[Питання 4](#_Toc73383684)

[Відповіді до питань до розділу "Асемблери" 9](#_Toc73383685)

[1. Схема трансляції, компонування і виконання програми. + 9](#_Toc73383686)

[2. Основні регістри мікропроцесора. + 10](#_Toc73383687)

[3. Позиційна незалежність програми в одно програмних ОС. + 11](#_Toc73383688)

[4. Режими адресування операндів в одно програмних ОС. + 11](#_Toc73383689)

[5. Моделі адресування операндів в командах мікропроцесора. + 12](#_Toc73383690)

[6. Моделі структури програм: головна програма, підпрограма, COM-програма (команди + дані). + 14](#_Toc73383691)

[7. Загальний формат бітової структури команди процесора Intel. Поля коду команди. + 17](#_Toc73383692)

[8. Будова байта ModRM способу адресування для 16- і 32-бітових режимів. + 20](#_Toc73383693)

[9. Схеми виконання команд мікропроцесора: команди без операндів (нуль-операндні), команди операцій і команди дії з одним операндом. + 22](#_Toc73383694)

[10. Схеми виконання команд мікропроцесора: команди з двома операндами, команди з трьома операндами. + 24](#_Toc73383695)

[11. Директиви асемблера та їх застосування. + 25](#_Toc73383696)

[12. Трансляція програм ASM за першим і другим переглядом. + 26](#_Toc73383697)

[13. Алгоритм першого перегляду асемблера при трансляції. + 28](#_Toc73383698)

[14. Алгоритм другого перегляду асемблера при трансляції. + 30](#_Toc73383699)

[15. Загальні принципи компонування програм. Об’єктні файли. + 32](#_Toc73383700)

[16. Компонувальники і принципи їх роботи.+ 33](#_Toc73383701)

[Відповіді до питань до розділу "Бібліотеки DLL" 36](#_Toc73383702)

[1. Переваги і недоліки статичного компонування програм. (+) 36](#_Toc73383703)

[2. Визначення динамічної бібліотеки. Схема динамічного завантаження в адресний простір процесу. (+) 36](#_Toc73383704)

[3. Переваги і недоліки використання динамічних бібліотек. (+) 37](#_Toc73383705)

[4. Зворотня сумісність динамічних бібліотек. (+) 38](#_Toc73383706)

[5. Загальні принципи зв’язування з DLL в алгоритмічних мовах для неявного і явного зв’язування. (+) 38](#_Toc73383707)

[6. Взаємодія динамічної бібліотеки з адресним простором процесу. Особливості об'єктного коду динамічних бібліотек. (+) 39](#_Toc73383708)

[7. Точка входу динамічної бібліотеки і приклади її раціонального використання. (+) 40](#_Toc73383709)

[8. Структура виконуваних файлів для Windows. Формат РЕ. (+) 41](#_Toc73383710)

[9. Схема процесу компонування для Windows у разі неявного зв'язування. (+) 42](#_Toc73383711)

[10. Схема процесу компонування для Windows у разі явного зв'язування. (+) 43](#_Toc73383712)

[11. Механізми передавання параметрів до процедур і функцій: за значенням; за посиланням (за адресою); за поверненим значенням; за результатом; за іменем; відкладеним обчисленням. Загальні визначення. (+) 44](#_Toc73383713)

[12. Способи передавання параметрів до процедур і функцій: в регістрах; в глобальних зміних; в стеку. Погодження (конвенції) для передавання параметрів у функцію через стек. (+) 46](#_Toc73383714)

[13. Модель конвенції stdcall передавання параметрів до процедур і функцій. (+) 47](#_Toc73383715)

[14. Стандартні типи даних Windows та їх еквіваленти в мові C. (+) 48](#_Toc73383716)

[15. Процедура підготовки функцій для DLL на прикладі алгоритмічної мови (C++ чи іншої). (+) 48](#_Toc73383717)

[16. Компіляція функцій в бібліотеку DLL (створення DLL). Налаштування компілятора і завантажувача. (+) 50](#_Toc73383718)

[17. Використання DLL в прикладних програмах методом явного зв’язування (на прикладі мови C чи іншої). (+) 51](#_Toc73383719)

[18. Використання DLL в прикладних програмах методом неявного зв’язування (на прикладі мови C чи іншої). (+) 53](#_Toc73383720)

[Відповіді до питань до розділу "Графічні редактори" 56](#_Toc73383721)

[1. Піксел. Роздільна здатність екрана. Палітра кольорів. Принцип малювання на екрані.+ 56](#_Toc73383722)

[2. Графічні примітиви. Графічні бібліотеки. Принципи використання.+ 57](#_Toc73383723)

[3. Графічний інтерфейс редактора з користувачем. Клавіатурний інтерфейс редактора з користувачем. + 59](#_Toc73383724)

[4. Прокручування поля, багатовіконність редактора, коректне припинення роботи.+ 60](#_Toc73383725)

[5. Сервісні можливості редактора: логічні групи команд; розтягання, стиснення, повороти; масштабування малюнка; контекстні і випливаючі меню; підказки про події на екрані; довідкова і навчальна підсистема редактора.+ 61](#_Toc73383726)

[6. Графічний програмний інструментарій: перо, пензель, шрифт. Загальні характеристики. + 62](#_Toc73383727)

[7. Операції читання/запису для графічних зображень. Формати збереження у файлах.+ 66](#_Toc73383728)

[8. Растрові і векторні методи малювання. Зберігання растрових і векторних зображень. Формати растрові і векторні.+ 72](#_Toc73383729)

[9. Принципи будови графічних редакторів на основі растрових і на основі векторних зображень. Характеристика особливостей растрового і векторного малювання, переваги і недоліки кожного методу. + 78](#_Toc73383730)

[Переваги та недоліки растрових і векторних зображень 80](#_Toc73383731)

[10. Події Windows, зв'язані з малюванням і відновленням зображень у вікні. Перелік подій та їх характеристика.+ 81](#_Toc73383732)

[11. Стандартні класи системи програмування для малювання. Класи без зберігання малюнка (без поля пам'яті) і класи зі зберіганням малюнка у власній пам'яті.+ 81](#_Toc73383733)

[12. Поняття про пересування та зміну розмірів видимих елементів вікна під час виконання програми.+ 85](#_Toc73383734)

[13. Події миші і клавіатури, які можна використати для пересування і зміни розмірів видимих елементів вікна під час виконання програми. Параметри подій.+ 86](#_Toc73383735)

[14. Об'єкт пересування та зміни розміру, зміщення об'єкта в процесі пересування, зовнішні розміри об'єкта, поточна канва малювання.+ 88](#_Toc73383736)

[15. Особливості малювання многокутників. Фіксування вершин полігону. Відображення проміжних полігонів. Динаміка малювання ребер полігона. Події, придатні до малювання полігона і відображення динаміки малювання.+ 89](#_Toc73383737)

[16. Масштабування зображень в процесі малювання. Проблеми, які можуть виникати при ручному програмуванні масштабування.+ 90](#_Toc73383738)

[17. Опрацювання зображень способом фільтрування. Приклади фільтрів та їх формули. Застосування фільтрів до частини загального зображення.+ 90](#_Toc73383739)

[18. Контекстні підказки про хід виконання графічних операцій. Перемикання контекстом підказки. 92](#_Toc73383740)

[Відповіді до питань до розділу "Електронні таблиці" 93](#_Toc73383741)

[1. Означення термінів "активна комірка", "впливаюча комірка", "залежна комірка", "ряд даних", "властивості комірки". Алгоритм розпізнавання статусу комірки. +- 93](#_Toc73383742)

[2. Формули в комірках ЕТ. Структура формули на рівні змістового призначення. Типи можливих синтаксичних і семантичних помилок в формулах. Критерії визначення помилки. + 93](#_Toc73383743)

[3. Граматики для розпізнавання типу комірки. Алгоритм розпізнавання. + 94](#_Toc73383744)

[4. Поняття лексеми формули. Перелік лексем на прикладі граматики формули. Формальне визначення лексем. + 95](#_Toc73383745)

[5. Загальна схема сканера. Визначення структур або класів для розпізнавання лексем сканером. Перелік параметрів лексем для їх опрацювання. + 96](#_Toc73383746)

[6. Алгоритми розпізнавання лексем формул у вигляді діаграми станів скінченного автомата. + 98](#_Toc73383747)

[7. Схема основного циклу сканування формули і кодування лексем. + 99](#_Toc73383748)

[8. Програмовані функції для кодування і декодування адресів комірок ЕТ. Зображення комірок як лексем. 103](#_Toc73383749)

[9. Перетворення формул ЕТ з інфіксної форми у постфіксну. Правила перетворення. Алгоритми, необхідні для перетворення. + 103](#_Toc73383750)

[10. Побудова постфіксної форми виразів на основі граматики виразів. Нисхідний граматичний розбір виразів методом рекурсивного спуску. 105](#_Toc73383751)

[11. Модифікація граматичних правил виразів для випадку розбору методом рекурсивного спуску. Питання ліво рекурсивних правил і факторизації. 105](#_Toc73383752)

[12. Алгоритм взаємодії функцій розпізнавання елементів виразів для перетворення формул ЕТ з інфіксної форми у постфіксну. 105](#_Toc73383753)

[13. Алгоритм фіксування помилок у формулах і друкування діагностики в процесі перетворення до постфіксної форми. 105](#_Toc73383754)

[14. Інтерпретація формул ЕТ на основі постфіксної форми зображення. Структура основного алгоритму інтерпретації. 105](#_Toc73383755)

[15. Алгоритм застосування стеку і правила опрацювання постфіксної форми виразів в процесі інтерпретації. 106](#_Toc73383756)

[16. Алгоритм опрацювання зв'язаних формул в процесі інтерпретації (залежна-впливаюча). 106](#_Toc73383757)

[17. Організація алгоритму інтерпретації прямо рекурсивної формули. Необхідні допоміжні параметри для коректної інтерпретації та їх використання 106](#_Toc73383758)

**ІСПИТ ОССП**

# Питання

**Список питань до розділу "Асемблери"**

1. Схема трансляції, компонування і виконання програми.

2. Основні регістри мікропроцесора.

3. Позиційна незалежність програми в однопрограмних ОС.

4. Режими адресування операндів в однопрограмних ОС.

5. Моделі адресування операндів в командах мікропроцесора.

6. Моделі структури програм: головна програма, підпрограма, COM-програма (команди + дані).

7. Загальний формат бітової структури команди процесора Intel. Поля коду команди.

8. Будова байта ModRM способу адресування для 16- і 32-бітових режимів.

9. Схеми виконання команд мікропроцесора: команди без операндів (нуль-операндні), команди операцій і команди дії з одним операндом.

10. Схеми виконання команд мікропроцесора: команди з двома операндами, команди з трьома операндами.

11. Директиви асемблера та їх застосування.

12. Трансляція програм ASM за першим і другим переглядом.

13. Алгоритм першого перегляду асемблера при трансляції.

14. Алгоритм другого перегляду асемблера при трансляції.

15. Загальні принципи компонування програм. Об’єктні файли.

16. Компонувальники і принципи їх роботи.

**Список питань до розділу "Бібліотеки DLL"**

1. Переваги і недоліки статичного компонування програм.

2. Визначення динамічної бібліотеки. Схема динамічного завантаження в адресний простір процесу.

3. Переваги і недоліки використання динамічних бібліотек.

4. Зворотна сумісність динамічних бібліотек.

5. Загальні принципи зв’язування з DLL в алгоритмічних мовах для неявного і явного зв’язування.

6. Взаємодія динамічної бібліотеки з адресним простором процесу. Особливості об'єктного коду динамічних бібліотек.

7. Точка входу динамічної бібліотеки і приклади її раціонального використання.

8. Структура виконуваних файлів для Windows. Формат РЕ.

9. Схема процесу компонування для Windows у разі неявного зв'язування.

10. Схема процесу компонування для Windows у разі явного зв'язування.

11. Механізми передавання параметрів до процедур і функцій: за значенням; за посиланням (за адресою); за поверненим значенням; за результатом; за іменем; відкладеним обчисленням. Загальні визначення.

12. Способи передавання параметрів до процедур і функцій: в регістрах; в глобальних змінних; в стеку. Погодження (конвенції) для передавання параметрів у функцію через стек.

13. Модель конвенції stdcall передавання параметрів до процедур і функцій.

14. Стандартні типи даних Windows та їх еквіваленти в мові C.

15. Процедура підготовки функцій для DLL на прикладі алгоритмічної мови (C++ чи іншої).

16. Компіляція функцій в бібліотеку DLL (створення DLL). Налаштування компілятора і завантажувача.

17. Використання DLL в прикладних програмах методом явного зв’язування (на прикладі мови C чи іншої).

18. Використання DLL в прикладних програмах методом неявного зв’язування (на прикладі мови C чи іншої).

**Список питань до розділу "Графічні редактори"**

1. Піксель. Роздільна здатність екрана. Палітра кольорів. Принцип малювання на екрані.

2. Графічні примітиви. Графічні бібліотеки. Принципи використання.

3. Графічний інтерфейс редактора з користувачем. Клавіатурний інтерфейс редактора з користувачем.

4. Прокручування поля, багатовіконність редактора, коректне припинення роботи.

5. Сервісні можливості редактора: логічні групи команд; розтягання, стиснення, повороти; масштабування малюнка; контекстні і випливаючі меню; підказки про події на екрані; довідкова і навчальна підсистема редактора.

6. Графічний програмний інструментарій: перо, пензель, шрифт. Загальні характеристики.

7. Операції читання/запису для графічних зображень. Формати збереження у файлах.

8. Растрові і векторні методи малювання. Зберігання растрових і векторних зображень. Формати растрові і векторні.

9. Принципи будови графічних редакторів на основі растрових і на основі векторних зображень. Характеристика особливостей растрового і векторного малювання, переваги і недоліки кожного методу.

10. Події Windows, зв'язані з малюванням і відновленням зображень у вікні. Перелік подій та їх характеристика.

11. Стандартні класи системи програмування для малювання. Класи без зберігання малюнка (без поля пам'яті) і класи зі зберіганням малюнка у власній пам'яті.

12. Поняття про пересування та зміну розмірів видимих елементів вікна під час виконання програми.

13. Події миші і клавіатури, які можна використати для пересування і зміни розмірів видимих елементів вікна під час виконання програми. Параметри подій.

14. Об'єкт пересування та зміни розміру, зміщення об'єкта в процесі пересування, зовнішні розміри об'єкта, поточна канва малювання.

15. Особливості малювання многокутників. Фіксування вершин полігону. Відображення проміжних полігонів. Динаміка малювання ребер полігона. Події, придатні до малювання полігона і відображення динаміки малювання.

16. Масштабування зображень в процесі малювання. Проблеми, які можуть виникати при ручному програмуванні масштабування.

17. Опрацювання зображень способом фільтрування. Приклади фільтрів та їх формули. Застосування фільтрів до частини загального зображення.

18. Контекстні підказки про хід виконання графічних операцій. Перемикання контекстом підказки.

**Список питань до розділу "Електронні таблиці"**

1. Означення термінів "активна комірка", "впливаюча комірка", "залежна комірка", "ряд даних", "властивості комірки". Алгоритм розпізнавання статусу комірки.

2. Формули в комірках ЕТ. Структура формули на рівні змістового призначення. Типи можливих синтаксичних і семантичних помилок в формулах. Критерії визначення помилки.

3. Граматики для розпізнавання типу комірки. Алгоритм розпізнавання.

4. Поняття лексеми формули. Перелік лексем на прикладі граматики формули. Формальне визначення лексем.

5. Загальна схема сканера. Визначення структур або класів для розпізнавання лексем сканером. Перелік параметрів лексем для їх опрацювання.

6. Алгоритми розпізнавання лексем формул у вигляді діаграми станів скінченного автомата.

7. Схема основного циклу сканування формули і кодування лексем.

8. Програмовані функції для кодування і декодування адресів комірок ЕТ. Зображення комірок як лексем.

9. Перетворення формул ЕТ з інфіксної форми у постфіксну. Правила перетворення. Алгоритми, необхідні для перетворення.

10. Побудова постфіксної форми виразів на основі граматики виразів. Нисхідний граматичний розбір виразів методом рекурсивного спуску.

11. Модифікація граматичних правил виразів для випадку розбору методом рекурсивного спуску. Питання ліво рекурсивних правил і факторизації.

12. Алгоритм взаємодії функцій розпізнавання елементів виразів для перетворення формул ЕТ з інфіксної форми у постфіксну.

13. Алгоритм фіксування помилок у формулах і друкування діагностики в процесі перетворення до постфіксної форми.

14. Інтерпретація формул ЕТ на основі постфіксної форми зображення. Структура основного алгоритму інтерпретації.

15. Алгоритм застосування стеку і правила опрацювання постфіксної форми виразів в процесі інтерпретації.

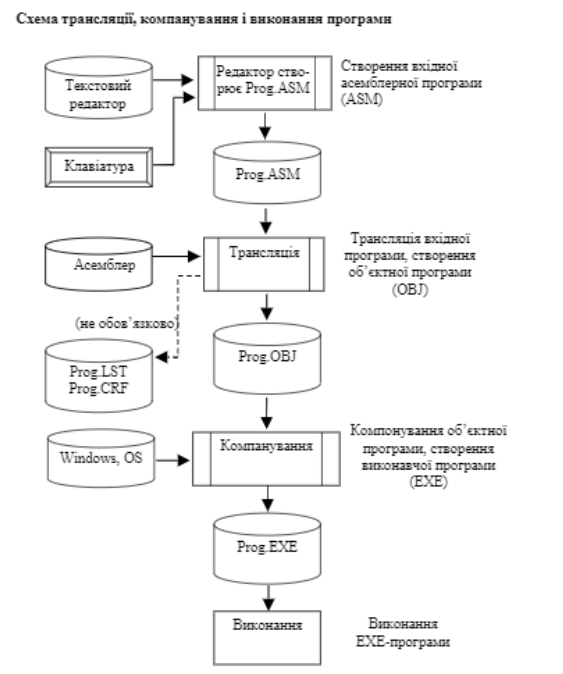
16. Алгоритм опрацювання зв'язаних формул в процесі інтерпретації (залежна-впливаюча).

