

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет “Львівська політехніка”



ОСНОВНІ ПОНЯТТЯ МОВИ С
РЕАЛІЗАЦІЯ АЛГОРИТМІВ З РОЗГАЛУЖЕННЯМ

ІНСТРУКЦІЯ
до лабораторної роботи № 2 з курсу
“Основи програмування”
для базового напрямку “Програмна інженерія”

Затверджено
На засіданні кафедри
програмного забезпечення
Протокол № від

ЛЬВІВ – 2018

1. МЕТА РОБОТИ

Мета роботи – навчитися програмувати на мові C найпростіші лінійні алгоритми та алгоритми з галуженням

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1. Вступ

Мова C була створена на основі двох ранніх мов програмування – BCPL і B. Мова BCPL була створена у 1967 році Мартіном Річардом як мова для написання компіляторів і програмного забезпечення операційних систем. Автором мови B, яка була дублікатом мови BCPL, вважається Кен Томпсон. Він використовував її для створення ранніх версій операційної системи (ОС) UNIX в компанії Bell Laboratories в 1970 р. І BCPL, і B були “нетиповими” мовами – кожний елемент даних займав одне машинне слово в пам’яті і увесь тягар обробки елемента даних, наприклад, як цілого числа чи дійсного числа лягав на плечі програміста.

Мова C була розвинута з мови B Деннісом Рітчі в Bell Laboratories і вперше була реалізована на комп’ютері DEC в 1972 р. Першопочатково мова C набула широкої популярності як мова розробки ОС UNIX. Сьогодні більшість операційних систем написані на C або C++. Протягом двох наступних десятиліть мова C стала доступною для переважної більшості комп’ютерів. Зараз прийнято вважати, що мова C є апаратно незалежною, що означає, що при належній і ретельній розробці на C можна створювати *мобільні* програми, які можуть бути перенесені на комп’ютер будь-якої архітектури. До кінця 70-х років минулого століття, внаслідок постійного вдосконалення, мова C “доросла” до рівня, який тепер прийнято називати “класичним C” або “C Кернігана і Рітчі”. Публікація книги Кернігана і Рітчі “The C programming language” стала справжньою віхою в історії цієї мови (і програмування в цілому). Однак широке поширення мови C для різних типів комп’ютерів привело, на жаль, до великої кількості варіацій мови, які були схожими, але несумісними. Це стало серйозною проблемою для спільноти розробників програмного забезпечення, оскільки вони мали велику потребу в написанні таких програм, які можна було б виконувати на різних апаратних платформах (тобто сумісних програм). Стало очевидно, що необхідна стандартна версія мови C. У 1983 р. Американський Національний Інститут Стандартів ANSI приступив до розробки такого стандарту, який був затверджений у 1989 р. Пізніше в кооперації з Міжнародною організацією стандартів ISO був створений і опублікований у 1990 р. міжнародний стандарт ANSI/ISO 9899:1990. Згодом був введений стандарт ANSI/ISO 9899:1999 у 1999 році. Стандарт C99 ввів такі особливості, як вкладені функції, вільні місця оголошення змінних, нові типи даних, масиви зі змінними довжинами тощо. Останнім опублікованим стандартом є C11 - ISO/IEC 9899:2011 під кодовим ім’ям C11X. У новій специфікації збільшена сумісність з мовою C++ і представлені деякі нові можливості. Проте варто зазначити, що не всі компілятори повністю підтримують останні два стандарти. Наприклад, з часу розвитку мови C++ компанії Microsoft і Borland зосередилися переважно на останній, оскільки вона забезпечує подібну функціональність поряд з можливостями об’єктно-орієнтованої мови.

2.2. Алфавіт та елементарні конструкції мови C

Кожна мова програмування має свій *алфавіт*, тобто набір допустимих символів з яких може складатися програма. Алфавіт мови C містить:

- 1) великі і малі букви латинського алфавіту (A...Z, a...z);
- 2) арабські цифри від 0 до 9;
- 3) прості спеціальні символи: ‘+’, ‘-’, ‘*’, ‘/’, ‘:’, ‘;’, ‘,’, ‘=’, ‘>’, ‘<’, ‘[’, ‘]’, ‘(’, ‘)’, ‘{’, ‘}’, ‘.’, ‘_’;
- 4) складені спеціальні символи: ‘!=’, ‘>=’, ‘<=’, ‘->’.

Із цих символів складаються базові елементи мови. До них належать:

- 1) ідентифікатори;
- 2) числа;
- 3) рядки.

Усі імена, які використовуються в програмі для означення різних об'єктів і конструкцій називаються *ідентифікаторами*. Усі ідентифікатори мови C можна поділити на три групи:

I. **Службові (зарезервовані) слова** мови: призначення цих слів визначено самою мовою, їх не можна використовувати інакше, як це передбачено правилами (синтаксисом) мови. Основні службові слова подано нижче:

asm	enum	signed
auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while

II. **Бібліотечні ідентифікатори**: використовуються для позначення констант, функцій та визначень з різних бібліотек, які є складовою частиною системи програмування C. Наприклад, COS – ідентифікатор для позначення стандартної функції cos(x).

III. **Ідентифікатори користувача**: використовуються для позначення об'єктів (типів, змінних, констант, функцій тощо) *визначених самим програмістом*.

Слід пам'ятати такі основні правила запису ідентифікаторів користувача:

- ✓ Ідентифікатор складається з букв, цифр і знаку підкреслення (до складу ідентифікатора не може входити будь-який інший спеціальний символ).
- ✓ Першим символом в ідентифікаторі може бути буква або символ підкреслення. Останній НЕ рекомендується використовувати на початку ідентифікатора, оскільки багато бібліотечних змінних та функцій починаються саме з цього знаку.
- ✓ У мові C розрізняється регістр букв, тобто, X і x – це два різні ідентифікатори.
- ✓ Ідентифікатори користувача не повинні співпадати зі службовими словами та бібліотечними ідентифікаторами.

Числа в мові C можуть записуватися в десятковій системі числення, а також у вісімковій (починаються з префікса 0) та шістнадцятковій (починаються з префікса 0x) системах. Числа у десятковій формі можуть бути цілими і дійсними. *Цілі числа* записуються у звичайній формі послідовності цифр. Особливістю мови C є те, що при записі цілих чисел можна використовувати суфікси, які означають довжину цілого (кількість двійкових розрядів). Наприклад, суфікс L означає довге ціле зі знаком, а суфікс UL – беззнакове довге ціле. Числа дійсного типу мають дві форми запису:

1) з фіксованою крапкою – у вигляді цілої та дробової частини, розділеної крапкою: 25.34, 3.1415;

2) з плаваючою крапкою – у вигляді мантиси та порядку, розділених буквою E: 0.31415E1.

