdin** в стандартний файл виводу **stdout** має вигляд:

```
#include <stdio.h>
int main()
{   char ch;
    while((ch=getchar())!=EOF) putchar(ch);
    return 0;
}
```

Якщо існує потреба вводити інформацію не з стандартного файлу (тобто з клавіатури), а з файлу що міститься на якомусь іншому периферійному пристрою, необхідно вказати комп'ютеру що джерело даних є файл, а не клавіатура. Це можна зробити двома методами:

1) Явно, використовуючи стандартні функції, що відкривають і закривають файли, організують зчитування і запис даних.
2) Не змінюючи програми (тобто функцій вводу-виводу), а використовуючи при її запуску засоби консольних команд операційної системи перемкнути ввід-вивід, тобто вказати комп'ютеру при виконанні програми, що вхідні дані містяться не у стандартному файлі **stdin**, а наприклад у файлі **data.txt**.

Історично операція перемикання вводу-виводу – це засіб операційної системи UNIX, а не самої мови С. Але вона виявилась настільки корисною, що при переносі компілятора з мови С на інші системи найчастіше переноситься і ця операція. Перемикання вводу здійснюється за допомогою знака "**<**". Наприклад, якщо програма **test.exe** для вводу використовує функцію **getchar()**, то командна стрічка

```
test.exe<data.txt
```

вказує програмі, що вхідні дані вводяться не з клавіатури, а з файлу **data.txt**.

Перемикання виводу здійснюється за допомогою знаку "**>**".

```
test.exe>rez.txt
```

Вивід результатів **test.exe** буде здійснюватись у файл **rez.txt**.

---

[1] Для лінійки операційних систем Windows, **EOF** на консолі відповідає комбінація клавіш **Ctrl+Z**.

### 3.2. Контрольні запитання

1. Що таке директиви препроцесора, для чого вони існують?
2. Які функції має директива **`#define`**?
3. Які директиви умовної компіляції?
4. Які Ви знаєте оператори циклу у мові С?
5. Що Ви знаєте про оператори **`break`** і **`continue`**?
6. Які функції вводу-виводу Ви знаєте?
7. Що таке перемикання вводу-виводу?

### 3.3. Практичне завдання

1. Ознайомитися з директивами препроцесора мови С, з операторами циклу і функціями вводу-виводу.
2. Написати просту програму с використанням операторів циклу, директиви препроцесора та форматованого ввіду-виводу у мові С.

### 3.4. Зміст звіту

1. Титульний аркуш.
2. Мета роботи.
3. Блок-схема алгоритмів у відповідності до завдання.
4. Тексти програм у відповідності до завдання.
5. Результати обчислень.
6. Аналіз результатів, висновки.