17. Організація алгоритму інтерпретації прямо рекурсивної формули. Необхідні допоміжні параметри для коректної інтерпретації та їх використання

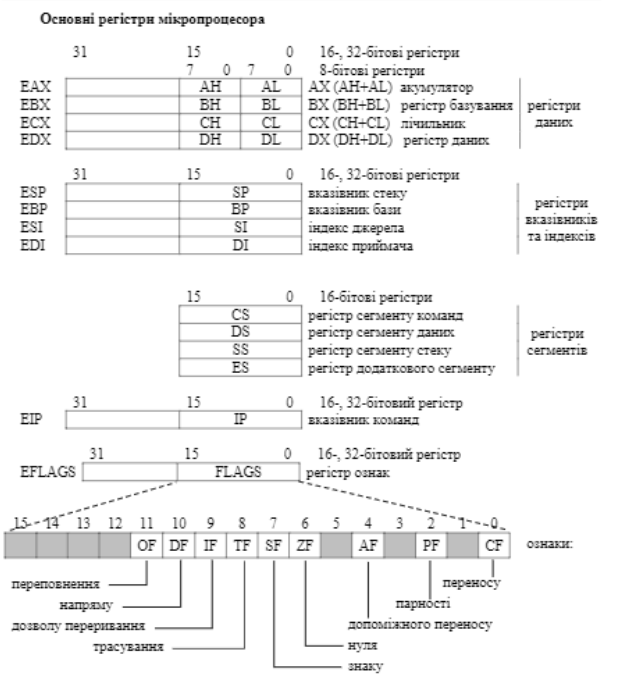
**Відповіді**

# Відповіді до питань до розділу "Асемблери"

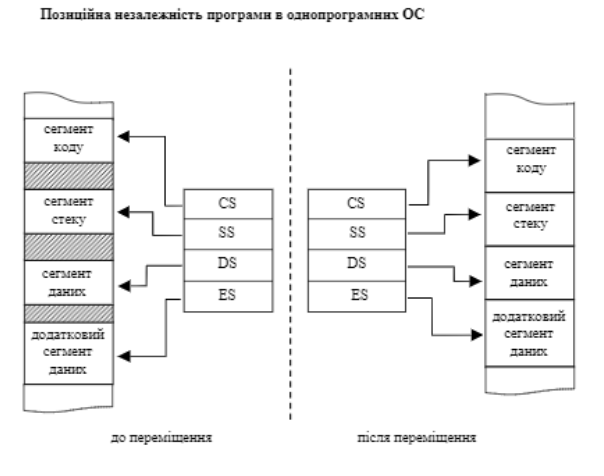
## 1. Схема трансляції, компонування і виконання програми. +



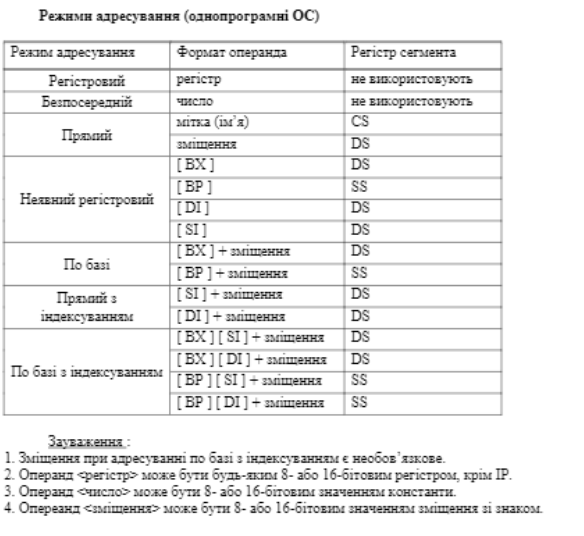
## 2. Основні регістри мікропроцесора. +



## 3. Позиційна незалежність програми в одно програмних ОС. +



## 4. Режими адресування операндів в одно програмних ОС. +



## 5. Моделі адресування операндів в командах мікропроцесора. +

Регістрове адресування

При регістровому адресуванні мікропроцесор вибирає операнд з регістра або завантажує (записує) операнд в регістр:

MOV AX , CX

Безпосереднє адресування

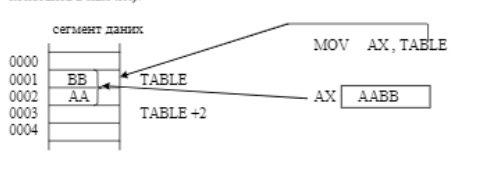
Дозволяє вказувати значення константи, як операнда джерела. Така константа записана в самій команді:

MOV CX , 500 K EQV 1024

MOV AL , -30 MOV CX , K

Пряме адресування

Найчастіше застосовують, коли операндом є окрема комірка пам’яті (проста змінна або константа в пам’яті):



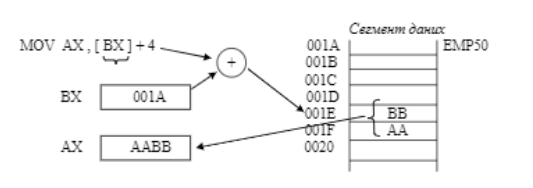
Неявне регістрове адресування

Адреса операнда міститься в одному з регістрів BX, BP, SI, DI :



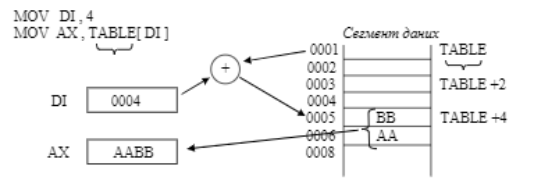
Адресування по базі

Адреса є сумою вмісту регістра ВХ або ВР та зміщення. Зручно використовувати для доступу до структурованих записів даних, розташованих в різних ділянках пам'яті:



Пряме адресування з індексуванням

Адреса є сумою вмісту регістра SI або DI та зміщення. Зручно використовувати для доступу до елементів таблиці (вектора):



Адресування по базі з індексуванням

Зручно використовувати для двовимірних масивів: базовий регістр–початкова адреса, індексний регістр–зміщення до початку відповідного рядка, зміщення–зміщення від початку рядка до потрібного елементу (або зміщення, базовий регістр, індексний регістр відповідно).Адреса є сумою вмісту базового регістру, індексного регістру і, можливо, зміщення. Зручно також використовувати для адресування масиву, розташованого в стеку.

## 6. Моделі структури програм: головна програма, підпрограма, COM-програма (команди + дані). +

1. Головна програма

<оператор EXTRN, якщо потрібний>

STACK SEGMENT PARA STACK ‘STACK’ ; сегмент стеку

DB64 DUP(‘STACK\_\_\_’) ; ділянка пам’яті стеку

STACK ENDS

DSEG SEGMENT PARA PUBLIC ‘DATA’ ; сегмент даних

< тут записують дані >

DSEGENDS

; основна програма

CSEG SEGMENT PARA PUBLIC ‘CODE’ ; сегмент команд

ASSUMECS:CSEG, DS: DSEG, SS: STACK ; зв’язування сегм. регістрів

ENTRY PROCFAR ; функція: точка входу в програму

; формування стеку для правильного повернення в DOS(для емулятор DOS не потрібний)

PUSH DS

SUB AX,AX

PUSH AX

; встановити адресу сегменту даних(в захищеному режимі сегментні регістри не можна зачіпати)

MOV AX, DSEG

MOV DS, AX

< тут записують команди програми >

RET; повернення в DOS(MOV AX,4C00h ; функція 4C (76)

INT 21h ; функція DOS–вихід з програми)

ENTRY ENDP ; кінець функції

CSEG ENDS ; кінець сегменту команд

END ENTRY; кінець головної програми і точка входу

2. Підпрограма

PUBLIC PNAME

<оператор PUBLIC для змінних сегменту даних, якщо потрібно >

DSEG SEGMENT PARA PUBLIC ’DATA’ ; сегмент даних(продовження)

< тут записують дані>

DSEG ENDS

CSEG SEGMENT PARA PUBLIC ‘CODE’ ; сегмент команд (продовження)

ASSUME CS:SCEG, DS:DSEG; зв’язування сегм. Регістрів

PNAME PROC NEAR; процедура ближнього типу NEAR

< тут записують команди підпрограми >

RET ; повернути в основну програму

PNAME END

PCSEG END

SEND ; кінець підпрограми

3. COM-програма (команди + дані)

CODE SGSEGMENT PARA ‘CODE’

ASSUME CS : CODESG,DS:CODESG,SS:CODESG,ES:CODESG

ORG 100H ; резерв для PSP-префікса прогр. Сегмента

BEGIN: JMP MAINPROG; обхід даних

MESS1 DB ‘Тестовий приклад: введіть рядок’

DB ‘$’

MAINPROG PROC NEAR ; процедура ближнього типу NEAR

MOV AH,09 ; друкувати на екран рядок до ‘$’

LEA DX,MESS1; адреса початку рядка запрошення

INT 21H

CALL READLINE ; виклик підпрограми

CALL WRITELN

RET

MAINPROG ENDP

;процедура читання рядка

READLINE PROC NEAR

MOV AH,0AH; прочитати з клавіатури рядок до Enter

LEA DX,MAXLEN ; адреса списку параметрів

INT 21H ; виконати DOS-функцію 0A(10)

RET ;

READLINE ENDP

;пам’ять та дані для процедури читання рядка

MAXLEN DB 30; максимальна довжина введеного рядка

REALLEN DB ? ; кількість фактично введених символів

POLE DB 30 DUP‘\_’;поле ( 30 літер ) для введених символів

; процедура виведення на екран

WRITELN PROC NEAR

MOV AH,09

LEA DX,MESS2; рядок-повідомлення

INT 21H

MOV AH,40H;друкуватирядок вказаної довжини

MOV BX,1;на екран

MOV CX,30 ; 30 символів

LEA DX,POLE ; адреса початку рядка

INT 21H

RET

MESS2 DB ‘Ви ввели такий рядок : ’

DB ‘$’

WRITELN ENDP

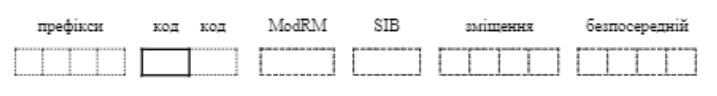
CODES GENDS

END BEGIN ; кінець програми і точка входу

## 7. Загальний формат бітової структури команди процесора Intel. Поля коду команди. +

**Загальний формат команди процесора Intel**

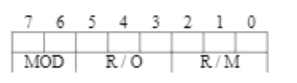
Команда може мати до шести полів:



1. Префікси –від нуля до чотирьох однобайтових префіксів.

2. Код –один або два байти, що визначають команду.

3. ModRM–1 байт способу адресування (якщо він потрібний), який описує операнди(нумерація бітів справа наліво починаючи з нуля):

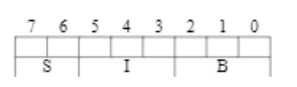


біти 7-6: поле MOD–режим адресування;

біти 5-3: поле R/O–або визначає регістр, або є продовженням коду команди;

біти 2-0: поле R/M–або визначає регістр, або разом з MOD–режим адресування.

4. SIB–1 байт, якщо він потрібний (розширення ModRMдля 32-бітового адресування):



біти 7-6: S–коефіцієнт масштабування;

біти 5-3: I–індексний регістр;

біти 2-0: B–регістр бази.

5. Зміщення –0, 1, 2 або 4 байти.

6. Безпосередній операнд –0, 1, 2 або 4 байти.

**Значення полів коду команди**

У кодах деяких команд зустрічаються спеціальні біти і групи бітів, котрі позначають w, s, d, reg, sreg, cond:

w= 0, якщо команда працює з байтами;

w= 1, якщо команда працює з словами або подвійними словами;

s= 0, якщо безпосередній операнд записаний в команді повністю;

s= 1, якщо безпосередній операнд є молодшим байтом більшого операнда і має розглядатись як число зі знаком;

d= 0, якщо код джерела записаний в полі R/O, а приймача –в полі R/M;

d= 1, якщо код джерела записаний в полі R/M, а приймача –в полі R/O.

Поле reg визначає потрібний регістр і має довжину 3 біти:

000 –AL / AX / EAX / ST(0) / MM0 / XMM0

001 –CL / CX / ECX / ST(1) / MM1 / XMM1

010 –DL / DX / EDX / ST(2) / MM2 / XMM2

011 –BL / BX / EBX / ST(3) / MM3 / XMM3

100 –AH / SP / ESP / ST(4) / MM4 / XMM4

101 –CH / BP / EBP / ST(5) / MM5 / XMM5

110 –DH / SI / ESI / ST(6) / MM6 / XMM6

111 –BH / DI / EDI / ST(7) / MM7 / XMM7

Поле sreg визначає потрібний сегментний регістр:

000 –ES

001 –CS

010 –SS

011 –DS

100 –FS

101 –GS

Поле cond визначає умову для команд Jcc, CMOVcc, SETcc, FCMOVcc. Воно має такі значення для різних команд:

0000 –O

0001 –NO

0010 –C/ B/ NAE

0011 –NC / NB / AE

0100 –E / Z

0101 –NE / NZ

0110 –BE / NA

0111 –NBE / A

1000 –S

1001 –NS

1010 –P / PE

1011 –NP / PO

1100 –L / NGE

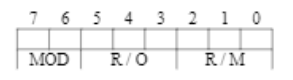
1101 –NL / GE

1110 –LE / NG

1111 –NLE / G

## 8. Будова байта ModRM способу адресування для 16- і 32-бітових режимів. +

**Значення полів байта ModRM способу адресування**



Поле R/O(біти 5-3) має або додаткові три біти коду команди, або код операнда, який є регістром. В таблицях команд другий випадок позначають reg, а в першому записують потрібні біти.

Поля MOD(біти 7-6) і R/M(біти 2-0) визначають операнд, який може бути як регістром, так і коміркою пам’яті:

MOD= 11, якщо використовується регістрове адресування і поле R/M має код регістра reg;

MOD= 00, якщо використовується адресування в пам’яті без зміщення ([BX+SI]або [EDX]), тобто неявне адресування через регістри;

MOD= 01, якщо використовується адресування в пам’яті з 8-бітовим зміщенням (var[BX+SI]чи [BX+SI]+число);

MOD= 10, –те ж саме, що й MOD=01, тільки зміщення 16-бітове або 32-бітове.

Значення поля R/M відрізняється в 16-і 32-бітових режимах.

**R/M в 16-бітовому режимі:**

000 –[BX + SI]

001 –[BX + DI]

010 –[BP + SI]

011 –[BP + DI]

100 –[SI]

101 –[DI]

110 –[BP](крім MOD=00–в цьому випадку після байта ModRM записується 16-бітове зміщення, тобто використовується пряме адресування: ADD DX, Table)

111 –[BX]

**R/M в 32-бітовому режимі:**

000 –[EAX]

001 –[ECX]

010 –[EDX]

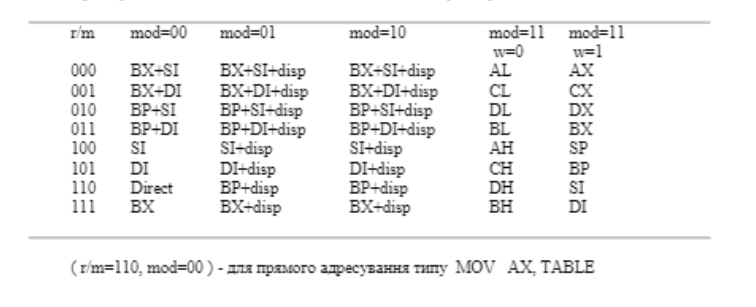
011 –[EBX]

100 –використовується байт SIB

101 –[EBP](крім MOD=00–в цьому випадку після байта ModRM записується 32-бітове зміщення, тобто використовується пряме адресування: ADD EDX, Table)

110 –[ESI]

111 –[EDI]

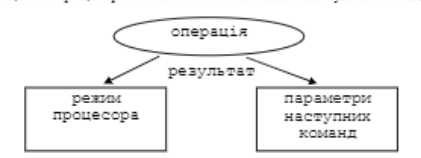


## 9. Схеми виконання команд мікропроцесора: команди без операндів (нуль-операндні), команди операцій і команди дії з одним операндом. +

**Схеми виконання команд мікропроцесора**

Команда може мати нуль, один, два або три операнди. Загальні правила визначають способи адресування операндів і місце для результату виконання команди.

**1)Команди без операндів (нуль-операндні).** Такі команди призначені для керування режимом роботи цілого процесора або способом виконання наступних команд.



Приклади команд:

REP - префікс; означає повторення рядкової операції наступної команди ECX разів; різновиди: REPZ(REPE)–виконувати так само, поки виконується умова рівності (ZF=1), закінчити повторення, якщо після чергової ітерації ZF=0(до знайдення іншого значення);REPNZ(REPNE)-так само, але закінчити приZF≠0 (до знайдення такого самого значення)

CLD - ознака напряму DFрегістра ознак встановлюється в нуль; це означає автоматичне збільшення SIі DI на 1 за кожною ітерацією операцій над рядками

STD - DF=1, SI і DI зменшуються на 1 за кожною ітерацією

**2)Нуль-операндні команди з неявними операндами,** визначеними окремо. Такі команди виконують групу елементарних операцій за одне виконання, самі операнди визначають попередньо або зафіксовані командою.



Приклади команд:

XLAT - завантажити в AL байт з таблиці сегменту даних, на початок якої вказує EBX (BX), а початкове значення AL є зміщенням; тобто, це є заміна одного коду іншим за таблицею

PUSHAD - записати в стек регістри EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP

CWD - розширення слова (AX)до подвійного слова (DX:AX) з копіюванням знакового біту

CWDE - розширення слова (AX)до подвійного слова (EAX) з копіюванням знакового біту

**3)Команди операцій з одним операндом.** Команда опрацьовує заданий операнд відповідно до свого призначення, результат записує на місце операнда. Отже, після виконання команди операнд буде модифікований.