Рядки – це послідовність символів, записаних між парою подвійних лапок: "Hello World!".

2.3. Поняття змінної та константи.

Будь-яка програма задає конкретні команди, які виконує комп'ютер. Ці команди мають вигляд операторів мови програмування, і вони, як правило, передбачають наявність певних величин або об'єктів над якими виконуються дії задані операторами. Ці величини прийнято називати *даними*. Іншими словами, в найбільш загальному випадку можна стверджувати, що **програма=оператори+дані**. Для представлення даних в програмі найчастіше використовуються такі програмні об'єкти як змінні та константи. *Змінні* – це величини, які можуть змінюватися в процесі виконання програми, а *константи* – це величини, які не змінюють свого значення протягом усього часу виконання програми. Константи записуються або значеннями: 3.1415, 10, або іменами: Pi, EPS – іменовані константи.

Всі змінні в С-програмі повинні бути обов'язково оголошені (задекларовані) до того як вони будуть використані. Оголошення змінної має задавати *тип даних змінної* та ідентифікатор змінної, а також може містити початкове значення змінної (*ініціалізація змінної*). Типи даних у мові С поділяються на *прості (скалярні) та складені (структуровані)*. До скалярних типів відносяться:

- 1) **char** – одиничний байт, що містить один символ;
- 2) **int** – ціле число;
- 3) **float** – число з плаваючою крапкою одиничної точності;
- 4) **double** – число з плаваючою крапкою подвійної точності.

Для розширення базових типів використовуються *кваліфікатори*:

- 1) **short** – короткий; 2) **long** – довгий.

Ці кваліфікатори застосовуються до цілого типу: **short int** (можна писати просто **short**) – короткий цілий, **long int** (скорочено **long**) – довгий цілий. Наприклад, якщо ціле число типу **int** може займати в пам'яті машини або 16 біт або 32 біти, то **long** займає 32 біти, а **short** – 16 біт.

Кваліфікатор **long** може розширювати тип **double**. Тип **long double** – числа з плаваючою крапкою підвищеної точності.

Кваліфікатор 1) **signed** – із знаком; 2) **unsigned** – без знака. Застосовуються до типів **int** і **char**. Якщо значенню типу **char** відводиться 8 біт (1 байт), то **unsigned** приймає значення від 0 до 255, а **signed** від -128 до 127.

У мові С не існує логічного або булевського типу, хоча логічні операції використовуються. Треба запам'ятати, що значенню “істина” відповідає “не нуль”, тобто будь-яке число, що не дорівнює нулю, а “не істина” – “нуль”.

Вибір імені (ідентифікатора) змінної бажано узгоджувати з її змістом. Приклади оголошення та ініціалізації змінних:

```
char sym, ls='A';
int n=10, m, k;
float eps=0.001;
long int number;
unsigned char s;
```

До будь-якої змінної в оголошенні можна використати кваліфікатор **const**, тоді змінна перетворюється в константу, тобто її значення при виконанні програми не змінюється:

```
const int n=20;
const double pi=3.1415;
```

2.4. Структура програми

Будь-яка мова програмування високого рівня (С не є виключенням) має свої вимоги та правила щодо організації програми. Традиційно для пояснення структури програми мовою С розглядається найпростіша програма, яка виводить певне текстове повідомлення (наприклад, “Моя перша програма мовою С!”) на екран монітора. Вихідний текст (адаптований до середовища програмування MS Visual C++ 2010 Express Edition) такої програми має вигляд:

```
// My first C application
#include <stdio.h>

main()
{
    printf("My first C application! \n");
    return 0;
}
```

Ця проста програма ілюструє цілий ряд важливих особливостей мови С. Розглянемо детально кожний рядок цієї програми.

Перший рядок

```
// My first C application
```

починається з символу `//`, який означає, що наступний після нього текст є коментарем. Коментарі використовуються для того, щоб документувати програму і полегшити її розуміння.

Коментарі допомагають іншим людям читати і розуміти Вашу програму. Коментарі ігноруються компілятором C, і тому вони не викликають ніяких дій з боку комп'ютера. Як правило, коментар у першому рядку програми просто описує призначення програми. Коментар, який починається з символа `//` називається *однострічковим* коментарем, тому, що він закінчується в кінці поточного рядка. *Багатострічковий* коментар в C повинен розміщатися між парою символів `/*` та `*/`.

Рекомендація. Кожна програма повинна починатися з коментаря, який описує ціль програми.

Рядок

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

є директивою *препроцесора* мови C, яка дає вказівку препроцесору включити в текст програми вміст файлу **stdio.h**. Сам файл з розширенням `.h` прийнято називати *заголовочним файлом* або *header-файлом*. Рядки програми на C, які починаються з символа `#` є директивами і обробляються препроцесором перед початком компіляції програми. В даному випадку заголовний файл **stdio.h** містить оголошення функцій вводу/виводу.

Рядок

```
main()
```

повинен бути приступним у кожній програмі на C. Пара круглих дужок після слова **main** означають, що `main` – це програмний блок, який називається *функцією*. Будь-яка C-програма складається з однієї чи декількох функцій, одна з яких повинна обов'язково називатися `main`. Функція `main` називається *головною функцією* і вона є точкою входу в програму. Це означає, що кожна програма на C починає виконуватися з функції `main`, навіть якщо ця функція не є першою в тексті програми.

Ліва фігурна дужка `{` починає *тіло* кожної функції. Відповідно, права фігурна дужка `}` повинна завершувати кожну функцію. Проводячи аналогію з мовою Паскаль, пара фігурних дужок `{}` відповідає ключовим словам `begin end` мови Паскаль.

Рядок

```
printf("My first C application! \n");
```

задає команду комп'ютеру вивести на екран рядок символів, поміщених в подвійні лапки. Ця команда має вигляд виклику функції виведення на консоль. Такий виклик складається з ідентифікатора (назви) функції `printf()`, рядка-параметра “Моя перша програма мовою C! \n” та символа крапки з комою `;`, який є *ознакою завершення оператора чи команди*. Кожний оператор або команда в C повинні закінчуватися цим символом.

Однак, якщо виконати цю програму, то можна побачити, що символи `\n` з даного літералу не виводяться на екран. Символ `\` (*обернений слеш*) називається *символом переходу* або *escape-символом*. Він позначає собою початок так званої *керуючої послідовності* або *escape-послідовності*, яка задає виведення певного спеціального символу. У нашому випадку керуюча послідовність `\n` означає *початок нового рядка* і викликає переміщення курсора (тобто індикатора поточної позиції на екрані) на початок наступного рядка на екрані. Деякі інші, найчастіше вживані escape-послідовності наведені в табл.1.