Приклади команд:

INC EAX - інкремент операнда

NEG testvalue - змінити знак операнда

**4)Команди дії за одним операндом.** Такі команди виконують задану дію, використовуючи операнд. Сам операнд не міняється.

Приклади команд:

PUSH EDI - записати в стек значення регістра

POP memx - прочитати з стеку слово чи подвійне слово

JMP target - перейти в програмі до вказаної адреси (різні форми адресування)

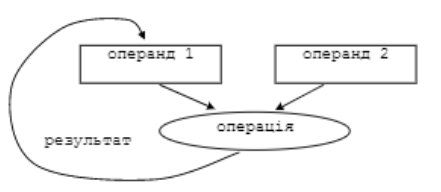
JGE / JNL target - перейти, якщо більше або рівне

CALL target - передача керування процедурі за вказаною адресою; в стек зберігається адреса команди, наступної за CALL-командою

INT n - двобайтова команда; спочатку в стек записує регістр ознак, після нього –повну адресу повернення; крім того, скидає ознаку TF; після цього виконує неявний перехід через n-й елемент дескрипторної таблиці переривань

## 10. Схеми виконання команд мікропроцесора: команди з двома операндами, команди з трьома операндами. +

**5)Команди з двома операндами.** Можливі комбінації операндів розглянуто далі. Результат записують на місце першого операнда, отже, він набуває нового значення, тому порядок запису операндів команди є важливий.



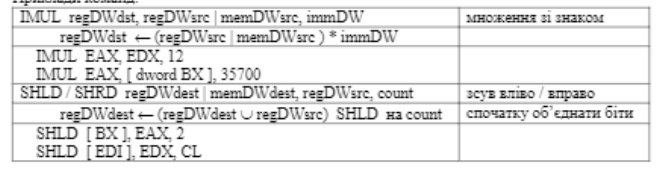
Приклади команд:

ADDEDX,alfa - додавання, результат –в регістрі EDX

OR beta,ECX - логічне порозрядне «або», результат –в пам’яті за адресою beta

**6)Команди з трьома операндами.** Такі команди є лише для окремих різновидів операцій. Результат буде на місці операнда, визначеного структурою команди.

Приклади команд:



## 11. Директиви асемблера та їх застосування. +

В простішій формі директиви виглядають так:

<ім’я> SEGMENT - визначає ім’я сегмента (частини) програми

< ім’я>ENDS - вказує на кінець сегмента

< ім’я>PROC - фіксує початок процедури

< ім’я>ENDP - фіксує кінець процедури

END< ім’я> - вказує на кінець тексту програми і визначає першу команду програми (точка входу)

ORG < число> - зміна лічильника адреси до вказаного числа, і, відповідно, адреси наступного елемента програми; LOCCTR:= <число>

[< ім’я > ] DW <число > - формує двобайтову константу –число; константа може бути іменована, тобто мати власну адресу, або неіменована

[< ім’я > ] DW ? - резервує 2 байти (слово) пам’яті

[< ім’я > ] DB <число > - формує однобайтову константу -число

[< ім’я > ] DB ? - резервує 1 байт пам’яті

[< ім’я > ] DB 'рядок' - формує в пам’яті заданий рядок; довжина рядка (кількість байтів) дорівнює кількості літер рядка

[< ім’я >] DD< значення> [< ім’я > ] DD ? - аналогічно до вказаного вище, визначає подвійне слово (4 байти)

[< ім’я > ] DF < значення> [< ім’я > ] DF ? - аналогічно, визначає 6 байтів (адреса в форматі 16-бітний селектор і 32-бітове зміщення

[< ім’я > ] DQ<значення> [< ім’я > ] DQ ? - аналогічно, визначає почетверене слово (8 байтів)

[< ім’я > ] DT< значення>[< ім’я > ] DT ? - визначити 10 байтів (80-бітові типи даних, які використовує FPU для операцій з форматами дійсних чисел)

В складнішій формі поле значення може мати декілька чисел, рядків літер, операторів ? і DUP, розділених комами. Всі такі дані будуть записані в пам’ять підряд, а ім’я буде відповідати адресі найпершого значення.

Наприклад:

table db 1,2,3,4,5,6,7,8,9,0AH,0BH,0CH,0DH,0EH,0FH

tabtwo db 512 dup (?)

## 12. Трансляція програм ASM за першим і другим переглядом. +

**За першим переглядом** транслятор послідовно рядок за рядком переглядає текст програми. При цьому виконує таку роботу:

1)керує лічильником поточної адреси LOCCTR;

2)обчислює довжину кожної команди чи даних в байтах;

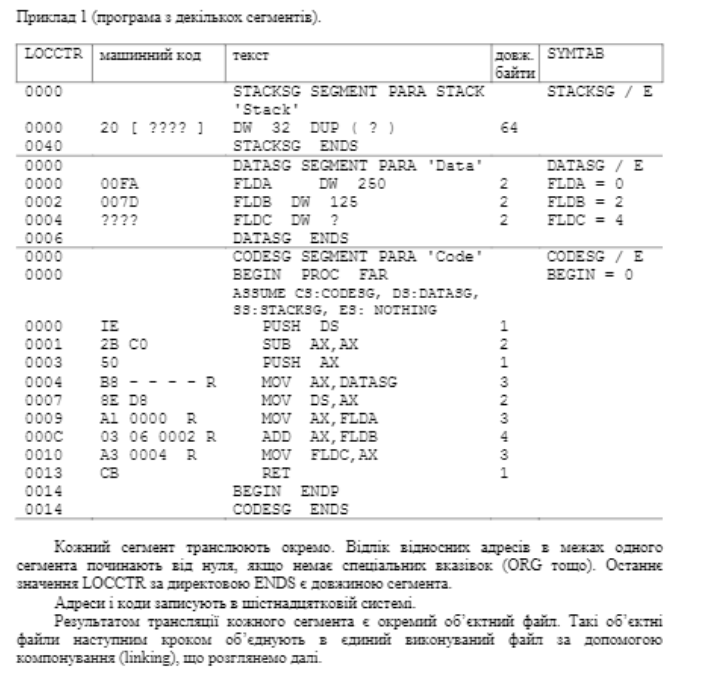
3)відстежує адресу (зміщення) кожної команди і даних в сегменті;

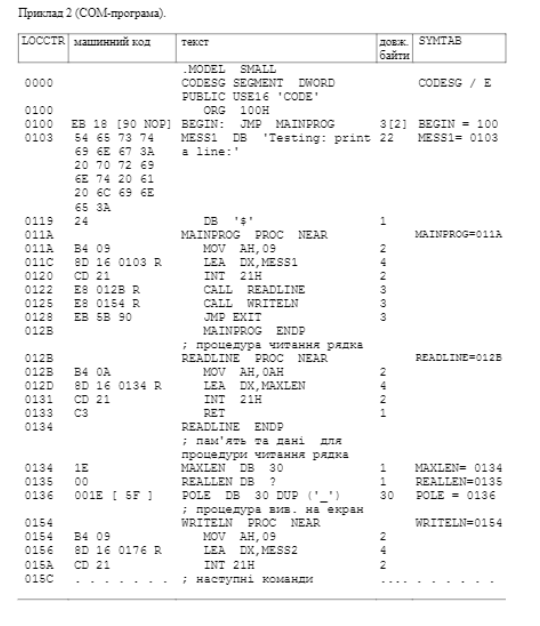
4)виконує основне перетворення в машинні коди за кожною командою чи даними;

5)будує таблицю символів SYMTAB з адресами всіх іменованих ділянок пам’яті, які будуть дописані в коди команд за другим переглядом.

**За другим переглядом** транслятор виконує підстановку обчислених адресів операндів пам’яті, які побудовані в таблиці SYMTAB в результат і першого перегляду.

Далі записані два приклади програм з показом процедури трансляції за першим і ж другим переглядом.





## 13. Алгоритм першого перегляду асемблера при трансляції. +

begin{ pass 1 }

записати 0 в LOCCTR; OKEY:= true

прочитати перший рядок тексту програми

while(OPCODE<>"END" and OKEY) do begin

If це не рядок коментар then begin

if є поле мітки then begin

пошук мітки в SYMTAB

If знайшли then begin

встановити ознаку помилки-повторне визначення

OKEY:=false

end

Else занести (< мітка >,LOCCTR) в SYMTAB

End

пошук OPCODE в OPTAB

If знайшли then begin

аналіз поля операндів

If формат правильний then begin

визначення формату та довжини команди

додати до LOCCTR довжину команди

End

Else OKEY:=false(помилка воператорах)

If OKEY then перевірка наявності та правильності коментаря

end{ знайшли код операції }

Else if OPCODE="ORG" then LOCCTR:=< число >{ операнд ORG }

Else if OPCODE="DW" then додати 2 до LOCCTR

Else if OPCODE="DB" then begin

аналіз поля операндів DB, визначення довжини

додати довжину до LOCCTR

End

Else встановити ознаку помилки -неправильний код операції

(OKEY:=false)

end{ if це не рядок-коментар }

записати рядок в проміжний файл

прочитати наступний рядок тексту програми

end{ while}

If not OKEY then сформувати діагностику про помилку

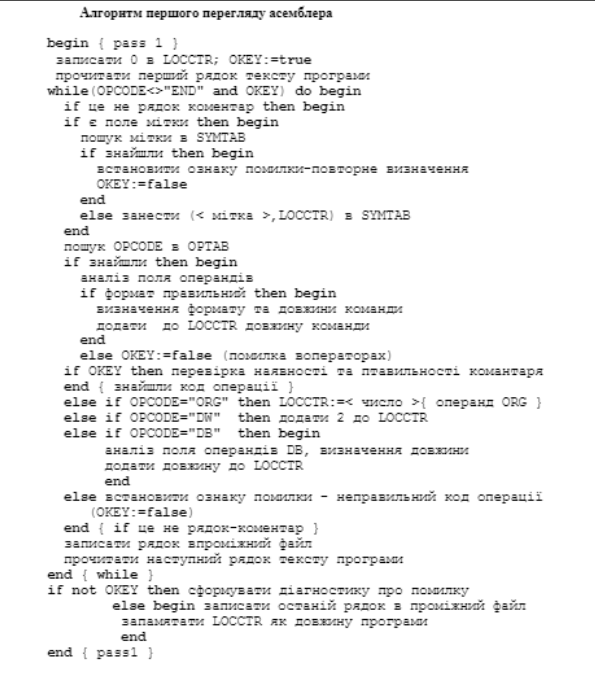
Else begin записати останій рядок в проміжний файл

запамятати LOCCTR як довжину програми

End

end{ pass1 }

ТО САМЕ



## 14. Алгоритм другого перегляду асемблера при трансляції. +

begin{ pass2 }

сформувати запис-заголовок в об'єктній програмі

прочитати перший рядок з проміжного файлу

ініціалізувати перший запис тіла програми

OKEY:=true

While (OPCODE<>"END") and OKEY do begin

If це не рядок-коментар then begin

пошук OPCODE в OPTAB

If знайшли then begin

аналіз поля операндів

If в полі операндів є імя then begin

пошук імені в SYMTAB

If знайшли then взяти з SYMTAB адресу

Else OKEY:=false( не визначене імя )

end

перевести команду в об'єктне зображення

end( ifзнайшли )

Else if OPCODE="DW" або "DB" then

перетворити константу або рядок в об'єктне зображення

занести об'єктний код в запис тіла програми

ініціалізувати новий запис тіла програми

end{ if це не рядок-коментар }

записати у файл для роздруку черговий рядок

прочитати наступний рядок з промідного файлу

end{ while}

If not OKEY then сформувати діагностику про помилку

Else begin

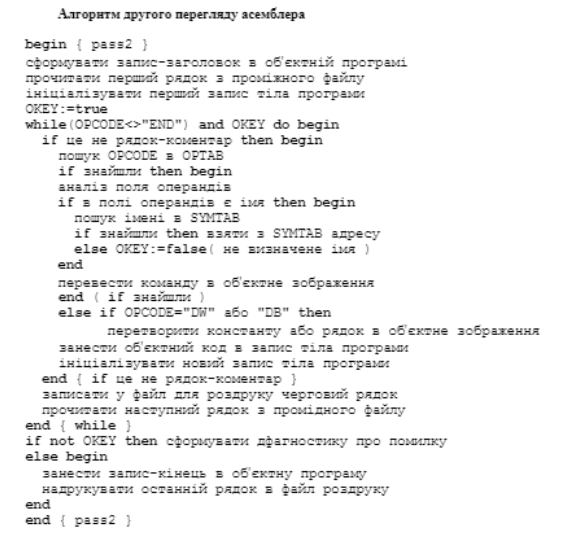
занести запис-кінець в об'єктну програму

надрукувати останній рядок в файл роздруку

End

end{ pass2 }

ТО САМЕ



## 15. Загальні принципи компонування програм. Об’єктні файли. +

**Компонуванням(linking)** називають процес створення фізичного або логічного виконуваного файла (модуля) з набору об’єктних файлів і файлів бібліотек для наступного виконання. У разі створення фізичного виконуваного файла компонування називають статичним, в такому файлі є все потрібне для виконання програми. У разі створення логічного виконуваного файла в часі виконання програми компонування називають динамічним, в цьому випадку образ виконуваного модуля будують «на ходу».

Тут ми розглядаємо статичне компонування виконуваних файлів. Динамічне компонування пов’язане з використанням динамічних бібліотек DLL, це питання будемо розглядати окремим розділом курсу.

**Об’єктні файли**

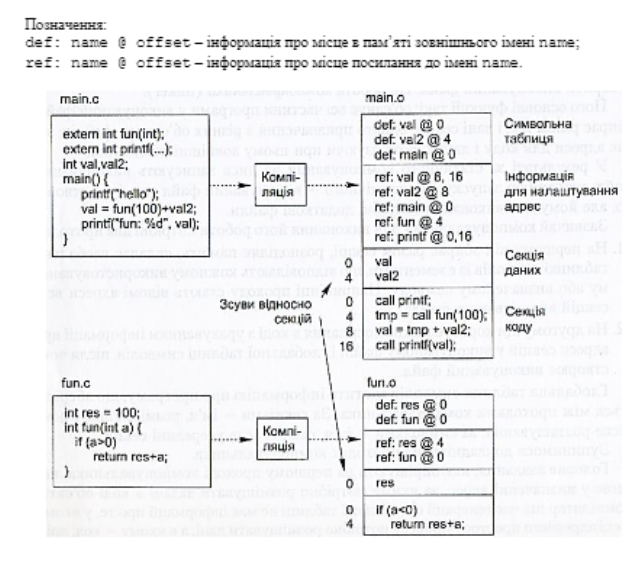
Під час компонування виконуваний файл будують з об’єктних файлів (object files), які створює компілятор. Об’єктний файл має:

1)заголовок, де визначено розмір ділянок коду і даних, а також позицію таблиці символів в цьому файлі;

2)об’єктний код(інструкції і дані, згенеровані компілятором), який зазвичай розділений на іменовані ділянки (секції) залежно від призначення;

3)таблицю символів(symbol table), де визначено відносні адреси кожної іменованої комірки пам’яті програми.

Приклад створення об’єктних файлів показано(без заголовку)на рисунку.



## 16. Компонувальники і принципи їх роботи.+

Компілятор не розв’язує зовнішні посилання, а отже, не може створити виконуваний файл. Це робота компонувальника(linker).

Основні функції компонувальника:

-об’єднати всі частини програми у виконуваний файл;

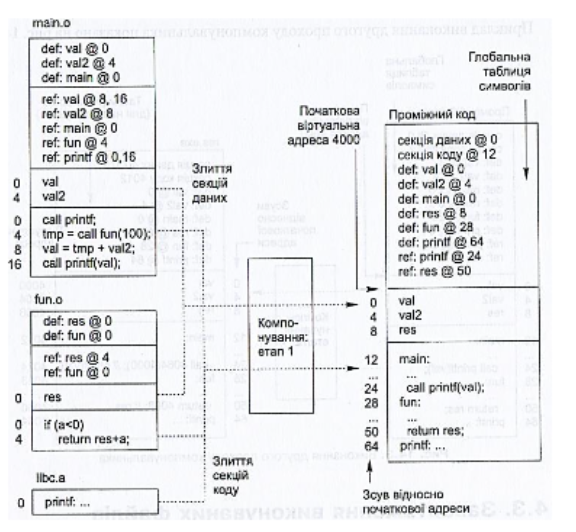
-зібрати разом код і дані секцій однакового призначення з різних об’єктних файлів;

-визначити остаточно всі адреси для коду і даних, розв’язуючи при цьому зовнішні посилання.

В результаті за статичного компонування на диск записують виконуваний файл, готовий до запуску (EXE-файл), за динамічного –виконуваний файл так само буде створений, але йому для виконання потрібні додаткові файли.

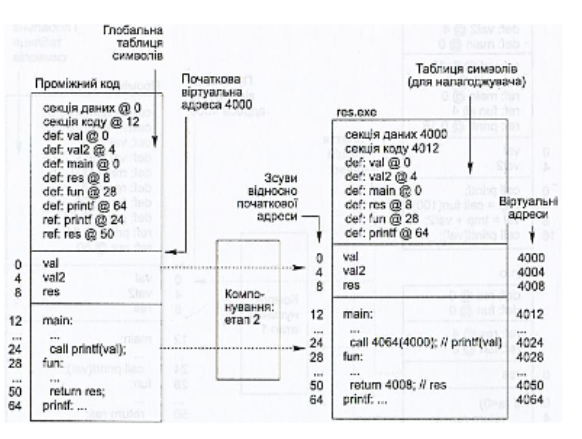
Головне завдання першого проходу компонувальника –визначити відносні адреси, за якими треба розташувати задані в коді об’єкти.

Приклад виконання першого проходу показано на рисунку.



Головне завдання другого проходу компонувальника –корекція всіх адрес в об’єктному коді та розв’язання всіх зовнішніх посилань. Перевіряє, щоб кожний символ мав тільки одне визначення (використати можна багато разів), і виконує корекцію посилань фактичними адресами, з врахуванням отриманої початкової віртуальної адреси.

Приклад виконання другого проходу показано на рисунку.



**Завантаження виконуваних файлів за статичного компонування**

Виконуваний файл, отриманий внаслідок описаного статичного компонування, має все необхідне для створення процесу. Завантаження такого файла в пам’ять виконує окремий компонент ОС –системний завантажувач. Зазвичай він відображає виконуваний файл в адресний простір процесу та ініціалізує керуючий блок процесу так, щоб процес був в стані готовності до виконання.

Під час відображення виконуваного файла в пам’яті автоматично розташовують його код та ініціалізовані дані. Стек і динамічну ділянку пам’яті зазвичай створюють заново, при цьому для динамічної ділянки компілятор і компонувальник можуть лише задати її початок, а всю інформацію для керування стеком визначає компілятор використовуючи адресування щодо вказівника стеку, який буде встановлений завантажувачем.

# Відповіді до питань до розділу "Бібліотеки DLL"

## Переваги і недоліки статичного компонування програм. (+)

Статичне компонування виконуваних файлів має низку недоліків.

1. Якщо декілька різних прикладних програм використовують спільний код (наприклад, коди функцій бібліотеки мови С), кожний виконуваний файл буде мати окрему копію цього коду; в результаті такі файли займатимуть більше місця на диску і в пам’яті. Варто мати можливість зберігати на диску і завантажувати в пам’ять лише одну копію спільного коду.
2. У разі кожного оновлення прикладну програму потрібно повністю перекомпілювати, перекомпонувати, повторно тестувати і перевстановити.
3. Неможливо реалізувати динамічне завантаження програмного коду під час виконання, наприклад, якщо потрібно реалізувати модульну структуру програми, подібно до того, як це зроблено в ядрі Linux.

Основною перевагою статичного зв'язування у порівнянні з динамічним є те, що програма є незалежною від зовнішніх файлів бібліотек. Іншими словами, всі бібліотечні функції вже містяться всередині програми, і їх версії є саме тими, які використовувалися при побудові програми. Таким чином усувається пекло динамічних бібліотек (або іншими словами пекло залежностей), а дистриб’юція і встановлення програми максимально спрощуються.

## Визначення динамічної бібліотеки. Схема динамічного завантаження в адресний простір процесу. (+)

Динамічна бібліотека – набір функцій та програмних ресурсів, скомпонованих разом в бінарний файл, який можна динамічно завантажити в адресний простір процесу, що використовує ці функції.

Динамічне завантаження (dynamic loading) – завантаження під час виконання процесу, зазвичай реалізоване як відображення файла бібліотеки в його адресний простір.

Динамічне компонування (dynamic linking) – компонування образу виконуваного файла під час виконання процесу з використанням динамічних бібліотек.

## Переваги і недоліки використання динамічних бібліотек. (+)

***Переваги***

► Оскільки бібліотечні функції містяться в окремому файлі, розмір виконуваного файла стає меншим. Якщо врахувати, що є динамічні бібліотеки, які використовують майже всі застосування у системі (стандартна бібліотека мови С у Linux, бібліотека підсистеми Win32 у Windows), то очевидно, що так заощаджують дуже багато дискового простору.

► Якщо динамічну бібліотеку використовують кілька процесів, у пам'ять завантажують лише одну її копію, після чого сторінки коду бібліотеки відображаються в адресний простір кожного з цих процесів. Це дає змогу ефективніше використовувати пам'ять.

► Оновлення застосування може бути зведене до встановлення нової версії динамічної бібліотеки без необхідності перекомпонування тих його частин, які не змінилися.

► Динамічні бібліотеки дають змогу застосуванню реалізувати динамічне завантаження модулів на вимогу. На базі цього може бути реалізований розширюваний АРІ застосування. Для додавання нових функцій до такого АРІ стороннім розробникам достатньо буде створити і встановити нову динамічну бібліотеку, яка підлягає певним правилам.

► Динамічні бібліотеки дають можливість спільно використовувати ресурси застосування (наприклад, така бібліотека може містити спільний набір піктограм), крім того, вони дають змогу спростити локалізацію застосування (якщо всі рядки, які використовуються програмою, помістити в окрему DLL, для заміни мови застосування достатньо буде замінити тільки цю DLL).

► Оскільки динамічні бібліотеки є двійковими файлами, можна організувати спільну роботу бібліотек, розроблених із використанням різних мов програмування і програмних засобів, що спрощує створення застосувань на основі програмних компонентів (отже, динамічне компонування лежить в основі компонентного підходу до розробки програмного забезпечення).

***Недоліки.*** Динамічні бібліотеки не позбавлені недоліків (хоча вони тільки в окремих випадках виправдовують використання статичного компонування).

► Використання DLL сповільнює завантаження застосування. Що більше таких бібліотек потрібно процесу, то більше файлів треба йому відобразити у свій адресний простір під час завантаження, а відображення кожного файла забирає час. Для прискорення завантаження рекомендують укрупнювати DLL, об'єднуючи кілька взаємозалежних бібліотек в одну загальну.

► У деяких ситуаціях (наприклад, під час аварійного завантаження системи із дискети) використання спільних системних DLL неприйнятне через нестачу дискового простору для їхнього зберігання (такі системні DLL, подібно до стандартної бібліотеки мови С, можуть займати кілька мегабайтів дискового простору, при цьому застосування часто потребують усього по кілька функцій із них). У такій ситуації найчастіше використовують версії застосувань, статично скомпоновані таким чином, щоб у їхні виконувані файли був включений зі стандартних бібліотек код лише тих функцій, які їм потрібні.

► Найбільшою проблемою у використанні динамічного компонування є проблема зворотної сумісності динамічних бібліотек. Розглянемо її окремо.

## Зворотня сумісність динамічних бібліотек. (+)

Ця проблема виникає в ситуації, коли прикладна програма встановлює нову версію DLL поверх попередньої. Якщо нова версія не має зворотної сумісності з попередніми, прикладні програми, розраховані на використання попередніх версій бібліотеки, можуть припинити роботу. Досягти такої сумісності досить складно, особливо коли попередня версія містила відомі помилки, і прикладні програми, що використовують бібліотеку, розробили код їхнього обходу – виправлення помилки у бібліотеці може зробити код прикладної програми невірним (кажуть, що в цьому разі порушується сумісність за помилками – bug-to-bug compatibility).

Деякі ОС (переважно це стосується Windows-систем, але певні проблеми є й в UNIX) ускладнювали цю проблему через те, що не давали можливості кільком версіям однієї й тієї самої бібліотеки одночасно бути завантаженими у пам'ять; виділяли для динамічних бібліотек усіх застосувань спільний каталог, цим «запрошуючи» застосування перезаписувати динамічні бібліотеки один одного (можлива була навіть ситуація, коли стару версію бібліотеки записували поверх нової); не зберігали в динамічних бібліотеках і застосуваннях інформації про точні версії бібліотек, від яких вони залежать.

## Загальні принципи зв’язування з DLL в алгоритмічних мовах для неявного і явного зв’язування. (+)

Є два основні способи завантаження динамічних бібліотек в адресний простір процесу – неявне і явне зв'язування (implicit і explicit binding).

Неявне – основний спосіб завантаження динамічних бібліотек у сучасних ОС. При цьому бібліотеку завантажують автоматично до початку виконання застосування під час завантаження виконуваного файла, за це відповідає завантажувач виконуваних файлів ОС. У деяких системах такий завантажувач є частиною ядра ОС, у деяких – окремим застосуванням. Список бібліотек, потрібних для завантаження, зберігають у виконуваному файлі.

До переваг цього методу належать:

► простота і прозорість з погляду програміста (йому не потрібно писати код завантаження бібліотек, достатньо у налаштуваннях компонувальника вказати список бібліотек, які йому потрібні);

► висока ефективність роботи процесу після початкового завантаження (усі необхідні бібліотеки до цього часу вже завантажені у його адресний простір).

Недоліком неявного зв'язування можна вважати зниження гнучкості (так, наприклад, якщо хоча б однієї з необхідних бібліотек не буде на місці, процес завантаження не обійдеться без проблем, навіть коли для виконання конкретної задачі ця бібліотека не потрібна). Крім того, збільшуються час завантаження і початковий обсяг необхідної пам'яті.

Альтернативним для неявного є явне зв'язування, коли динамічну бібліотеку завантажують в адресний простір процесу виконанням системного виклику з його коду. Після цього, використовуючи інший системний виклик, застосування отримує адресу необхідної йому функції бібліотеки і може її викликати. Після використання бібліотеку можна вилучити з пам'яті. Компонувальних при цьому нічого про неї не знає, завантажувач ОС автоматично бібліотек не завантажує (здебільшого неявне зв'язування зводиться до автоматичного виконання тих самих викликів, які сам програміст виконує за явного).

Такий підхід вимагає від програміста додаткових зусиль, але має більшу гнучкість.

## Взаємодія динамічної бібліотеки з адресним простором процесу. Особливості об'єктного коду динамічних бібліотек. (+)

**Динамічні бібліотеки та адресний простір процесу**

Після того, як динамічна бібліотека була відображена в адресний простір процесу, вона стає майже прозорою для програмного коду, що тут виконується.

Усі функції бібліотеки стають доступними для всіх потоків цього процесу, фактично її код і дані набувають вигляду доданих до адресного простору процесу. Зазначимо, що під час відображення бібліотеки у пам'ять використовують технологію копіювання під час записування, тому кожен процес матиме свою копію стека і даних бібліотеки.

З іншого боку, для коду бібліотечної функції будуть доступні такі ресурси, як дескриптори відкритих файлів процесу і стек потоку, шо викликав дану функцію. Не слід, однак, забувати про те, що під час роботи із даними потоку із коду бібліотеки потрібно виявляти обережність, зокрема, ніколи не вивільняти пам'ять, розподілену не в цій бібліотеці. Іншими словами, код бібліотек, розрахованих на використання у багатопотокових застосуваннях, має бути безпечним з погляду потоків (thread-safe).

**Особливості об'єктного коду динамічних бібліотек**

Код динамічних бібліотек зазвичай зберігають у виконуваних файлах формату, стандартного для цієї ОС (далі побачимо, що виконувані файли і файли DLL можуть відрізнятися тільки одним бітом у заголовку), але з погляду характеру цього коду є одна важлива відмінність між кодом DLL і кодом звичайних виконуваних файлів. Вона полягає в тому, що код DLL в один і той самий час повинен мати можливість завантажуватися за різними адресами. Для того щоб це було можливо, такий код потрібно робити позиційно-незалежним.

Позиційно-незалежний код завжди використовує відносну адресацію (базову адресу додають до зсуву). Базову адресу налаштовують у момент завантаження DLL в адресний простір процесу і називають також базовою адресою бібліотеки; такі адреси відрізняються для різних процесів. Зсув у цьому разі називають внутрішнім зсувом об'єкта.

## Точка входу динамічної бібліотеки і приклади її раціонального використання. (+)

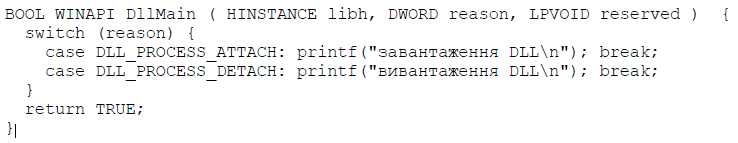
Одна із функцій динамічної бібліотеки може бути позначена як її точка входу. Така функція автоматично виконуватиметься завжди, коли цю DLL відображають в адресний простір процесу (явно або неявно); у неї можна поміщати код ініціалізації структур даних бібліотеки. Багато систем дають змогу задавати також і функцію, що буде викликана в разі вивантаження DLL із пам'яті.

**Точка входу в DLL**

Точку входу в DLL у Win32 АРІ описують як функцію:

BOOL WINAPI DllMain(HINSTANCE libh, DWORD reason, LPVOID reserved);

Першим параметром для неї є дескриптор екземпляра бібліотеки, другим – індикатор причини виклику (DLL\_PROCESS\_ATTACH – під час завантаження бібліотеки, DLL\_PROCESS\_DETACH – у разі її вивантаження), третій параметр не використовують.



## Структура виконуваних файлів для Windows. Формат РЕ. (+)

Файли формату РЕ (Portable Executable) з'явилися разом із Windows NT. Вони мають багато спільного із традиційними форматами UNIX-систем (об'єктні файли, на основі яких будують такий файл, можуть бути у форматі COFF – одному зі стандартних форматів об'єктних файлів у UNIX). Загальна структура файла в форматі РЕ наведена на рисунку.



Основні відмінності між двома форматами наведені нижче.

► Перед основним заголовком РЕ-файла поміщають так званий DOS-заголовок. Він містить невелику програму, яка у разі запуску під MS-DOS виводить повідомлення і завершується. Цей заголовок залишився відтоді, коли РЕ-файли часто намагалися запускати під керуванням MS-DOS або Windows 3.x. Під час звичайного запуску (під Windows) керування негайно передають за адресою початку виконання, заданою в основному заголовку.

► Інформацію, необхідну для виконання файла, не виносять в окрему структуру даних, а зберігають у стандартних структурах – основному заголовку, заголовках секцій тощо.

► Для зручності доступу адреси найважливіших об'єктів усередині файла (секцій експорту, імпорту, ресурсів тощо) утримують у заголовку файла як окремий масив DataDirectory.

► Окрема секція .rsrc зарезервована для зберігання ресурсів програми. Ресурси всередині цієї секції мають деревоподібну організацію з каталогами і підкаталогами.

## Схема процесу компонування для Windows у разі неявного зв'язування. (+)

► У Windows динамічне компонування здійснює система, а не окремий динамічний компонувальник; очевидно, що всередині виконуваного файла ім'я такого компонувальника не задають.

► Окрема секція .edata виділена для опису експортованих символів. Це досить важлива відмінність від ELF, для якого всі символи за замовчуванням є експортованими.

► Відмінності має і секція імпортованих функцій. Тут створена спеціальна таблиця, що визначає всі імпортовані функції; після запуску її заповнюють адресами функцій з DLL. Таку таблицю називають таблицею імпорту адрес (ІАТ). Використання ІАТ дає змогу звести всю інформацію про імпортовані адреси в одне місце у файлі.

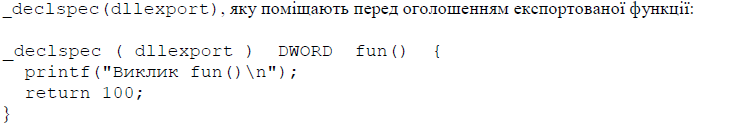
**Процес компонування у разі неявного зв'язування**

Для реалізації неявного зв'язування все, що потрібно від розробника застосування, – це вказати компонувальнику список необхідних DLL і впевнитися, що під час виконання всі вони можуть бути знайдені. Завантажувач ОС забезпечить пошук і виконання потрібного коду. Зауважимо, що якщо під час завантаження DLL виникла помилка, весь процес завантаження переривається. До початку виконання основного потоку процесу всі необхідні DLL мають бути відображені в його адресний простір.

У розробці DLL, на відміну від UNIX, основним завданням програміста є задання списку експортованих функцій. Цього можна домогтися такими способами:

► створити спеціальний файл із розширенням .DEF, у якому перелічити всі такі функції, і передати його компонувальнику;

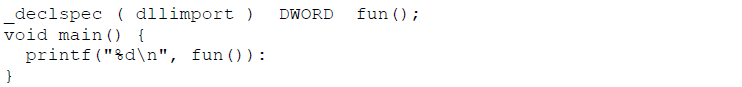
► у разі використання засобів компілятора C++ скористатися спеціальною конструкцією



При цьому додатково до створення DLL компонувальник згенерує спеціальний файл заглушок – статичну бібліотеку з розширенням .LIB, яку компонують з клієнтським застосуванням і яка містить код заглушок для створення зв'язків з DLL під час завантаження.

Ім'я цієї бібліотеки має бути явно задане як один із параметрів виклику компонувальника під час компонування клієнтського застосування.

У разі використання функцій з DLL їх, на відміну від UNIX, потрібно імпортувати. Це може мати такий вигляд:



Тоді під час компонування будуть узяті функції з .LIB-файла, а в разі виконання заглушки звертатимуться до справжніх функцій з DLL.

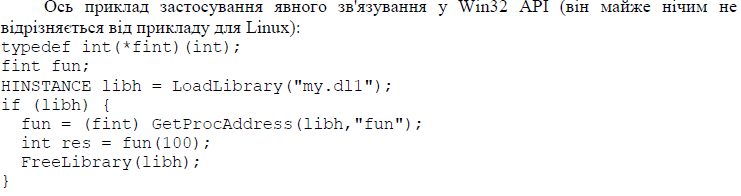
## Схема процесу компонування для Windows у разі явного зв'язування. (+)

Процес явного зв'язування DLL у Windows складається переважно з тих самих кроків, що і для Linux.

1. Для того щоб відобразити DLL в адресний простір процесу, використовують функцію LoadLibrary(libpath) з одним параметром, який задає шлях до файла бібліотеки. Ця функція повертає дескриптор екземпляра бібліотеки (значення типу HINSTANCE). Вона є аналогом dlopen().

2. Аналогом dlsym() для отримання покажчика на функцію за її іменем є функція GetProcAddress(libh,sym). Її параметри за змістом ті самі, що і для dlsym() – дескриптор екземпляра і рядок з іменем функції.

3. Для вивантаження бібліотеки з пам'яті використовують функцію FreeLibrary(libh).



## Механізми передавання параметрів до процедур і функцій: за значенням; за посиланням (за адресою); за поверненим значенням; за результатом; за іменем; відкладеним обчисленням. Загальні визначення. (+)

***Процедура*** – це блок коду програми, який має одну точку входу і одну точку виходу і який повертає керування до наступної команди після команди передачі керування процедурі. Розглядаємо процедурний підхід як найбільш популярний на даний час.

Процедури можуть отримувати або не отримувати параметри від викликаючої процедури і можуть повертати або не повертати результати. Процедури, які повертають що-небудь, раніше мали окрему назву «функції», проте зараз функціями називають будь-який окремий блок коду, незалежно від повернення результату, який повертає керування до місця виклику.