Таблиця 1. Деякі керуючі послідовності

Керуюча послідовність	Опис призначення
<code>\n</code>	Новий рядок, курсор встановлюється на початок нового рядка
<code>\t</code>	Горизонтальна табуляція, курсор встановлюється на наступну позицію табуляції
<code>\r</code>	Повернення каретки, курсор встановлюється на початок поточного рядка
<code>\\</code>	Друк символу <code>\</code>
<code>\"</code>	Друк символу <code>"</code>

Рядком

```
return 0;
```

має завершуватися функція `main()`, якщо вона має тип результату відмінний від **void**. Відсутність типу результату функції `main()` в заголовку означає, що вона повертає результат типу **int** за замовчуванням. Сам оператор **return** вказує на вихід з функції (тобто на завершення виконання функції), причому значення 0 означає, що функція `main()`, а отже і сама програма, завершилася успішно.

2.5. Операції в мові C

На даному етапі вивчення мови C відомі нам елементарні операції можна поділити на три групи:

- 1) арифметичні операції;
- 2) операції порівняння;
- 3) операція присвоєння.

Арифметичні операції: додавання +, віднімання -, множення *, ділення /, залишок від ділення % виконуються над операндами цілого або дійсного типу. Наприклад, $2+7$ результат 9 цілого типу, $2.5+7=9.5$ – результат дійсного типу. До операції ділення в мові C потрібно відноситись дуже уважно. Якщо обидва операнди цілого типу, то і результат буде цілого типу, тому в результаті виконання оператора $S=2/5$; змінній S буде присвоєно значення 0, щоб одержати правильний результат необхідно щоб хоча б один операнд був дійсного типу, тобто $S=2.0 / 5.0$; Операція % виконується тільки над операндами цілого типу.

5	2	
4	2	← $5 / 2 = 2$ (ціле частини)
1		← $5 \% 2 = 1$ (остача)

Перераховані вище операції є *бінарними*, оскільки вони мають два *операнди*. Крім них у мові C є специфічні *унарні* операції (тобто такі які мають лише один операнд) *інкременту* ++ та *декрименту* --. Інкрементна операція ++ збільшує значення свого операнда на 1. Оператор $n++$; аналогічний такому оператору: $n=n+1$; Декриментна операція -- віднімає 1 від поточного значення свого операнда. Розрізняють дві форми запису цих операцій:

1. *префіксну* ++n - змінна n збільшується на 1 до того, як використовується у виразі;
2. *постфіксну* n++ - змінна n збільшується на 1 після того, як її значення буде використано у виразі.

Для демонстрації виконання цих операцій виконайте таку програму:

```
#include <stdio.h>

main()
{
    int a=1, b=1;
    int aplus, plusb;

    aplus=a++;
    plusb=++b;
    printf("a=%d aplus=%d\n", a, aplus);
    printf("b=%d plusb=%d\n", b, plusb);
    return 0;
}
```

В результаті виконання цієї програми одержимо:

```
a=2 aplus=1
b=2 plusb=2
```

Значення *a* збільшилось на 1 після того як виконалась операція присвоєння. Значення *b* спочатку збільшилось на 1, а тоді виконалась операція присвоєння.

Операції порівняння : менше або рівне \leq , більше $>$, менше $<$, більше або рівне \geq , рівне $==$ та нерівне $!=$ можуть виконуватись над операндами будь-якого однакового типу. У мові C результатом виконання операцій порівняння є значення 1 (відповідає логічному значенню TRUE в мові Паскаль), якщо операція істинна, і значення 0 (відповідає логічному значенню FALSE в мові Паскаль), якщо операція хибна.

Операція присвоєння: на відміну від більшості інших мов програмування високого рівня присвоєння в C розглядається як бінарна операція $=$. У найпростішому випадку операція присвоєння має вигляд **<змінна>=<вираз>**; і полягає у тому, що змінній (лівий операнд) присвоюється значення, яке є результатом обчислення виразу (правий операнд).

Розглянуті вище операції можна звести в одну таблицю 2.

Таблиця 2. Елементарні операції мови C

Назва операції	Операція в C	Алгебраїчний вираз	Вираз мовою C
Додавання	+	$a+5$	$a+5$
Віднімання	-	$x-y$	$x-y$
Множення	*	abc	$a*b*c$
Ділення	/	$\frac{z}{2}$	$z/2$
Остача від ділення	%	$a \bmod 2$	$a\%2$
Інкремент	++	$a+1$	$a++$
Декремент	--	$k-1$	$k--$
Рівність	==	$x=y$	$x==y$
Нерівність	!=	$x \neq y$	$x!=y$
Більше	>	$x>y$	$x>y$
Менше	<	$x<y$	$x<y$
Більше або рівне	>=	$x \geq y$	$x \geq y$
Менше або рівне	<=	$x \leq y$	$x \leq y$
Присвоєння	=	$a=b+c$	$a=b+c$

При використанні операцій важливе значення відіграють пріоритет (старшинство) та асоціативність (порядок виконання) операцій. Для тих операцій з якими ми вже познайомилися групування за старшинством (в порядку спадання пріоритету) наведено у таблиці 3.

Таблиця 3. Пріоритет та асоціативність операцій

Операція	Асоціативність
()	Зліва направо
++ --	Справа наліво
* / %	Зліва направо
+ -	Зліва направо
< <= > >=	Зліва направо
== !=	Зліва направо
<< >>	Зліва направо
=	Справа наліво

Зауважимо, що круглі дужки () у виразах мови C розглядаються як операція *взяти в дужки*.

2.6. Стандартні функції

Програми на C містять частини, які називаються *функціями*. Ви можете програмувати кожну частину, якщо вам необхідно сформулювати C-програму. Але велика частина програмістів на C користується перевагами наборів функцій з бібліотеки стандартних функцій ANSI C.

Використання функцій стандартної бібліотеки замість написання власних версій тих же функцій може підвищити ефективність програм, оскільки ці функції написані спеціально з урахуванням ефективності їх виконання. Окрім того, використання функцій стандартної бібліотеки замість написання власних версій тих же функцій може підвищити мобільність програм, оскільки ці функції включені практично в усі реалізації С.

При виклику стандартної функції слід вказати її ім'я і в дужках її аргумент.

Перелік основних математичних функцій з стандартної бібліотеки мови С (math.h), з описом типу аргументів і результату приведено в таблиці 4.