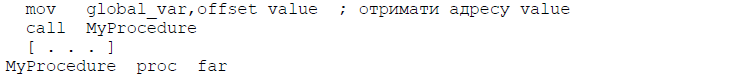
***Передавання параметрів за значенням***. Функції передають власне значення параметра. При цьому фактично копіюють значення параметра, а функція використовує його копію. Отже, модифікація вихідного параметра стає неможливою. Такий механіхм використовують для передавання невеликих параметрів, розміром до одного слова. Наприклад, якщо параметри передавати в регістрах:

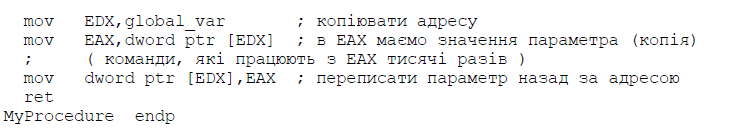


***Передавання параметрів за посиланням (за адресою).*** Функції передають не значення змінної, а її адресу, за якою функція сама прочитає значення пераметра. Цей механізм є зручний для передавання великих масивів даних, а також у випадках, коли функція має модифікувати параметри:



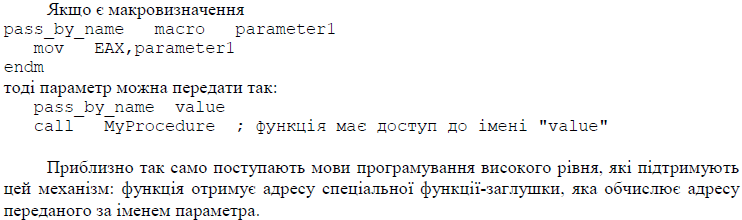
***Передавання параметрів за поверненим значенням.*** Цей механізм об’єднує передавання за значенням і за посиланням. Функції передають адресу змінної, а функція будує локальну копію параметра, після цього працює з нею, а в кінці записує локальну копію назад за переданою адресою. Цей метод ефективніший від звичайного передавання за посиланням в тих випадках, коли функція має дуже часте звертання до параметра, наприклад, при використанні параметра в глобальній величині:





***Передавання параметрів за результатом.*** Цей механізм відрізняється від попереднього лише тим, що при виклику функції значення параметра не отримують, функція самостійно будує результат, а передану адресу використовують лише для записування за нею результату.

***Передавання параметрів за іменем.*** Цей механізм використовують у макровизначеннях, в директиві EQU і в препроцесорі C в часі опрацювання команди #define. При реалізації цього механізму в компільованій мові програмування доводиться виконати заміну параметра за іменем іншими механізмами, зокрема, за допомогою макровизначень.



***Передавання параметрів відкладеним обчисленням***. Як і в попередньому випадку, викликана функція отримує адресу іншої функції, що обчислює значення параметра. Такий механізм є зручним, коли обчислення параметра вимагає багато ресурсів або часу, або коли не потрібно мати зразу цілий ряд обчислених значень, а достатньо отримувати від викликаної функції по одному результату.

Під час передачі параметра відкладеним обчисленням функція отримує адресу заглушки, яка при першому звертанні до неї обчислює значення параметра і зберігає його у внутрішній локальній пам’яті, а при наступних викликах повертає раніше обчислене значення.

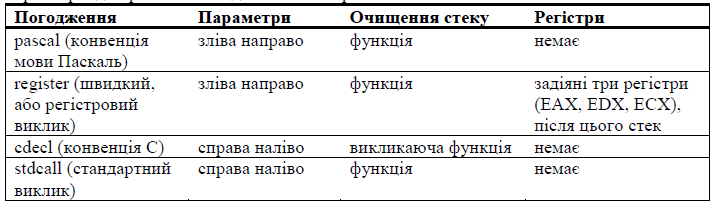
Такий механізм найчастіше використовують в системах штучного інтелекту і в операційних системах, а також в сучасній алгоритмічній мові Python. В мові Python реалізовані функції-генератори, які автоматично призупиняють і відновлюють своє виконання, зберігаючи при цьому інформацію, необхідну для генерування (відкладеного обчислення) значень.

## Способи передавання параметрів до процедур і функцій: в регістрах; в глобальних зміних; в стеку. Погодження (конвенції) для передавання параметрів у функцію через стек. (+)

***Передавання параметрів в регістрах.*** Якщо функція має невелику кількість параметрів та їх розміри не більші чотирьох байтів, то найкращим місцем є регістри загального призначення. Прикладами можуть бути майже всі виклики переривань DOS і BIOS. Мови високого рівня використовують регістр EAX для того, щоб повернути результат роботи функції, а також регістр ST(0) співпроцесора для повернення результату дійсного числа.

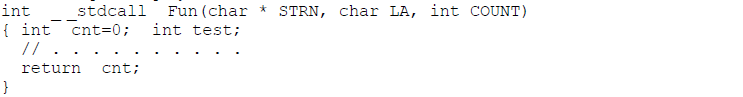
***Передавання параметрів в глобальних змінних.*** Якщо не вистарчає регістрів, один з способів обійти таке обмеження – записати параметр у величину, до якої пізніше буде звертатись функція, подібним механізмом, як було показано при передаванні параметрів за поверненим значенням. Проте такий метод не можна використати для рекурсивних функцій чи функцій з повторним викликом.

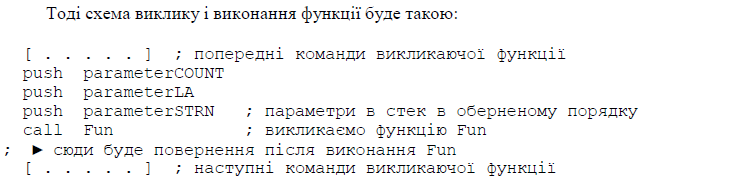
***Передавання параметрів в стеку***. Це є основний спосіб зв’язку між функціями, який придатний для функцій різної будови і різних операційних систем. Параметри записують в стек безпосередньо перед викликом функції. Такий спосіб використовують мови високого рівня компільованого типу. Для читання параметрів з стеку використовують не команду POP, а регістр EBP, основне призначення якого в архітектурі Intel якраз полягає в забезпеченні роботи з стеком як зі звичайною ділянкою пам’яті.

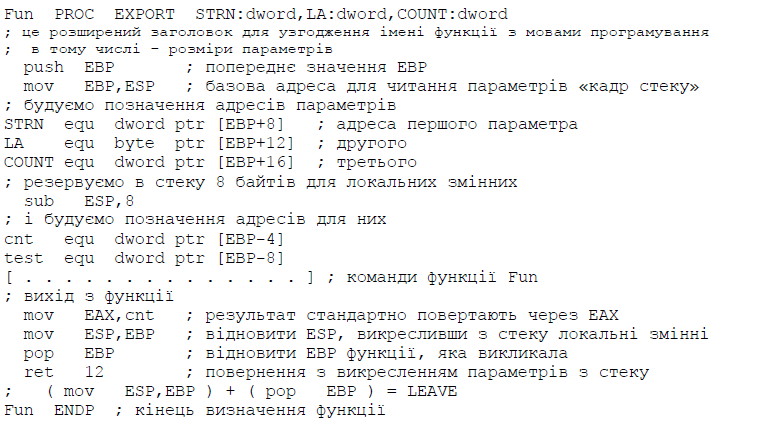


## 13. Модель конвенції stdcall передавання параметрів до процедур і функцій. (+)

Нехай маємо деяку функцію з трьома параметрами, двома локальними змінними і одним поверненим результатом:







Тепер розглянемо кроки виклику і виконання функції.

1. початковий стан стеку
2. в стек записали параметри push
3. викликали функцію call
4. налаштували «кадр стеку»
5. місце локальних змінних sub ESP,8
6. вихід з функції – відновили ESP і EBP
7. повернення з викресленням параметрів з стеку: ret 12 - стан 1.

## Стандартні типи даних Windows та їх еквіваленти в мові C. (+)

|  |  |  |
| --- | --- | --- |
| **Windows** | **означення** | **аналог C** |
| BOOL | логічний (4-байтовий) має два значення - 0 або 1. При використанні WINAPI прийнято вживати замість 0 специфікатор NULL | bool  (1-байтовий) |
| BYTE | байт, або восьмибітне беззнакове ціле число | unsigned char |
| DWORD, UINT | 32-бітове беззнакове ціле | unsigned long int |
| INT, LONG | 32-бітове ціле | long int |
| DOUBLE | 8-байтове дійсне число | double |
| NULL | нульовий вказівник (4-байтовий)  або ціле число 0; | void \* NULL=0;  int NULL=0; |
| HANDLE | дескриптор – ідентифікатор якого-небудь об'єкта (4-байтовий); для різних типів об'єктів існують різні дескриптори |  |
| HCURSOR | дескриптор курсора (4-байтовий) |  |
| HDC | дескриптор контексту пристрою |  |
| HINSTANCE | дескриптор екземпляра додатка (програми) |  |
| HWND | дескриптор вікна (4-байтовий) |  |
| LPINT | вказівник на ціле (4-байтовий) | int \* |
| LPSTR | вказівник на будь-який рядок, що закінчується нуль-кодом (4-байтовий) | char \*  (4-байтовий) |
| LPTSTR | вказівник на рядок без юнікоду; це надбудова функції LPSTR | char \* |
| LPWSTR | вказівник на UNICODE рядок; це надбудова функції LPSTR | wchar\_t \*  (4-байтовий) |
| CHAR, TCHAR | символьний тип (1-байтовий) | char |
| WCHAR | символьний тип (2-байтовий) | wchar\_t  (2-байтовий) |
| LPARAM | довгий параметр (4-байтовий); використо-вують разом з WPARAM в деяких функціях |  |
| WPARAM | параметр-слово (4 байти); використовують разом з LPARAM в деяких функціях |  |
| LRESULT | значення, яке повертає віконна процедура  (4-байтове) | long |

## Процедура підготовки функцій для DLL на прикладі алгоритмічної мови (C++ чи іншої). (+)

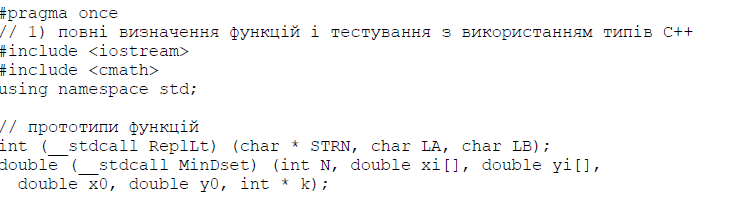
Першим кроком треба виконати проектування, програмування і тестування функцій, які будуть поміщені в DLL. Для цього можна створити проект типу “Empty Project”, який не зв’язаний з конкретними типами проектів.

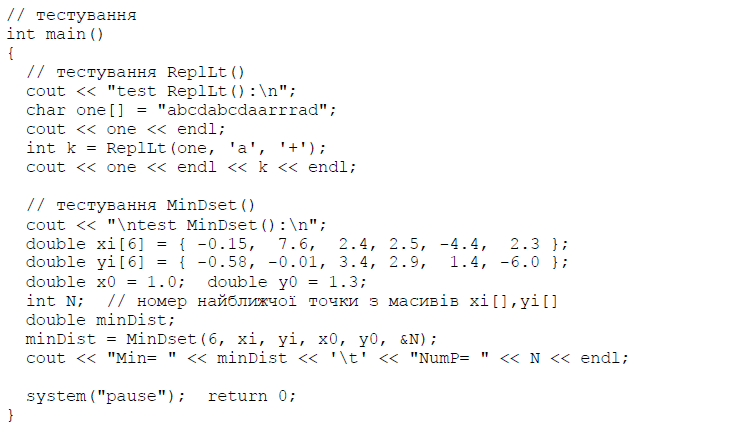
*Приклад 1.* Функція виконує заміну в рядку літер STRN кожної букви LA на LB. Крім того, обчислює кількість виконаних замін.

*Приклад 2.* Задано координати масиву точок (*x*i,*y*i), i=1,...,*N*, на площині. Знайти точку масиву, яка найближча до заданої окремої точки (*x0*,*y0*), а саме: обчислити мінімальну відстань *min* від (*x0*,*y0*) до знайденої точки масиву та номер *k* цієї точки в масиві.

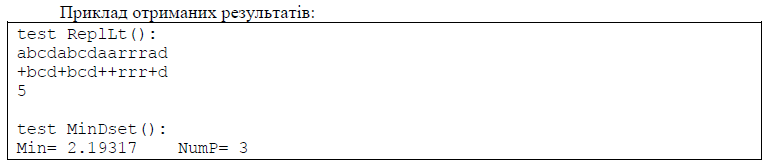
*Крок 1. Тестування з використанням типів C++.*

Виконуємо розробку і тестування в межах одного середовища програмування C++. В цьому разі достатньо використати стандартні типи даних мови C++, бо виклики функцій відбуваються так само в межах мови C++, і проблеми узгодження типів параметрів не виникає.





…



*Крок 2. Тестування з використанням типів Windows.*

Як було зазначено раніше, функції, поміщені в DLL, в загальному випадку розраховані на використання в різномовних середовищах програмування. Тому варто організувати повторне тестування тих самих функцій, використавши для параметрів стандартні типи Windows. При цьому треба одночасно замінити типи деяких локальних змінних функцій.

Прототипи функцій за кроком 2 зазвичай подають як інструкцію до використання функцій нашої бібліотеки. Тоді користувач має узгодити типи даних іншої А-мови програмування за схемою:

типи Windows ↔ типи А-мови

Кроку 2 можна не виконувати, тоді інструкцією до бібліотеки будуть прототипи функцій мовою C++, і узгодження відбувається за схемою:

типи C++ ↔ типи А-мови

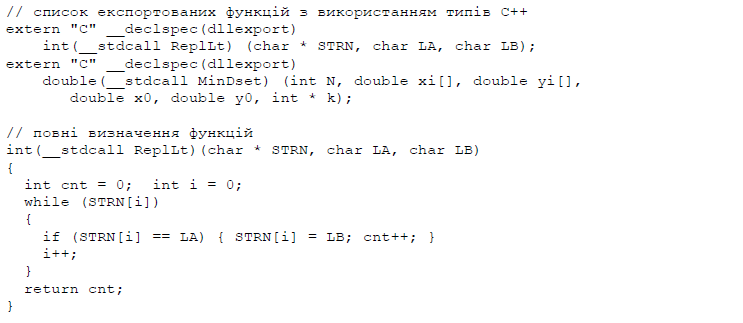
## Компіляція функцій в бібліотеку DLL (створення DLL). Налаштування компілятора і завантажувача. (+)

Побудовані попереднім кроком функції тепер використаємо для компіляції в бібліотеку DLL. Процедура створення DLL регламентована в кожній системі програмування відповідною інструкцією, отже треба її отримати і виконати.

Після запуску Visual Studio:

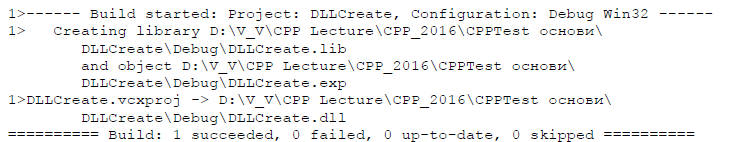
New Project → Windows Desktop → Dynamic Link Library (DLL) → визначити Name і Location → Create directory for solution → OK

У вікні редактора коду файла DLLCreate.cpp друкуємо (копіюємо з файла тестування) тексти наших функцій, визначаючи додатково директиви експорту функцій. При цьому вирішуємо, чи додати в бібліотеку функцію входу DllMain. Ця функція має шаблон визначення в зв’язаному файлі dllmain.cpp, який редагуємо за потреби (відкриваємо через Solution Explorer).



Компілюємо DLL: Build → Build Solution.

Якщо все зроблено правильно, у вікні Output має бути приблизно таке:



В результаті в папці Debug цілого проекту мають бути файли DLLCreate.dll, DLLCreate.lib та інші. Названі два файли будуть потрібні для використання в інших програмах користувачів, тому їх варто копіювати в окрему від проекту папку, до якої будуть мати доступ прикладні програми користувачів. Необхідно, щоб ці два файли залишались в межах однієї папки.

## Використання DLL в прикладних програмах методом явного зв’язування (на прикладі мови C чи іншої). (+)

Альтернативним для неявного є явне зв'язування, коли динамічну бібліотеку завантажують в адресний простір процесу виконанням системного виклику з його коду. Після цього, використовуючи інший системний виклик, застосування отримує адресу необхідної йому функції бібліотеки і може її викликати. Після використання бібліотеку можна вилучити з пам'яті. Компонувальник при цьому нічого про неї не знає, завантажувач ОС автоматично бібліотек не завантажує, отже компіляція і запуск застосування відбуваються без клопотів щодо пошуку і підключення бібліотеки.

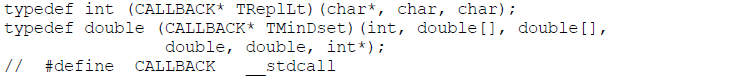
Зауважимо, що розглянуте вище неявне зв'язування здебільшого зводиться до автоматичного виконання тих самих викликів, які сам програміст виконує за явного.

Для реалізації явного зв’язування треба виконати такі кроки.

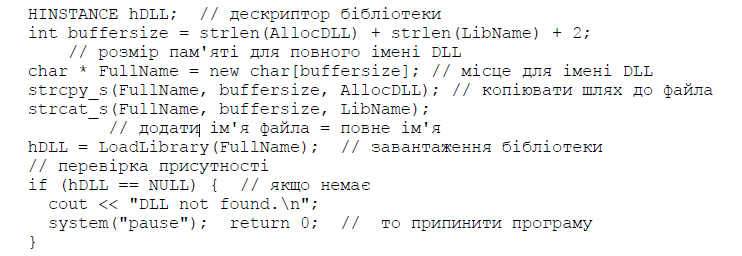
*Крок 1*. Визначити місце розташування та ім'я бібліотеки DLL:



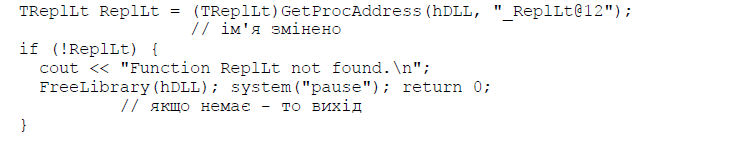
*Крок 2.* Оголошення типів вказівників на зовнішні функції DLL:



*Крок 3.* Завантажити і перевірити наявність бібліотеки:



*Крок 4.* Побудувати вказівники на функції і перевірити наявність функцій в бібліотеці:



*Крок 5.* В процесі виконання програми викликати функції як звичайно:



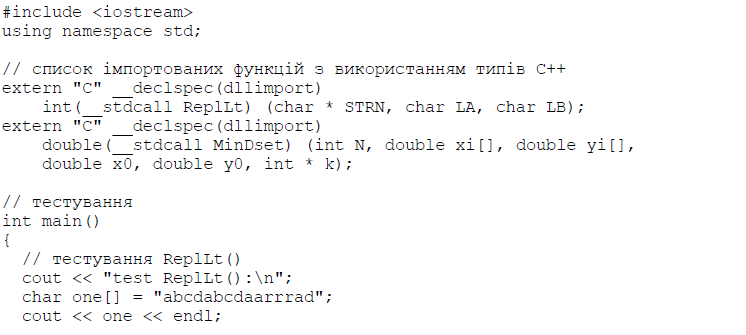
*Крок 6.* В кінці виконання програми звільнити бібліотеку:

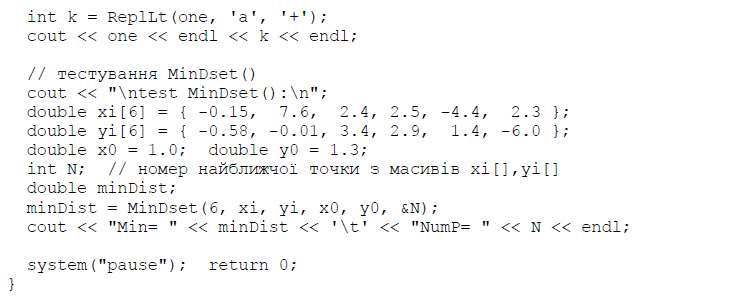


## Використання DLL в прикладних програмах методом неявного зв’язування (на прикладі мови C чи іншої). (+)

Неявне зв’язування з DLL виконують на етапі компіляції і будови прикладної програми користувача. Середовище програмування має самостійно виконати пошук бібліотеки і зв’язати потрібні функції DLL з прикладною програмою. В результаті компонування будуть автоматично додані засоби операційної системи для завантаження DLL під час виконання прикладної програми і виклику функцій з DLL.

*Крок 1.* Проектування прикладної програми. Створюємо новий проект “Empty Project”, додаємо до нього файл \*.cpp, і записуємо текст:





В прототипах функцій визначаємо їх параметром dllimport, тобто такими, які є в зовнішній бібліотеці. Для правильної компіляції нашої програми прототипів достатньо. Щоб переконатись, виконаємо команду Build→Compile, і побачимо повідомлення про успішну компіляцію.

*Крок 2.* Налаштування зв’язку з файлом імпорту \*.lib.

Якщо спробувати виконати компонування Build→Build Solution, то побачимо повідомлення завантажувача (компонувальника) про нерозв’язані зовнішні посилання на функції. Компонувальнику потрібний файл імпорту \*.lib.

Отже треба виконати налаштування для пошуку файла DLLCreate.lib.

Для цього виконуємо такі дії (для режиму Debug):

2.1. Відкрити вікно Project→Properties→Property Pages.

2.2. На лівій панелі вибрати Configuration Properties→Linker→Input.

2.3. На панелі властивостей вибрати Additional Dependencies і кнопкою вибрати <Edit...>.

2.4. У вікні Additional Dependencies у верхній частині друкуємо назву файла імпорту: DLLCreate.lib.

2.5. Натиснути OK, щоб повернутись до сторінки властивостей.

2.6. На лівій панелі вибираємо Configuration Properties→Linker→General.

2.7. На панелі властивостей вибрати Additional Library Directories, відкрити кнопкою вікно і записати шлях до розташування файла імпорту: d:\MyDLLfun\

2.8. Кнопками OK закриваємо вікна. Тепер прикладну програму можна успішно компілювати і компонувати, але не виконувати.

Зазначені дії стосуються системи Visual Studio. В інших системах програмування можуть бути потрібні інакші способи налаштування. В будь-якому разі необхідні дві речі: назва файла імпорту і шлях до розташування файла імпорту.

*Крок 3*. Налаштування зв’язку з файлом бібліотеки \*.dll.

Під час виконання програми операційна система намагатиметься шукати саму бібліотеку DLLCreate.dll. За правилами пошук виконують в такому порядку:

пошук в каталозі, звідки запущена програма;

пошук в поточному каталозі;

пошук в системному каталозі (GetSystemDirectory);

пошук в каталозі Windows (GetWindowsDirectory);

пошук в каталогах, визначених в середовищі (PATH).

Якщо пошук виявився невдалим, програма не буде виконана.

Найпростіший спосіб уникнути цієї проблеми – копіювати DLL в каталог, що містить виконуваний EXE-файл прикладної програми. Для середовища Visual Studio копіювати DLL треба на початку в основну папку проекту – туди, де файли текстів програми.

Інші варіанти – копіювати DDL в одну з системних папок Windows, або за вказівкою змінної PATH.

*Крок 4.* Будова остаточного варіанта прикладної програми.

Якщо прикладна програма вважається остаточно налагодженою, будують фінальний випуск програми Release.

4.1. Project→Properties→ Property Pages→Configuration→Release.

4.2. Повторити налаштування 2.2-2.8 для режиму Release.

4.3. Після повернення до головного вікна наа панелі інструментів обираємо Solution Configuration→Release.

4.4. Повторно компілюємо Build→Compile і компонуємо програму Build→Build Solution.

4.5. Має з’явитись папка Release з EXE-файлом та іншими. Цю папку зазвичай передають користувачам нашої програми.

4.6. Програму можна виконати в середовищі програмування, командою Local Windows Debugger (кнопка ►). В цьому разі DDL підключається автоматично самою системою. Якщо ж спробувати виконати EXE-програму зовнішнім запуском з папки Release, то отримаємо повідомлення про відсутність бібліотеки DLLCreate.dll.

*Крок 5.* Копіювання файла бібліотеки DLLCreate.dll.

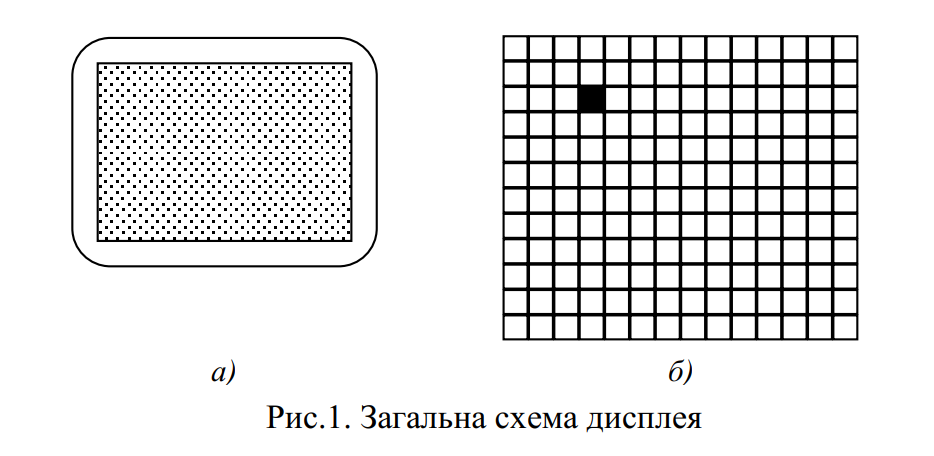
Копіюємо будь-яким способом файл DLLCreate.dll в папку Debug чи Release, залежно від остаточного варіанта програми.

Запускаємо програму на виконання зовні. Виконуємо за потреби додаткове тестування.

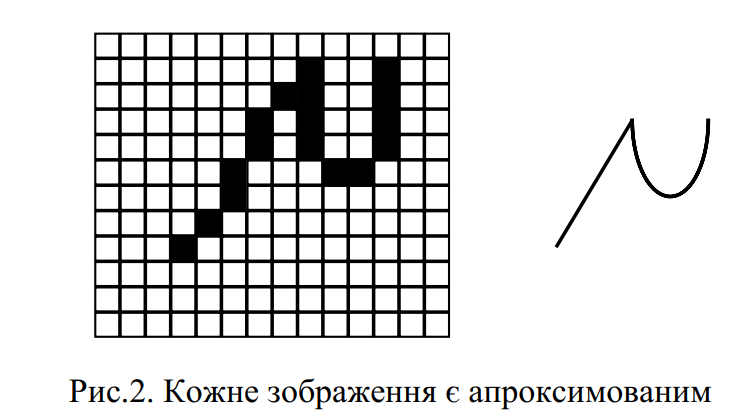
# **Відповіді до питань до розділу "Графічні редактори"**

## Піксел. Роздільна здатність екрана. Палітра кольорів. Принцип малювання на екрані.+

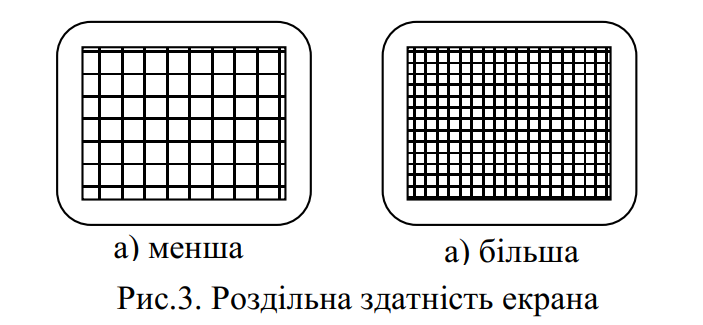
Активна видима ділянка екрана складається з прямокутних пікселів – елементарних одиниць, зафарбованих у будь-який колір (рис.1,б). Зауважимо, що піксели певних типів дисплеїв можуть не бути точними квадратами, однак переважно все ж таки є квадратними.



Малювання на екрані полягає у зафарбовуванні певної групи пікселів. Оскільки вони квадратної форми, то зображення завжди є апроксимацією (наближенням) реальної картинки (рис.2).



Екран можна перемикати на різні режими роботи. Режими відрізняються роздільною здатністю – кількістю пікселів на одиницю довжини (переважно на дюйм: DPI – dots per inch). Чим більша роздільна здатність, тим кращу якість картинки можна отримати (рис.3).

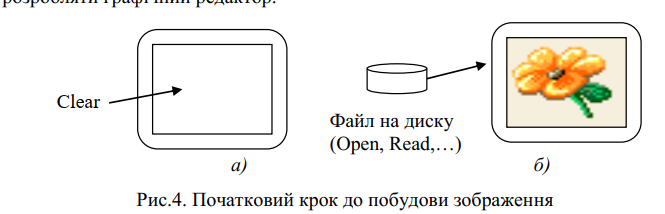


Переважно користуються стандартними розмірами (роздільними здатностями) екрана, наприклад: 640 x 480, 800 x 600, 1024 x 768, 1152 x 864, 1280 x 1024 тощо.

Кількість різних кольорів, у які можна зафарбовувати окремий піксел, теж залежить від режиму роботи, а також від апаратних можливостей монітора, зокрема, від розміру відео пам’яті Сучасні комп’ютери зазвичай використовують палітру кольорів, яка може мати 256 x 256 x 256 різних значень, однак використовують також палітру з 256-ти кольорів і навіть з 16-ти кольорів.

## Графічні примітиви. Графічні бібліотеки. Принципи використання.+

Початково екран можна очистити (рис.4,а) або завантажити з файла раніше створену картинку (рис.4,б). Малювання виконують за допомогою графічних примітивів – операцій, таких, як Ellipse(x1,y1,x2,y2), LineTo(x,y), Rectangle(x1,y1,x2,y2), FloodFill(x,y,Color,FillStyle), TextOut(x,y,Text), Draw(x,y,Graphic) тощо. Сукупність таких операцій реалізують як окремі модулі чи бібліотеки, що належать до складу системи програмування (System.Drawing C#; Open CV, Pillow Python; Open CV, Qt C++). Якщо примітиви відсутні, то доцільно спершу створити їхню бібліотеку, а потім розробляти графічний редактор.

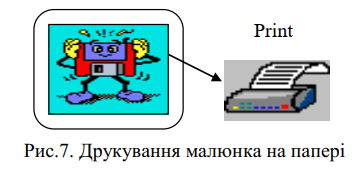


Малюнок можна будувати з окремих частин інших малюнків, як зображено на рис.5. З цією метою необхідно передбачити відповідні графічні операції, наприклад: виокремити частину малюнка; зберегти блок (виокремлену частину) у файлі; вставити блок з файла у задану позицію екрана тощо.



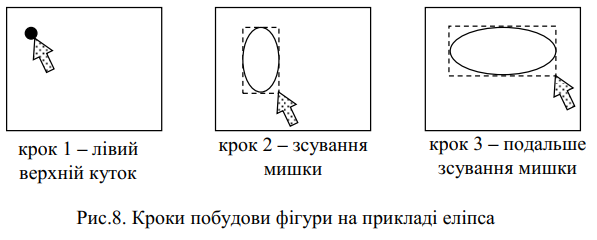
Необхідно забезпечити збереження малюнка у файлі (рис.6). Додаткова бажана можливість – збереження у файлі в різних форматах, а також сумісність формату малюнка з аналогічними форматами інших програм. Доцільно мати можливість друкувати малюнок або його фрагмент на папері чи іншому фізичному носії – слайд тощо (рис.7).





## Графічний інтерфейс редактора з користувачем. Клавіатурний інтерфейс редактора з користувачем. +

Графічний інтерфейс. Частина екрана пристосована під панель інструментів –зображення графічних операцій (примітивів), якими можна користуватися. Команди редактора (інструменти) можна вибирати як мишкою, так і комбінацією клавіш зі стрілками і, можливо, інших. Порядок виконання однієї операції зазвичай є таким: вибір (мишею) виду операції → фіксація місця початку фігури (верхнього лівого кутка) → фіксація місця закінчення фігури (правого нижнього кутка). При фіксуванні не обов’язково вибирати кутки як зазначено – достатньо обрати два протилежні кутки у довільному порядку. При малюванні декількох однотипних фігур повторний вибір зазвичай непотрібний. До моменту закінчення малювання фігури бажано мати змогу бачити прообраз майбутньої фігури. Наприклад, для еліпса послідовність кроків може бути такою, як на рис.8.

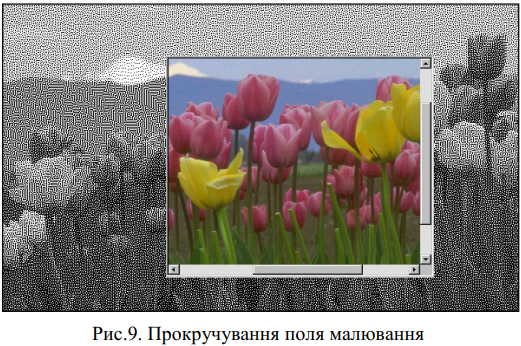


Остаточно фігура фіксується або при відпусканні клавіші миші, або при повторному її натисненні – залежно від реалізації.

*Клавіатурний інтерфейс*. Графічні операції обирають натисканням певних клавіш, наприклад: Е → Ellipse, C → Circle, R → Rectangle і т.д. У цьому випадку відображення панелі інструментів на екрані не є обов’язковим, але необхідно мати в інформаційній частині повідомлення про вибрану команду. Фіксування місця та розмірів фігури можна виконати клавішами зі стрілками, або мишкою – так само, як при графічному інтерфейсі.

## Прокручування поля, багатовіконність редактора, коректне припинення роботи.+

*Прокручування поля малювання*. Малюнок може бути довільних розмірів, зокрема, більшим, ніж поле малювання. У цьому випадку необхідне горизонтальне та вертикальне прокручування поля малювання за допомогою смуг перегляду. В кожен момент на полі малювання, як у вікні, відображається частина малюнка як на рис.9.



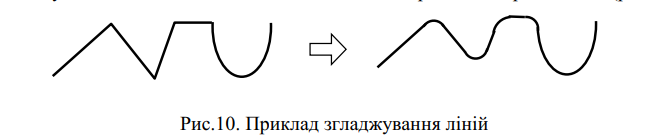
Весь малюнок зберігається як єдине ціле в пам’яті. Варто подумати про максимально припустимі розміри малюнка.

Багатовіконність. Це є можливість одночасної роботи з декількома малюнками – кожен у своєму вікні. Переключення між вікнами реалізується через меню або комбінацією клавіш, наприклад, Ctrl+F6. Для ефективної роботи необхідно мати команди (функції) для копіювання фрагментів і цілих малюнків у внутрішній буфер графічного редактора та вставляння з буфера. Графічні редактори можуть використовувати одночасно свій власний внутрішній буфер та буфер Windows, при цьому копіювання у різні буфери необхідно здійснювати різними командами, і так само вставляння.

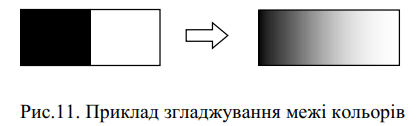
Закінчення роботи. Перевіряємо збереження малюнків у файлі, виконуємо очищення буферів пам’яті, звільняємо надану під час роботи пам’ять, закриваємо файли, викреслюємо тимчасові файли і т.п.