Таблиця 4. Основні математичні функції мови С

Ім'я функції	Математичний запис	Тип і межі зміни аргументів	Тип результату
sin(x)	$\sin x$	double	double
cos(x)	$\cos x$	double	double
tan(x)	$\operatorname{tg} x$	double	double
asin(x)	$\arcsin x$	double $x \in [-1, 1]$	$[-\pi/2, \pi/2]$
acos(x)	$\arccos x$	$x \in [-1, 1]$	$[0, \pi]$
atan(x)	$\operatorname{arctg} x$	$x \in [-\pi/2, \pi/2]$	double
sinh(x)	$\operatorname{sh} x$	double	double
cosh(x)	$\operatorname{ch} x$	double	double
tanh(x)	$\operatorname{th} x$	double	double
exp(x)	e^x	double	double
log(x)	$\ln x$	$x > 0$	double
log10(x)	$\lg x$	$x > 0$	double
pow(x,y)	x^y	$x \neq 0; y > 0$	double
sqrt(x)	\sqrt{x}	$x \geq 0$	double
fabs(x)	$ x $	double	double
ldexp(x,n)	$x \cdot 2^n$	x-double, n-int	double
fmod(x,y)	Залишок від ділення дійсних чисел x на y	double	double

2.7. Приклад програми обчислення математичних виразів мовою С

Написати програму для обчислення виразу: $Y = \sin(\sqrt{2x} - \lg(4x))^3$. Значення x ввести з клавіатури, результат у вивести на екран.

Програма матиме такий вигляд:

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

void main()
{
    double x, y;

    printf("Enter x: ");
    scanf("%lf", &x);
    y=pow(sin(sqrt(2*x)-log10(4*x)), 3);
    printf("x=%lf y=%lf\n", x, y);
    _getch();
}
```

2.8. Алгоритми

Вивчаючи основи програмування потрібно освоїти не лише одну з мов програмування високого рівня, а й навчитися складати та аналізувати алгоритми розв'язку різних задач, і таким чином, виробити алгоритмічне мислення.

Поняття алгоритму є одним з фундаментальних понять в комп'ютерних науках. Воно виникло задовго до появи перших комп'ютерів і стало одним з основних у математиці. Вважається, що саме слово “алгоритм” походить від імені видатного математика середньовіччя Мухаммеда бен-Муси аль-Хорезмі. Він розробив алгоритм (правила) виконання арифметичних дій над десятковими числами, яким ми користуємось до сих пір. Тепер слово “алгоритм” використовується не тільки в математиці, так говорять про алгоритм гри у шахи, алгоритм керування різними процесами тощо.

Для розуміння поняття “алгоритм” та його ролі в програмуванні не обов'язково застосовувати строго математичний формалізм, достатньо дати тлумачення цього поняття на “інтуїтивному” рівні. Тут, в першу чергу, слід відзначити важливе значення поняття “виконавець алгоритму”. Алгоритм завжди повинен бути сформульованим у розрахунку на конкретного виконавця, тобто він має бути керівництвом до дій для виконавця. Саме тому, значення слова “алгоритм” є схожим за змістом до значення слів “вказівка”, “інструкція”. Можна стверджувати, що *алгоритм – це зрозумілі й точні вказівки виконавцю (комп'ютеру) здійснити певну послідовність дій (обчислювальних операцій) для розв'язання поставленої задачі.*

Будь-який алгоритм стає алгоритмом лише тоді, коли він набере якої-небудь форми. В програмуванні склалися повністю визначені традиції в поданні алгоритмів, а саме розрізняють такі форми запису алгоритмів:

- словесний запис алгоритму (розмовна мова);
- псевдокод (структурно-стилізована мова);
- блок-схема (мова графічних символів);
- Комп'ютерна програма (мова програмування високого рівня).



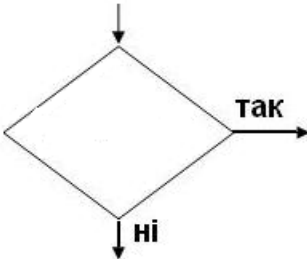

Словесний запис алгоритму орієнтований, взагалі кажучи, на людину як виконавця. В програмуванні він використовується лише на першому етапі розробки програми, коли потрібно подати алгоритм в цілому, усвідомити послідовність виконання дій та логічні зв'язки між ними, проглянути можливі варіанти. Словесна форма допускає найбільшу свободу дій в способах опису як послідовності дій, так і рівня їх деталізації, і таким чином вона є найменш формалізованою і строгою.

Наступним кроком у напрямку підвищення рівня формалізації опису алгоритму є використання так званого *псевдокоду*. Псевдокод – це система позначень і правил, призначена для одноманітного запису алгоритмів. У псевдокоді відсутні строгі синтаксичні правила для запису команд, тому програма на псевдокоді не може виконуватися комп'ютером. Його призначення – допомогти розробнику “осмислити” програму перед тим як спробувати її написати мовою програмування такою як С. А завдяки наявності конструкцій, які притаманні мовам програмування, значно полегшується перехід від детально підготовленої програми на псевдокоді до програми написаної мовою С.

На одному рівні формалізації опису алгоритму поряд з псевдокодом знаходиться графічна форма опису у вигляді *блок-схеми*. Основною перевагою блок-схем над псевдокодом є більша наочність подання алгоритму. Але для великих алгоритмів ця перевага втрачається. В блок-схемах для позначення алгоритму використовуються спеціальні символи у вигляді найпростіших геометричних фігур, які називаються блоками. Блоки з'єднуються за допомогою стрілок, які називаються лініями зв'язку і які вказують на черговість виконання. Самі команди поміщаються усередину блоків і можуть набувати, взагалі кажучи, довільної форми, наприклад, задаватися словами (хоча в такому випадку формалізація опису зменшуються). Блок-схеми можуть використовуватися як для опису повного алгоритму, так і для опису деякого фрагменту алгоритма. У першому випадку блок-схема повинна починатися з блоку у вигляді овалу з написом *Початок* і завершуватися також овалом тільки вже з написом *Кінець*. У другому випадку будемо починати і завершувати блок-схему символом маленького кола без напису, який називається блоком з'єднання (злиття). Для позначення всіх можливих елементарних дій

на блок-схемах будемо використовувати блок у вигляді прямокутника, який називатимемо блоком обробки інформації або блоком виконання дій. Цей блок повинен мати один вхід, заданий стрілкою, яка входить в блок зверху або збоку, і один вихід, заданий стрілкою, що виходить з блоку донизу (також можливо у бік). І нарешті, блок у вигляді ромба, який називається логічним блоком, буде використовуватися для перевірки умов і вибору напрямку виконання алгоритму залежно від умови. Сама умова записується усередині ромба. Аналогічно, вхід та вихід позначаються стрілками, але на відміну від блоку виконання дій, логічний блок завжди має один вхід та два виходи. Описані блоки та інші блоки, які будуть використовуватися нами для побудови блок-схем наведено у табл.1.

Таблиця 1. Мова графічних символів

Назва символу	Позначення	Пояснення
Блок виконання дій		Обчислювальна дія або послідовність обчислювальних дій
Блок з'єднання		Об'єднання ліній потоку
Логічний блок		Перевірка умови
Блок початку/кінця		Початок і закінчення алгоритму

З певних комбінацій блоку виконання дій та логічного блоку створюються керуючі структури або базові конструкції алгоритмів. В теорії алгоритмів доведено, що будь-який алгоритм може бути побудований з використанням всього лише трьох базових конструкцій, а саме з **конструкцій слідування, розгалуження та повторення**. Це перетворює процес побудови алгоритму в його “збірку” з набору базових конструкцій. Блок-схеми керуючих структур можна розглядати як блок-схеми фрагментів алгоритму, тобто вони починатимуться з блоку злиття, який буде відігравати роль точки входу в структуру і закінчуватися також цим блоком, який буде точкою виходу з структури. Таке трактування керуючих структур з одним входом і одним виходом значно полегшує процес побудови алгоритму шляхом збирання з базових конструкцій – керуючі структури зв'язуються послідовно шляхом з'єднання точки виходу однієї з них з точкою входу іншої. Іншим способом збирання цілого алгоритму з базових конструкцій є вкладення однієї керуючої структури в іншу. Його ми розглянемо детально пізніше.

Під час запису алгоритму у словесній формі, у вигляді блок-схеми чи на псевдокоді допускається певна довільність в записі команд. На практиці як виконавець алгоритмів використовують комп'ютер, тому алгоритм має бути записаний мовою “зрозумілою” комп'ютером. Тут на перший план виступає необхідність точного запису команд, яка не залишає місця для довільного тлумачення їх виконавцем. Тому мова для запису алгоритму має бути строго формалізована – вона називається мовою програмування, а форма алгоритму цією мовою – комп'ютерною програмою або просто програмою.

Отже, будь-який алгоритм може бути побудований з використанням всього лише трьох базових конструкцій, а саме конструкцій слідування, розгалуження та повторення. Структура слідування вбудована в мові С. Це означає, що оператори програми виконуються один за одним в тій послідовності, в якій вони записані у тексті програми, поки не буде вказано інший порядок виконання.

Організація розгалуження в С здійснюється за допомогою трьох типів програмних структур вибору: *оператора умови if з одиничним вибором, оператора умови if/else з подвійним вибором і оператора вибору switch з множинним вибором.*

Конструкція повторення реалізується в С за допомогою операторів циклу, які будуть розглянуті в наступній лабораторній.

2.9. Оператори умови в мові С

Оператор if виконує певну дію, якщо умова вибору є істинною, і пропускає виконання цієї дії, якщо умова є хибною. Синтаксичний опис оператора умови з одиничним вибором такий:

if (<умова>) <оператор>;

Спочатку обчислюється <умова>, яка синтаксично задається у вигляді виразу, результатом виконання якого є або ненульове значення (означає, що умова є істинною або умова виконується), або значення 0 (означає, що умова є хибною або умова не виконується). У мові С відсутній спеціальний логічний тип, якому відповідають значення типу TRUE і FALSE (як у мові Паскаль). Аналогом значення TRUE у С є будь-яке ненульове значення, а значення FALSE – 0. Таким чином, виконання оператора if полягає в наступному: якщо <умова> є істинною, то виконується <оператор>, якщо <умова> є хибною, то оператор заданий після умови виконуватися не буде, а програма продовжить своє виконання з наступного після if оператора.

Трактування умови в операторі if як звичайного виразу, дозволяє створювати в С умови виконання, які не обов'язково повинні містити операції, що за своєю логікою виконання завжди дають лише два можливих значення, як наприклад, операції порівняння. Ці операції, як відомо, дають результат 1, якщо відношення, задане операцією порівняння дійсно має місце, і результат 0 у протилежному випадку. Для прикладу, поекспериментуйте з наведеною нижче програмою, задаючи різні вхідні значення (додатне, від'ємне, нульове) для змінної n. Зверніть увагу на те, в яких випадках виводиться на екран той чи інший текст і спробуйте пояснити чому. Також, додайте відповідні оператори виведення для перевірки результату виконання інших операцій порівняння.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int n;
    printf("Enter n:");
    scanf("%d", &n);
    if (n) printf("TRUE\n"); // така умова в С є коректною!
    if (n>0) printf("n is positive");
    if (n<0) printf("n is negative");
    if (n==0) printf("n is zero");
    _getch();
    return 0;
}
```