## Сервісні можливості редактора: логічні групи команд; розтягання, стиснення, повороти; масштабування малюнка; контекстні і випливаючі меню; підказки про події на екрані; довідкова і навчальна підсистема редактора.+

* 1. Розбиття множини усіх припустимих для редактора команд на окремі логічні групи. Перемикання між групами.
  2. Збільшення малюнка (розтягання), зменшення (стиснення), а також повороти на кут *±n(π/2)*, n=1,2,3.
  3. Масштабування малюнка з метою точного виконання графічних операцій.
  4. Реалізація контекстного (наприклад, за клавішею F1) і випливаючого меню (наприклад, за правою клавішею мишки). Вибір та виконання команд через меню.
  5. Підказки про події на екрані.
  6. Згладжування ліній та інші подібні методи геометричного опрацювання (рис.10).



* 1. Згладжування межі зміни кольору (плавний перехід одного кольору в інший) та інші способи фільтрування малюнка (рис.11).



* 1. Формування у файлах набору демонстраційних малюнків.
  2. Наявність використання. навчальної підсистеми – опис редактора, команд, особливостей

## Графічний програмний інструментарій: перо, пензель, шрифт. Загальні характеристики. +

Клас Pen(перо)

З допомогою класу Pen створюється об’єкт-перо, який використовують для малювання ліній. Подаємо огляд потенційних параметрів класу. В різних системах програмування можуть бути реалізовані такі самі параметри або подібні за змістом. Остаточна реалізація залежить від можливостей бібліотеки графічних функцій.

Властивості класу є такі:

|  |  |
| --- | --- |
| property Color | Колір ліній, які креслить перо |
| property Width | Товщина ліній у пікселах екрана |
| property Style | Визначає стиль ліній (див. нижче). Враховується тільки для ліній товщиною в 1 піксел. Для товстих ліній завжди psSolid (суцільна) |
| property Mode | Визначає спосіб взаємодії ліній з тлом (див. нижче) |
| property Handle | Дескриптор пера. Використовується при безпосередньому звертанні до функцій API |

Тип стилю лінії Styleвизначається так:

enum TPenStyle = {

psSolid,

psDash,

psDot,

psDashDot,

psDashDotDot,

psClear,

psInsideFrame

};

Тип взаємодії лінії з тлом визначається так:

enum TPenMode = {

pmBlack,

pmWhite,

pmNop,

pmNot,

pmCopy,

pmNotCopy,

pmMergePenNot,

pmMaskPenNot,

pmMergeNotPen,

pmMaskNotPen,

pmMerge,

pmNotMerge,

pmMask,

pmNotMask,

pmXor,

pmNotXor

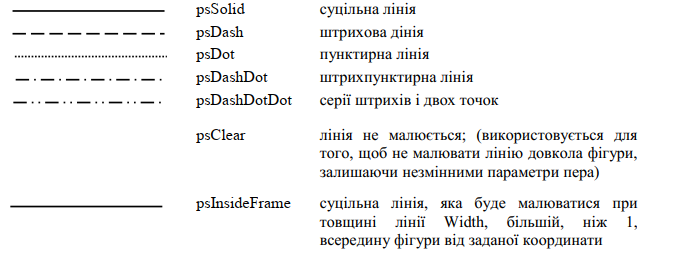
};

Властивість **Mode** може набувати одного з таких значень:

|  |  |
| --- | --- |
| pmBlack | Лінії завжди чорні. Властивості Color і Style ігноруються |
| pmWhite | Лінії завжди білі. Властивості Color і Style ігноруються |
| pmNop | Колір тла не змінюється (лінії невидимі) |
| pmNot | Інверсія кольору тла. Властивості Color і Style ігноруються |
| pmCopy | Колір ліній визначається властивістю Color пера |
| pmNotCopy | Інверсія кольору пера. Властивість Style ігнорується |
| pmMergePenNot | Комбінація кольору пера та інверсного кольору тла |
| pmMaskPenNot | Комбінація спільних кольорів для пера та інверсного кольору тла. Властивість Style ігнорується |
| pmMergeNotPen | Комбінація кольору тла та інверсного кольору пера |
| pmMaskNotPen | Комбінація спільних кольорів для інверсного кольору пера і кольору тла. Властивість Style ігнорується |
| pmMerge | Комбінація кольорів пера і тла |
| pmNotMerge | Інверсія комбінації кольорів пера і тла (інверсія pmMerge) |
| pmMask | Спільні кольори пера і тла |
| pmNotMask | Інверсія спільних кольорів пера і тла (інверсія pmMask) |
| pmXor | Об’єднання кольорів пера і тла операцією XOR |
| pmNotXor | Інверсія об’єднання кольорів пера і тла операцією XOR (інверсія pmXor) |

Зауважимо, що ми записали потенційні способи взаємодії з тлом. Залежно від системи програмування і наявної графічної бібліотеки можуть бути доступні лише окремі способи взаємодії.

Властивість **Style** може приймати одне із таких значень:



Клас Brush(пензель)

Об’єкти класу Brush (пензлі) використовують для заповнення внутрішнього простору замкнених фігур. Властивості класу можуть бути такими: Тип стилю пензля визначається так:

enum TBrushStyle = {

bsSolid,

bsClear,

bsHorizontal,

bsVertical,

bsFDiagonal,

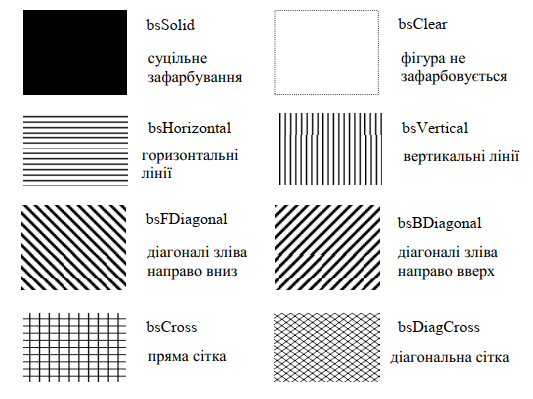
bsBDiagonal,

bsCross,

bsDiagCross

};

При цьому зафарбування буде таким:



Клас Font

<https://docs.microsoft.com/uk-ua/dotnet/api/system.drawing.font?view=net-5.0>

public bool Italic { get; } – курсив

public bool Underline { get; } – підкреслення

public bool Bold { get; } – жирний

public bool Strikeout { get; } – перекреслений

public System.Drawing.FontStyle Style { get; }

public enum FontStyle = {

Bold = 1

Italic = 2,

Regular = 3,

Strikeout = 4,

Underline = 5,

}

public float Size { get; } – The em-size of this [Font](https://docs.microsoft.com/uk-ua/dotnet/api/system.drawing.font?view=net-5.0).

public float SizeInPoints { get; } – The em-size, in points, of this [Font](https://docs.microsoft.com/uk-ua/dotnet/api/system.drawing.font?view=net-5.0).

public System.Drawing.FontFamily FontFamily { get; }

<https://docs.microsoft.com/uk-ua/dotnet/api/system.drawing.fontfamily?view=net-5.0>

## Операції читання/запису для графічних зображень. Формати збереження у файлах.+

*JPEG / JFIF*

JPEG (Joint Photographic Experts Group) - метод стиснення втрат. Стислі зображення JPEG зазвичай зберігаються у форматі файлів JFIF (JPEG File Interchange Format). Розширення назви файлів JPEG / JFIF - JPG або JPEG. Майже кожен цифровий фотоапарат може зберігати зображення у форматі JPEG / JFIF, який підтримує **восьмибітні зображення сірого кольору та 24-бітні кольорові зображення** (вісім біт для червоного, зеленого та синього). JPEG застосовує стиснення втрат до зображень, що може призвести до значного зменшення розміру файлу. Програми можуть визначати ступінь стиснення, що застосовується, а кількість стиснення впливає на візуальну якість результату. Якщо не надто велике, стиснення непомітно впливає на якість зображення чи не погіршує його, але JPEG- файли зазнають деградації поколінь при повторному редагуванні та збереженні. (JPEG також забезпечує зберігання зображень без втрат, але версія без втрат не підтримується широко.)

*PNG*

Формат файлу PNG (Portable Network Graphics) був створений як безкоштовна

альтернатива GIF з відкритим кодом. Формат файлу PNG підтримує восьмибітні палітри зображень (з додатковою прозорістю для всіх кольорів палітри) та 24-бітну truecolor (16 мільйонів кольорів) або 48-бітну truecolor з альфа-каналом і без нього - в той час як GIF підтримує лише 256 кольорів і один прозорий колір.

Порівняно з JPEG, PNG є кращим, якщо зображення має великі, рівномірні кольори. Навіть для фотографій - де JPEG часто є вибором для остаточного розповсюдження, оскільки техніка стиснення зазвичай дає менші розміри файлів - PNG все ще добре підходить для зберігання зображень під час процесу редагування через стиснення без втрат.

PNG забезпечує безпатентну заміну GIF (хоча GIF сам по собі зараз не є патентом), а також може замінити багато поширених видів використання TIFF. Підтримуються зображення з індексованим кольором, відтінками сірого та справжнім кольором, плюс додатковий альфа-канал. Adam7 переплетення дозволяє ранній попередній перегляд, навіть коли був переданий лише невеликий відсоток даних зображення. PNG може зберігати дані гамми та кольоровості для поліпшення відповідності кольорів на неоднорідних платформах.

PNG розроблений для того, щоб добре працювати в онлайнових програмах перегляду, таких як веб-браузери, і може бути повністю потоково переданий за допомогою прогресивної опції відображення. PNG надійний, забезпечує як повну перевірку цілісності файлів, так і просте виявлення поширених помилок передачі.

*GIF*

GIF (формат графічного обміну) в звичайному використанні обмежується 8-бітною палітрою, або 256 кольорів ( в той час як 24-бітна глибина кольору технічно можлива). GIF найбільше придатний для зберігання графіки з кількома кольорами, наприклад простими діаграмами, формами, логотипами та зображеннями в стилі мультфільму, оскільки використовує стиснення без втрат LZW, що є більш ефективним, коли великі площі мають один колір, і менш ефективні для фотографічних або пофарбованих зображень. Завдяки простоті та віку GIF досяг майже універсальної програмної підтримки. Завдяки анімаційним можливостям він все ще широко використовується для надання ефектів анімації зображень, незважаючи на низький коефіцієнт стиснення порівняно з сучасними відеоформатами.

*BMP*

Формат файлу BMP (растрова карта Windows) обробляє графічні файли в операційній системі Microsoft Windows. Зазвичай файли BMP не стискаються, а значить, великі та без втрат; їх перевагою є їх проста структура та широке сприйняття в програмах Windows.

*TIFF*

Формат TIFF (Tagged Image File Format) - гнучкий формат, який зазвичай зберігає вісім біт або шістнадцять біт на колір (червоний, зелений, синій) для 24-бітних і 48-бітових зображень відповідно, зазвичай використовуючи розширення імені файлу TIFF, або TIF. Ця структура була розроблена так, щоб вона легко розширювалася, і багато виробників ввели власні власні теги спеціального призначення - в результаті чого ніхто не читає будь-який варіант файлу TIFF. TIFF можуть бути втратними або без втрат, залежно від метода, обраного для зберігання пікселів зображення. Дехто пропонує порівняно гарну компресію без втрат для дворівневих (чорно-білих) зображень. Деякі цифрові камери можуть зберігати зображення у форматі TIFF, використовуючи алгоритм стиснення LZW для зберігання без втрат. Формат зображення TIFF не підтримується широко веб-браузерами. TIFF залишається широко прийнятим як стандарт фотофайлів у поліграфічній справі. TIFF може обробляти кольорові простори, характерні для пристрою, такі як CMYK, визначений певним набором фарб друкарського друку. Програмні пакети OCR (Optical Character Recognition) зазвичай генерують певну форму зображення TIFF (часто однотонне ) для сканованих текстових сторінок.

*SVG*

SVG (масштабована векторна графіка) - це відкритий стандарт, створений та

розроблений Всесвітнім консорціумом веб-сторінок для вирішення потреб (та спроб декількох корпорацій) в універсальному, доступному для написання та універсальному векторному форматі для Інтернету та іншим способом. Формат SVG не має власної схеми стиснення, але через текстовий характер XML графіку SVG можна стиснути за допомогою програми, такої як gzip. Через свій сценарій сценаріїв, SVG є ключовим компонентом у веб-додатках: інтерактивні веб-сторінки, які виглядають і діють як програми.

*CGM*

CGM (Computer Graphics Metafile) - це формат файлів для двовимірної векторної графіки, растрової графіки та тексту, визначається ISO/IEC 8632. Усі графічні елементи можна вказати у текстовому вихідному файлі, який можна скласти у двійковий файл або в одне з двох текстових подань. CGM забезпечує засіб обміну графічними даними для комп’ютерного представлення 2D графічної інформації, незалежної від будь-якого конкретного додатку, системи, платформи чи пристрою. Вона певною мірою була прийнята в галузі технічної ілюстрації та професійного дизайну, але значною мірою його витіснили формати, такі як SVG та DXF .

*ODF*

Формат Open Document для додатків Office, (ODF), також відомий як OpenDocument , є ZIP стиснутим XML-форматом файла для електронних таблиць, діаграм, презентацій і обробки текстів документів. Він був розроблений з метою надання відкритої специфікації формату файлів на основі XML для офісних програм. Стандарт був розроблений технічним комітетом в консорціумі Організації просування структурованих інформаційних стандартів (OASIS). Він базувався на специфікації Sun Microsystems для OpenOffice.org XML, форматі за замовчуванням для OpenOffice.org та LibreOffice. Спочатку було розроблений для StarOffice, "щоб забезпечити відкритий стандарт для офісних документів". Крім того , що стандарт OASIS, він був опублікований як ISO/IEC міжнародного стандарту ISO/IEC 26300 - Формат Open Document для додатків Office (OpenDocument).

*EPS*

Інкапсульований PostScript (EPS) - це конвенція про структурування документів - відповідність (DSC) формату документа PostScript, яку можна використовувати у форматі графічного файлу. Файли EPS - це більш-менш автономні, досить передбачувані документи PostScript, що описують зображення або малюнок, і їх можна помістити в інший документ PostScript. Файл EPS - це по суті програма PostScript, яка зберігається як єдиний файл, який включає попередній перегляд із низькою роздільною здатністю, "інкапсульований" всередині документа, що дозволяє деяким програмам відображати попередній перегляд на екрані.

Файл EPS містить коментар DSC BoundingBox, що описує прямокутник, який містить зображення, описане у файлі EPS. Програми можуть використовувати цю інформацію для викладення сторінки, навіть якщо вони не в змозі безпосередньо відобразити PostScript всередині.

*XML*

Як можна використовувати графіку в XML? Зробити посилання на HTML або використати XLink. Або вбудувати SVG.

Графіка традиційно є лише посиланнями, які, як правило, мають файл зображення в результаті опрацювання, а не інший фрагмент тексту. Тому вони можуть бути реалізовані будь-яким способом, який підтримується специфікаціями XLink та XPointer, включаючи використання аналогічного синтаксису для існуючих зображень HTML. На них також можна посилатися, використовуючи вбудований механізм NOTATION та ENTITY XML аналогічно стандартному SGML, як і зовнішні нерозділені об'єкти.

Однак масштабована векторна графіка (специфікація XML для векторної графіки) дозволяє використовувати розмітку XML для малювання об'єктів векторної графіки безпосередньо у вашому XML-файлі. Це забезпечує величезну потужність для включення портативної графіки, особливо інтерактивної або анімованої послідовності, і тепер вона стає підтримуваною в браузерах, і їх можна експортувати зі стандартних графічних програм (малюнків), таких як GIMP.

Характеристики посилань на XML для зовнішніх зображень дають набагато кращий контроль над переходом та активацією посилань, тому автор може, наприклад, показувати зображення під час завантаження сторінки чи вибору мишкою в окремому вікні, не вдаючись до сценаріїв. Сам XML не передбачає або обмежує формати графічних файлів: GIF, JPG, TIFF, PNG, CGM, EPS та SVG. Проте векторні формати (EPS, SVG), як правило, необхідні для нефотографічних зображень (діаграм).

*Вбудована двійкова графіка*

Не можна вбудовувати необроблений бінарний графічний файл (або будь-який інший бінарний [нетекстовий]) безпосередньо у XML-файл, оскільки будь-який байт, що нагадує розмітку, буде неправильно інтерпретований: потрібно зробити на нього посилання. Однак можна включити текстово-кодовану трансформацію бінарного файлу як розділ, позначений CDATA, використовуючи щось на зразок UUencode з символами розмітки ], та >, і викреслити з карти, щоб вони не були подані як помилка CDATA і припинення послідовності та бути неправильно інтерпретованим. Ви навіть можете використовувати просте шістнадцяткове кодування, як використовується у PostScript. Однак для векторної графіки рішення - використовувати SVG.

## Растрові і векторні методи малювання. Зберігання растрових і векторних зображень. Формати растрові і векторні.+

***Растрова графіка*** складається з крихітних квадратів, званих пікселями. Після створення растрової графіки певного розміру (тобто фіксованої кількості пікселів) її неможливо змінити без втрати якості зображення. Чим більша кількість пікселів у зображенні, тим більший розмір файлу - вони позитивно співвідносяться, оскільки комп'ютеру потрібно зберігати інформацію про кожен піксель. Широко використовуються формати растрових файлів: .jpg, .png, .gif, .bmp та .tiff.

***Векторна графіка***використовує математичні рівняння, щоб намалювати свої проекти. Ці математичні рівняння переводяться в точки, які з'єднані або лініями, або кривими, також відомими як векторні контури, і вони складають усі різні фігури, які ви бачите у векторній графіці.

Це дозволяє масштабувати векторну графіку до будь-якого розміру без шкоди для якості зображення, а також підтримувати невеликий розмір файлу. Поширені векторні формати файлів: .svg, .cgm, .odg, .eps і .xml.