Для вибору однієї із двох можливих дій, в залежності від значення певної умови, використовується оператор умови if/else, який має таку синтаксичну структуру:

if (<умова>) <оператор1>; else <оператор2>

Аналогічно до попереднього випадку, спочатку обчислюється <умова>, і якщо вона є істинною (має ненульове значення), то виконується <оператор1>, а якщо умова є хибною (має нульове значення), то виконується <оператор2>. Блок-схеми на рис.1 наочно демонструють логіку роботи обох операторів умови.

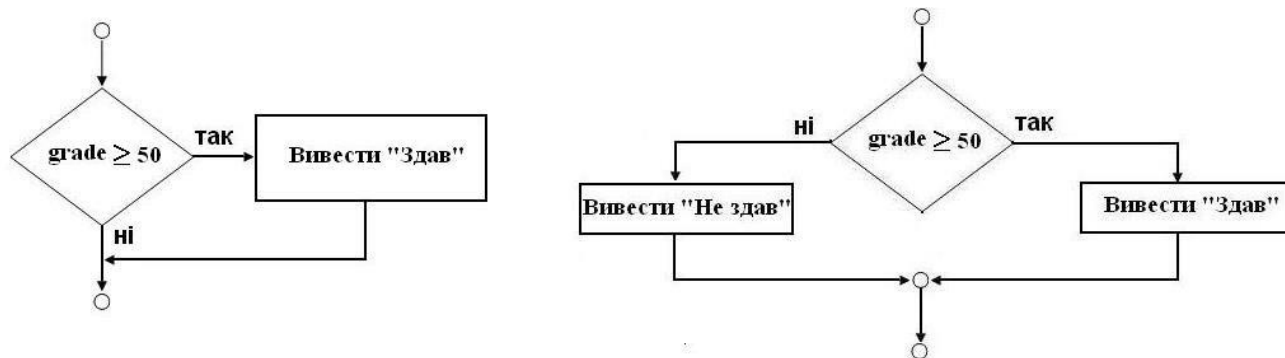


Рис.1. Керуюча логіка операторів **if** мови C

У ролі <оператор1> та <оператор2> може бути будь-який допустимий оператор мови C, в тому числі й сам оператор умови. У такому випадку отримуємо вкладені структури if/else, які дозволяють реалізувати розгалуження обчислювального процесу більше ніж у двох напрямках. Тут слід пам'ятати, що кожне службове слово else відноситься до першого перед ним слова if. Так, наприклад, при заданих початкових значеннях $x=1$; $y=-1$, після виконання оператора

if ($x>0$) if ($y>0$) $z=1$; else $z=2$;

змінна z буде мати значення 2. Якщо ми хочемо, щоб гілка else $z=2$; відносилася не до умови if ($y>0$), а до if ($x>0$), то треба цей оператор записати так:

if ($x>0$) { if ($y>0$) $z=1$; } else $z=2$;

За правилами мови C <оператор1> та <оператор2> мають бути структурно одним єдиним оператором. Якщо користувачеві необхідно виконати в цих місцях кілька операторів, то їх треба взяти в операторні дужки { }, тобто зробити ці декілька операторів складеним оператором. Складений оператор вживається у програмі завжди, якщо за синтаксисом мови C наступний оператор може бути тільки єдиним, а користувачеві потрібно виконати певну послідовність операторів. Складений оператор має таку структуру:

{ <оператор1>; <оператор2>; ... <операторN> }

У ролі прикладу використання складеного оператора в структурі **if/else**, розглянемо задачу обчислення виразу $y = 0.75\sqrt{x} - \frac{1}{2}\sqrt[3]{4}$ з врахуванням обмеження на область визначення функції квадратного кореня.

```

#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
    double x,y;
    printf("Введіть значення x:");
    scanf("%lf", &x);
    if (x<0)
        printf("При x<0 обчислення виразу неможливе!");
    else {
        y=0.75*sqrt(x)-0.5*pow(4, 1.0/3);
        printf("y=%lf", y);
    }
    _getch();
}
  
```

2.10. Логічні операції та умовна операція ?:

Іншим типом операцій в мові C, які завжди мають результат типу 'істина'/'не істина' є логічні операції. Цих операцій в C є три: операція логічного множення && (логічне І), операція логічного додавання || (логічне АБО) та операція логічного заперечення ! (логічне НЕ). Операції логічного множення і додавання працюють так, як показано у табл.2.

Таблиця 2. Логічне множення і логічне додавання

&&	0	Не нуль
0	0	0
Не нуль	0	1

	0	Не нуль
0	0	1
Не нуль	1	1

Логіка операції логічного заперечення така: $!0=1$ та $!1=0$.

За допомогою логічних операцій та операцій порівняння можна утворювати складні вирази для запису умови в операторі **if/else**. Наприклад, перевірка чи значення змінної x лежить в діапазоні від a до b може бути задана у вигляді такої умови:

$(x \geq a \ \&\& \ x \leq b)$

Мова C має також умовну операцію $?:$, яка є схожою на структуру **if/else**. Ця умовна операція є єдиною *тернарною операцією*, тобто такою, що має три операнда. Ці операнди разом з самою умовною операцією утворюють *умовний вираз*, який має такий вигляд:

<умова> ? <вираз1> : <вираз2>

Тут, перший операнд є умовою, другий операнд містить значення умовного виразу в тому випадку, якщо умова є істинною, а третій операнд рівний значенню умовного виразу, якщо умова є хибною. Наприклад, оператор виведення

```
printf (grade>=50 ? "ЗДАВ" : "НЕ ЗДАВ");
```

містить умовний вираз, значення якого рівне рядку "ЗДАВ", якщо умова **grade>=50** виконується, і рівне рядку "НЕ ЗДАВ", якщо ця умова не виконується. Таким чином, цей оператор з умовною операцією виконує фактично ті ж дії, що і аналогічний оператор **if/else**:

```
if (grade>=50) printf ("ЗДАВ"); else printf ("НЕ ЗДАВ");
```

Умовна операція має нижчий пріоритет ніж операція помістити в потік, тому сам умовний вираз має бути взятий в круглі дужки.

Значенням умовного виразу можуть також бути певні дії. Наприклад, умовний вираз

```
grade>=50 ? printf ("ЗДАВ") : printf ("НЕ ЗДАВ");
```

може бути прочитаний так: "Якщо оцінка $grade$ більша або рівна 50, то вивести повідомлення "ЗДАВ", інакше вивести повідомлення "НЕ ЗДАВ".

2.11. Оператор вибору switch

Часто в програмуванні виникає задача *вибору одного варіанта з багатьох*. Можна це зробити за допомогою вкладених структур **if/else**. Однак зручніший спосіб - використання *оператора вибору switch*, синтаксис загального вигляду якого такий:

```
switch (switch_expression)
{ case constant 1: statement 1; [break;]
.....
case constant i: statement i; [break;]
.....
case constant N: statement N; [break; ]
[default: statement N+1; ] }
```

Оператор **switch** виконується так. Спочатку обчислюється значення виразу **switch_expression**. Тип значення повинен бути одним із цілих - `char`, `int`, `unsigned int`, `long int` і `long unsigned`. Обчислене значення зрівнюється зі значеннями *констант вибору* або *константних виразів* **constant1**, ..., **constantN**. Заборонено використовувати в якості константи вибору змінну. При співпадінні значення `switch_expression` із `constanti` виконується оператор **statement1**. Потім керування передається на оператор відразу після `switch`, якщо в i -й гілці є присутнім оператор **break** (оператор `break` здійснює негайний вихід з оператора `switch`). У протилежному випадку виконуються оператори в гілках $i+1$, $i+2$ і так далі доти, поки в них не зустрінеться оператор `break` або не буде виконаний оператор `statement N+1`.

Якщо значення `switch_expression` не збіглося з жодною з констант `constanti`, ..., `constantN`, виконується оператор у гілці, позначеній **default**. При її відсутності виконується наступний після `switch` оператор.

Для прикладу напишемо програму, яка моделює роботу світлофора.

```

#include <stdio.h>
#include <conio.h>
void main()
{
    char ch;
    printf ("Enter letter:");
    scanf("%s", &ch);
    switch (ch) {
        case 'r': printf ("Stop! RED"); break;
        case 'y': printf ("Attention! YELLOW"); break;
        case 'g': printf ("Go! GREEN"); break;
        default: printf ("Wrong letter!");
    }

    _getch();
}

```

2.12. Оператор безумовного переходу

Для зміни послідовного виконання операторів використовується оператор безумовного переходу **goto**. Структура оператора goto:

```
goto <мітка>;
```

□

```
<мітка>: <оператор>;
```

Мітка задається за правилами запису ідентифікаторів. Якщо строго слідувати правилами структурного програмування, то нема ніякої необхідності в операторі безумовного переходу **goto**, який нещадно критикується в літературі по С. Однак існують окремі випадки, коли його використання може принести певну користь. Ці випадки будуть розглядатися пізніше.

2.13. Приклади програм з розгалуженням

Віавіа

Приклад 1. Задано три цілих числа a,b,c в діапазоні від 1000 до 9999. Знайти те з них сума цифр якого найбільша.

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int a,b,c,s,max,d; // оголошення змінних
    printf("Enter a,b,c:");
    scanf("%d %d %d",&a, &b, &c); // введення значення змінних
    // перевірка на відповідність введених значень вказаному діапазону
    if ((a<1000 || a>9999)|| (b<1000 || b>9999)|| (c<1000 || c>9999))
        // хоча б одне число не попадає в діапазон
        printf("All numbers must be between 1000 and 9999");
    else { // всі числа задані коректно, опрацьовуємо їх
        s=a/1000+ a/100%10 + a/10%10 + a%10; // сума цифр числа a
        max=s; // змінна max буде містити значення максимальної суми цифр
        d=a; // змінна d буде містити саме число сума цифр якого максимальна
        s=b/1000+ b/100%10 + b/10%10 + b%10; // сума цифр числа b
        // перевірка чи сума цифр числа b більша за поточне максимальне значення
        if (s>max) {
            max=s; // якщо так, то переприсвоюємо змінній max нове значення
            d=b; // і запам'ятовуємо число b в змінній d
        }

        s=c/1000+ c/100%10 + c/10%10 + c%10; // сума цифр числа c
        // перевірка чи сума цифр числа c більша за поточне максимальне значення
        if (s>max) {
            max=s; // якщо так, то переприсвоюємо змінній max нове значення
            d=c; // і запам'ятовуємо число c в змінній d
        }
        // тепер в змінній d маємо саме число, а в змінній max суму його цифр
    }
}

```

```

        // виводимо результат
        printf("Number with max sum of digits: %d",d);
        printf("Sum of digits is %d", max);
    }
    _getch();
}

```

Приклад 2. Задано чотири цілих числа a,b,c,d. Відомо, що одне з них відрізняється від трьох інших, які є однаковими. Знайти і вивести це число.

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int a,b,c,d; // оголошення змінних
    printf("Enter a,b,c,d:");
    scanf("%d %d %d %d",&a,&b,&c,&d); // введення значення змінних
    if (b==c && c==d && a!=b)
        printf("Number that differed is %d",a);
    else if (a==c && c==d && b!=a)
        printf("Number that differed is %d",b);
    else if (a==b && a==d && c!=a)
        printf("Number that differed is %d",c);
    else if (a==b && a==c && d!=a)
        printf("Number that differed is %d",d);
    else
        printf("Three of numbers must be equal!");
    _getch();
}

```

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке ідентифікатор? Які правила запису ідентифікаторів?
2. Що таке змінна? Для чого використовуються змінні? Яка відмінність змінної від константи?
3. Які є форми запису дійсних чисел у мові C?
4. Які прості типи даних мови C Ви знаєте? Які кваліфікатори можна до них застосовувати?
5. Чому в програмах доцільно використовувати коментарі? Які є типи коментарів в C?
6. Опишіть коротко структуру програми мовою C.
7. Як здійснюється виведення на екран в мові C?
8. Що таке керуюча послідовність? Наведіть приклади цих послідовностей.
9. Які операції в мові C Ви знаєте?
10. Які операції в мові C Ви знаєте?
11. Що таке пріоритет виконання операцій? Перерахуйте відомі Вам операції мови C в порядку зростання старшинства?
12. Що таке бібліотека стандартних функцій? Для чого вони використовуються?
13. Що таке алгоритм? Які є форми запису алгоритмів?
14. З яких базових конструкцій може складатися довільний алгоритм?
15. Які блоки можна використовувати на блок-схемах?
16. Як записується і працює умовний оператор if/else в C?
17. Які особливості вкладених структур if/else?
18. Які особливості виразу за допомогою якого задається умова оператора if ?
19. Напишіть вираз мовою C який перевіряє чи перша і остання цифри заданого трьохзначного цілого числа k рівні?
20. Що таке складений оператор? Коли він використовується?
21. Що таке умовна операція? Як вона працює?

22. Які логічні операції мови C Ви знаєте? Наведіть приклади їх використання.
23. Як реалізовано оператор вибору в C?
24. Для чого використовується оператор goto?

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичним матеріалом викладеним вище в даній інструкції і виконати приклади програм.
2. Одержати індивідуальне завдання з додатків 1 та 2.
3. Скласти програму на мові C у відповідності із завданням.
4. Виконати обчислення по програмі.
5. 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.

ДОДАТОК 1

Написати програму для обчислення заданих арифметичних виразів. Вважати, що X, Y – змінні, значення яких слід вводити з клавіатури, всі інші величини виразу описати як константи.

1. $R = \sqrt{x^2 + y^2 + z^2}$ де: x=5.2, y=6, z = -2
2. $U = (\cos x + \sin x)$, де: x = 3.2
3. $T_i = \frac{x}{1 - \frac{x^2}{3 - \frac{x^2}{5 - \frac{x^2}{7}}}}$, де: x = 0.49
4. $z = \sqrt{x} - \frac{x}{x+1} + 0.2x$ де: x = 6.4
5. $c = \frac{x}{a} - \frac{1}{ap} \lg(a + be^{px})$, де: x = 2, a = 3, p = 6, b = 2
6. $w = \frac{7.3910^{-3}x}{x^5(e^{1/k} - 1)}$, де: x = 2.6, k = 2
7. $s = \frac{x+y}{2x-y}(x+b)\sin x$, де: x = 2.6, y = 7.3, b = 2.1
8. $a = 2^{-x} \sqrt{x^{-4} + \sqrt{y}}$, де: x = 3.981, y = -1,625
9. $b = \sqrt{e^{x-1/\sin z}}$, де: x = 3.981, z = 0.512
10. $z = a^{(y^x)} + (3^x)^y$, де: x = 3.251, y = 0.325, a = 2
11. $a = \sqrt[4]{y + \sqrt[3]{x+1}}$, де: x = 17.421, y = 10.365

12. $b = x(\arctg z + e^{-(x+3)}),$ де: $x = -0.622, z = 5.541$
 13. $b = x(\sin \arctg x + \cos^2 y),$ де: $x = 0.335, y = 0.025, z = 32.005$
 14. $b = (1 + tg^2 \frac{z}{2})^{\sqrt{|y|+x}},$ де: $x = 0.100, y = -0.875, z = 0.765$
 15. $a = \lg(\sqrt{e^{x-y}} + x^{|y|} + z),$ де: $x = 1.542, y = -3.261, z = 80.005$
 16. $a = y^x + \sqrt{|x|+|y|},$ де: $x = -0.851, y = 1.250$
 17. $a = \left| x^{\frac{y}{x}} - \sqrt[3]{\frac{y}{x}} \right|,$ де: $x = 1.825, y = 18.225$
 18. $a = \sqrt{10(\sqrt[3]{x} + x^{(y+2)})},$ де: $x = 16.55, y = -2.75$
 19. $b = (\arcsin z)^2 + |x+y|,$ де: $x = 16.55, y = -2.75, z = 0.15$
 20. $a = \sqrt[3]{x + 4\sqrt{|y|}},$ де: $x = 37.15, y = -12.55$
 21. $b = \lg(\sqrt[3]{u} + \sqrt[4]{v} + 2),$ де: $u = 125.331, v = 33.075$
 22. $a = (2 + y^2) \frac{x + \frac{y}{2}}{y^2 + 1/(1 + y^2)},$ де: $x = 0.22, y = -6.72$
 23. $b = \sqrt{\sin^2 \arctgu + \cos v},$ де: $u = 10.05, v = 0.35$
 24. $Q = u^{(x+y)/2} - \sqrt[3]{\frac{x-1}{|y|+1}},$ де: $x = -12.650, y = -2.255, u = 3.205$
 25. $b = \frac{x + 3(x-y) + x^2}{(x-y)^2 + x^2},$ де: $x = -17.22, y = 6.33, z = 3.25$

ДОДАТОК 2

Не використовуючи оператор циклу скласти програму, яка

- За заданим кутом (в градусах) визначити знаки усіх тригонометричних функцій. (Не використовувати математичні функції з стандартної бібліотеки).
- Читає натуральні числа m, n і друкує всі натуральні числа менші від n , сума цифр яких дорівнює m .
- Обчислює корені квадратного рівняння $ax^2 + bx + c = 0$ для заданих довільних дійсних a, b, c . У випадку відсутності дійсних коренів цього рівняння друкує відповідне повідомлення.
- Визначити чи задана точка (x, y) належить півкругу, який описується нерівностями: $x^2 + y^2 \leq r^2$ та $y \geq 0$.
- Задані дійсні числа $a1, b1, c1, a2, b2, c2$. Надрукувати координати точки перетину прямих, які описуються рівняннями: $a1*x + b1*y = c1$ та $a2*x + b2*y = c2$ або повідомлення про те, що прямі не перетинаються.
- За заданим восьмизначним натуральним числом знаходить середнє арифметичне цифр цього числа та кількість ненульових цифр.
- За заданим натуральним дев'ятизначним числом знаходить суму цифр цього числа та кількість одиниць.
- Визначає номер максимального числа та середнє значення тільки додатних чисел з 10 чисел введених з клавіатури.

9. Визначає півсуму максимального та мінімального чисел, які знаходяться серед 10 введених з клавіатури додатних чисел.
10. Обчислює і виводить десяткове значення кожного з чисел, які є записом числа у системі числення з основою 5. З клавіатури послідовно ввести 4 п'ятизначних числа, контролюючи чи число є записаним в системі числення з основою 5.
11. Визначає номер мінімального числа та середнє значення всіх більших за -10 введених з клавіатури від'ємних чисел. (Кількість чисел 10).
12. За заданим натуральним восьмизначним числом знаходить подвоєний добуток ненульових цифр цього числа та кількість нулів.
13. За введеними трьома дійсними числами x, y, z обчислити значення

$$u = \begin{cases} \sqrt{|x \cdot y \cdot z|}, & \text{якщо } \max(x, y, z) < 19 \\ \frac{x}{|y| + z^2}, & \text{інакше} \end{cases}$$

14. Обчислює і виводить десяткове значення кожного з чисел, які є записом числа у системі числення з основою 7. З клавіатури послідовно ввести 3 шестизначних числа, контролюючи чи число є записаним в системі числення з основою 7.
15. За введеним натуральним дев'ятизначним числом видруковує нове число, утворене зворотнім записом цифр введеного числа.
16. За введеними трьома дійсними числами x, y, z обчислити значення

$$u = \frac{\max(x, y, z) - x \cdot \min(y, z)}{x \cdot y \cdot z + x^2 + y^3}$$

17. За введеними трьома дійсними числами x, y, z обчислити значення

$$u = \frac{\max(x, y, 5.65) + x \cdot \max(y, z)}{x^2 \cdot y^3 + z^4 - \min(x, y, z - 4)}$$

18. Визначає середнє арифметичне значення додатних чисел та кількість від'ємних чисел. Дійсні числа вводяться з клавіатури, кількість чисел 10.
19. За введеними значеннями коефіцієнтів лінійних алгебраїчних рівнянь визначити чи система цих рівнянь має розв'язок, якщо так – чи він єдиний. Система має вигляд:

$$\begin{cases} a_1x + b_1y + c_1z = d_1 \\ a_2x + b_2y + c_2z = d_2 \\ a_3x + b_3y + c_3z = d_3 \end{cases}$$

20. За введеними трьома дійсними числами x, y, z обчислити значення

$$u = \begin{cases} \max(x, y, z), & \text{якщо } x < 25 \\ \min(x, y, z), & \text{якщо } y > 14 \end{cases}$$

21. З клавіатури вводяться 10 додатних чисел. Обчислити кількість чисел більших за середнє арифметичне усіх введених та номери двох найбільших чисел.
22. Обчислює і виводить десяткове значення кожного з чисел, які є записом числа у системі числення з основою 8. З клавіатури послідовно ввести 4 шестизначних числа, контролюючи чи число є записаним в системі числення з основою 8.
23. З клавіатури ввести натуральне семизначне число. За сумою цифр цього числа надрукувати символ, ASCII-код якого дорівнює третій частині знайденої суми плюс 132.