**Формати графічних файлів. Растрові формати**

*JPEG / JFIF*

JPEG (Joint Photographic Experts Group) - метод стиснення втрат. Стислі зображення JPEG зазвичай зберігаються у форматі файлів JFIF (JPEG File Interchange Format). Розширення назви файлів JPEG / JFIF - JPG або JPEG. Майже кожен цифровий фотоапарат може зберігати зображення у форматі JPEG / JFIF, який підтримує **восьмибітні зображення сірого кольору та 24-бітні кольорові зображення** (вісім біт для червоного, зеленого та синього). JPEG застосовує стиснення втрат до зображень, що може призвести до значного зменшення розміру файлу. Програми можуть визначати ступінь стиснення, що застосовується, а кількість стиснення впливає на візуальну якість результату. Якщо не надто велике, стиснення непомітно впливає на якість зображення чи не погіршує його, але JPEG- файли зазнають деградації поколінь при повторному редагуванні та збереженні. (JPEG також забезпечує зберігання зображень без втрат, але версія без втрат не підтримується широко.)

*PNG*

Формат файлу PNG (Portable Network Graphics) був створений як безкоштовна

альтернатива GIF з відкритим кодом. Формат файлу PNG підтримує восьмибітні палітри зображень (з додатковою прозорістю для всіх кольорів палітри) та 24-бітну truecolor (16 мільйонів кольорів) або 48-бітну truecolor з альфа-каналом і без нього - в той час як GIF підтримує лише 256 кольорів і один прозорий колір.

Порівняно з JPEG, PNG є кращим, якщо зображення має великі, рівномірні кольори. Навіть для фотографій - де JPEG часто є вибором для остаточного розповсюдження, оскільки техніка стиснення зазвичай дає менші розміри файлів - PNG все ще добре підходить для зберігання зображень під час процесу редагування через стиснення без втрат.

PNG забезпечує безпатентну заміну GIF (хоча GIF сам по собі зараз не є патентом), а також може замінити багато поширених видів використання TIFF. Підтримуються зображення з індексованим кольором, відтінками сірого та справжнім кольором, плюс додатковий альфа-канал. Adam7 переплетення дозволяє ранній попередній перегляд, навіть коли був переданий лише невеликий відсоток даних зображення. PNG може зберігати дані гамми та кольоровості для поліпшення відповідності кольорів на неоднорідних платформах.

PNG розроблений для того, щоб добре працювати в онлайнових програмах перегляду, таких як веб-браузери, і може бути повністю потоково переданий за допомогою прогресивної опції відображення. PNG надійний, забезпечує як повну перевірку цілісності файлів, так і просте виявлення поширених помилок передачі.

*GIF*

GIF (формат графічного обміну) в звичайному використанні обмежується 8-бітною палітрою, або 256 кольорів ( в той час як 24-бітна глибина кольору технічно можлива). GIF найбільше придатний для зберігання графіки з кількома кольорами, наприклад простими діаграмами, формами, логотипами та зображеннями в стилі мультфільму, оскільки використовує стиснення без втрат LZW, що є більш ефективним, коли великі площі мають один колір, і менш ефективні для фотографічних або пофарбованих зображень. Завдяки простоті та віку GIF досяг майже універсальної програмної підтримки. Завдяки анімаційним можливостям він все ще широко використовується для надання ефектів анімації зображень, незважаючи на низький коефіцієнт стиснення порівняно з сучасними відеоформатами.

*BMP*

Формат файлу BMP (растрова карта Windows) обробляє графічні файли в операційній системі Microsoft Windows. Зазвичай файли BMP не стискаються, а значить, великі та без втрат; їх перевагою є їх проста структура та широке сприйняття в програмах Windows.

*TIFF*

Формат TIFF (Tagged Image File Format) - гнучкий формат, який зазвичай зберігає вісім біт або шістнадцять біт на колір (червоний, зелений, синій) для 24-бітних і 48-бітових зображень відповідно, зазвичай використовуючи розширення імені файлу TIFF, або TIF. Ця структура була розроблена так, щоб вона легко розширювалася, і багато виробників ввели власні власні теги спеціального призначення - в результаті чого ніхто не читає будь-який варіант файлу TIFF. TIFF можуть бути втратними або без втрат, залежно від метода, обраного для зберігання пікселів зображення. Дехто пропонує порівняно гарну компресію без втрат для дворівневих (чорно-білих) зображень. Деякі цифрові камери можуть зберігати зображення у форматі TIFF, використовуючи алгоритм стиснення LZW для зберігання без втрат. Формат зображення TIFF не підтримується широко веб-браузерами. TIFF залишається широко прийнятим як стандарт фотофайлів у поліграфічній справі. TIFF може обробляти кольорові простори, характерні для пристрою, такі як CMYK, визначений певним набором фарб друкарського друку. Програмні пакети OCR (Optical Character Recognition) зазвичай генерують певну форму зображення TIFF (часто однотонне ) для сканованих текстових сторінок.

**Формати графічних файлів. Векторні формати**

*SVG*

SVG (масштабована векторна графіка) - це відкритий стандарт, створений та

розроблений Всесвітнім консорціумом веб-сторінок для вирішення потреб (та спроб декількох корпорацій) в універсальному, доступному для написання та універсальному векторному форматі для Інтернету та іншим способом. Формат SVG не має власної схеми стиснення, але через текстовий характер XML графіку SVG можна стиснути за допомогою програми, такої як gzip. Через свій сценарій сценаріїв, SVG є ключовим компонентом у веб-додатках: інтерактивні веб-сторінки, які виглядають і діють як програми.

*CGM*

CGM (Computer Graphics Metafile) - це формат файлів для двовимірної векторної графіки, растрової графіки та тексту, визначається ISO/IEC 8632. Усі графічні елементи можна вказати у текстовому вихідному файлі, який можна скласти у двійковий файл або в одне з двох текстових подань. CGM забезпечує засіб обміну графічними даними для комп’ютерного представлення 2D графічної інформації, незалежної від будь-якого конкретного додатку, системи, платформи чи пристрою. Вона певною мірою була прийнята в галузі технічної ілюстрації та професійного дизайну, але значною мірою його витіснили формати, такі як SVG та DXF .

*ODF*

Формат Open Document для додатків Office, (ODF), також відомий як OpenDocument , є ZIP стиснутим XML-форматом файла для електронних таблиць, діаграм, презентацій і обробки текстів документів. Він був розроблений з метою надання відкритої специфікації формату файлів на основі XML для офісних програм. Стандарт був розроблений технічним комітетом в консорціумі Організації просування структурованих інформаційних стандартів (OASIS). Він базувався на специфікації Sun Microsystems для OpenOffice.org XML, форматі за замовчуванням для OpenOffice.org та LibreOffice. Спочатку було розроблений для StarOffice, "щоб забезпечити відкритий стандарт для офісних документів". Крім того , що стандарт OASIS, він був опублікований як ISO/IEC міжнародного стандарту ISO/IEC 26300 - Формат Open Document для додатків Office (OpenDocument).

*EPS*

Інкапсульований PostScript (EPS) - це конвенція про структурування документів - відповідність (DSC) формату документа PostScript, яку можна використовувати у форматі графічного файлу. Файли EPS - це більш-менш автономні, досить передбачувані документи PostScript, що описують зображення або малюнок, і їх можна помістити в інший документ PostScript. Файл EPS - це по суті програма PostScript, яка зберігається як єдиний файл, який включає попередній перегляд із низькою роздільною здатністю, "інкапсульований" всередині документа, що дозволяє деяким програмам відображати попередній перегляд на екрані.

Файл EPS містить коментар DSC BoundingBox, що описує прямокутник, який містить зображення, описане у файлі EPS. Програми можуть використовувати цю інформацію для викладення сторінки, навіть якщо вони не в змозі безпосередньо відобразити PostScript всередині.

*XML*

Як можна використовувати графіку в XML? Зробити посилання на HTML або використати XLink. Або вбудувати SVG.

Графіка традиційно є лише посиланнями, які, як правило, мають файл зображення в результаті опрацювання, а не інший фрагмент тексту. Тому вони можуть бути реалізовані будь-яким способом, який підтримується специфікаціями XLink та XPointer, включаючи використання аналогічного синтаксису для існуючих зображень HTML. На них також можна посилатися, використовуючи вбудований механізм NOTATION та ENTITY XML аналогічно стандартному SGML, як і зовнішні нерозділені об'єкти.

Однак масштабована векторна графіка (специфікація XML для векторної графіки) дозволяє використовувати розмітку XML для малювання об'єктів векторної графіки безпосередньо у вашому XML-файлі. Це забезпечує величезну потужність для включення портативної графіки, особливо інтерактивної або анімованої послідовності, і тепер вона стає підтримуваною в браузерах, і їх можна експортувати зі стандартних графічних програм (малюнків), таких як GIMP.

Характеристики посилань на XML для зовнішніх зображень дають набагато кращий контроль над переходом та активацією посилань, тому автор може, наприклад, показувати зображення під час завантаження сторінки чи вибору мишкою в окремому вікні, не вдаючись до сценаріїв. Сам XML не передбачає або обмежує формати графічних файлів: GIF, JPG, TIFF, PNG, CGM, EPS та SVG. Проте векторні формати (EPS, SVG), як правило, необхідні для нефотографічних зображень (діаграм).

*Вбудована двійкова графіка*

Не можна вбудовувати необроблений бінарний графічний файл (або будь-який інший бінарний [нетекстовий]) безпосередньо у XML-файл, оскільки будь-який байт, що нагадує розмітку, буде неправильно інтерпретований: потрібно зробити на нього посилання. Однак можна включити текстово-кодовану трансформацію бінарного файлу як розділ, позначений CDATA, використовуючи щось на зразок UUencode з символами розмітки ], та >, і викреслити з карти, щоб вони не були подані як помилка CDATA і припинення послідовності та бути неправильно інтерпретованим. Ви навіть можете використовувати просте шістнадцяткове кодування, як використовується у PostScript. Однак для векторної графіки рішення - використовувати SVG.

## Принципи будови графічних редакторів на основі растрових і на основі векторних зображень. Характеристика особливостей растрового і векторного малювання, переваги і недоліки кожного методу. +

**Растровий графічний редактор** — спеціалізована програма для створення і обробки растрових зображень. Ці програмні продукти знайшли широке застосування в роботі художників-ілюстраторів, при підготовці зображень до друку або на фотопапері, публікації в інтернеті.

Растрові графічні редактори дозволяють користувачеві створювати і редагувати зображення на екрані комп'ютера (серед звичних інструментів — декілька типів ліній, стирання, копіювання об'єктів, додавання тексту, заповнення кольору фону…), а також зберігати їх в різних растрових форматах. Формати збереження зображень поділяються на такі, що дозволяють зберігати растрову графіку з незначним зниженням якості за рахунок використання алгоритмів стиснення з втратами (JPEG, PNG, GIF і TIFF), та такі, що також підтримують стиснення (RLE), але загалом є «попіксельним» описом зображення (BMP).

На противагу векторним редакторам, растрові використовують для утворення зображень матрицю точок (*bitmap*). При цьому, більшість сучасних растрових редакторів містять векторні інструменти редагування як допоміжні.

**Векторний графічний редактор** - спеціалізована програма, призначена для створення та обробки векторних зображень. Такі програми використовуються в роботі художників і дизайнерів різних напрямків, конструкторів, мультиплікаторів, у сфері ділової графіки, для презентацій, публікації в Інтернеті і т.п.

Векторні графічні редактори надають можливість створювати і редагувати векторні зображення безпосередньо на екрані комп'ютера, а також зберігати їх у різних векторних форматах.

Перелічимо основні інструменти векторних редакторів.

* + Криві Безьє - дозволяють створювати прямі, ламані і гладкі криві, що проходять через вузлові точки, з певними дотичними в цих точках.
  + Набір геометричних фігур (примітивів).
  + Заливка - дозволяє зафарбовувати обмежені області певним кольором, або градієнтом.
  + Текст створиться за допомогою відповідного інструменту, а потім часто перетворюється в криві [2] (у векторне зображення), щоб забезпечити незалежність зображення від шрифтів, наявних (або відсутніх) на комп'ютері, використовуваному для перегляду.
  + Олівець - дозволяє створювати лінії "від руки".

**Характеристика растрового зображення*:***

*Розмір* – ширина та висота. Вимірюється в пікселях, сантиметрах, дюймах.

*Роздільність* – кількість пікселів на одиницю довжини. Вимірюється в пікселях на дюйм (dpi) або пікселях на см..

*Глибина кольору* – кількість бітів, що використовується для кодування кольору одного пікселя. Вимірюється в бітах на піксель (bpp).

**Характеристика векторного зображення:** колір та товщина контуру, колір та спосіб заливки, розмір.

## Переваги та недоліки растрових і векторних зображень

|  |  |  |
| --- | --- | --- |
| **Вид графічного зображення** | **Переваги** | **Недоліки** |
| **Растрове** | * Реалістичність зображень. * Природність кольорів. * Можливість отримання зображень з використанням спеціальних пристроїв | * Великі за розміром файли зображень. * Пікселізація зображення після збільшення. * Складність редагування ок­ремих елементів зображен­ня |
| **Векторне** | * Невеликі за розміром файли зображень. * Збереження якості після масштабування. * Простота редагування окремих елементів зображення | * Складність реалістичного відтворення об’єктів навко­лишнього середовища. * Відсутність пристроїв для автоматизованого створення зображення |

## Події Windows, зв'язані з малюванням і відновленням зображень у вікні. Перелік подій та їх характеристика.+

|  |  |
| --- | --- |
| [MouseClick](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.mouseclick) | Ця подія виникає, коли користувач натискає елемент керування. Обробник цієї події приймає аргумент типу MouseEventArgs. Обробляти цю подію слід в разі, коли необхідно отримати відомості про миш при клацанні. |
| [MouseDoubleClick](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.mousedoubleclick) | Ця подія виникає, коли користувач двічі клацає мишею. Обробник цієї події приймає аргумент типу MouseEventArgs. Обробляти цю подію слід в разі, коли необхідно отримати відомості про миш при подвійному клацанні. |
| [MouseDown](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.mousedown) | Ця подія відбувається при натисканні користувачем кнопки миші, коли курсор миші знаходиться на елементі управління. Обробник цієї події приймає аргумент типу MouseEventArgs. |
| [MouseMove](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.mousemove) | Ця подія виникає при переміщенні курсору миші на елемент управління. Обробник цієї події приймає аргумент типу MouseEventArgs. |
| [MouseUp](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.mouseup) | Ця подія виникає, коли курсор миші знаходиться на елементі управління і користувач відпускає кнопку миші. Обробник цієї події приймає аргумент типу MouseEventArgs. |

## Стандартні класи системи програмування для малювання. Класи без зберігання малюнка (без поля пам'яті) і класи зі зберіганням малюнка у власній пам'яті.+

**Клас Pen**

З допомогою класу Pen створюється об’єкт-перо, який використовують для малювання ліній. Подаємо огляд потенційних параметрів класу. В різних системах програмування можуть бути реалізовані такі самі параметри або подібні за змістом. Остаточна реалізація залежить від можливостей бібліотеки графічних функцій.

Властивості класу є такі:

|  |  |
| --- | --- |
| property Color | Колір ліній, які креслить перо |
| property Width | Товщина ліній у пікселах екрана |
| property Style | Визначає стиль ліній (див. нижче). Враховується тільки для ліній товщиною в 1 піксел. Для товстих ліній завжди psSolid (суцільна) |
| property Mode | Визначає спосіб взаємодії ліній з тлом (див. нижче) |
| property Handle | Дескриптор пера. Використовується при беспосередньому звертанні до функцій API |

Тип стилю лінії **Style** визначається так:

enum TPenStyle = { psSolid, psDash, psDot, psDashDot, psDashDotDot, psClear, psInsideFrame };

Тип взаємодії лінії з тлом визначається так:

enum TPenMode = { pmBlack, pmWhite, pmNop, pmNot, pmCopy, pmNotCopy, pmMergePenNot, pmMaskPenNot, pmMergeNotPen, pmMaskNotPen, pmMerge, pmNotMerge, pmMask, pmNotMask, pmXor, pmNotXor };

Властивість **Mode** може набувати одного з таких значень:

|  |  |
| --- | --- |
| pmBlack | Лінії завжди чорні. Властивості Color і Style ігноруються |
| pmWhite | Лінії завжди білі. Властивості Color і Style ігноруються |
| pmNop | Колір тла не змінюється (лінії невидимі) |
| pmNot | Інверсія кольору тла. Властивості Color і Style ігноруються |
| pmCopy | Колір ліній визначається властивістю Color пера |
| pmNotCopy | Інверсія кольору пера. Властивість Style ігнорується |
| pmMergePenNot | Комбінація кольору пера та інверсного кольору тла |
| pmMaskPenNot | Комбінація спільних кольорів для пера та інверсного кольору тла. Властивість Style ігнорується |
| pmMergeNotPen | Комбінація кольору тла та інверсного кольору пера |
| pmMaskNotPen | Комбінація спільних кольорів для інверсного кольору пера і кольору тла. Властивість Style ігнорується |
| pmMerge | Комбінація кольорів пера і тла |
| pmNotMerge | Інверсія комбінації кольорів пера і тла (інверсія pmMerge) |
| pmMask | Спільні кольори пера і тла |
| pmNotMask | Інверсія спільних кольорів пера і тла (інверсія pmMask) |
| pmXor | Об’єднання кольорів пера і тла операцією XOR |
| pmNotXor | Інверсія об’єднання кольорів пера і тла операцією XOR (інверсія pmXor) |

Зауважимо, що ми записали потенційні способи взаємодії з тлом. Залежно від системи програмування і наявної графічної бібліотеки можуть бути доступні лише окремі способи взаємодії.

Властивість **Style** може приймати одне із таких значень:

\_\_\_\_\_\_\_\_\_\_\_\_\_ psSolid суцільна лінія

\_ \_ \_ \_ \_ \_ \_ \_ \_ psDash штрихова дінія

…...................... psDot пунктирна лінія

\_ . \_ . \_ . \_ . \_ . \_ psDashDot штрихпунктирна лінія

\_ .. \_ .. \_ .. \_ .. \_ psDashDotDot серії штрихів і двох точок

psClear лінія не малюється; (використовується для

того, щоб не малювати лінію довкола фігури,

залишаючи незмінними параметри пера)

\_\_\_\_\_\_\_\_\_\_\_\_\_ psInsideFrame суцільна лінія, яка буде малюватися

при товщині лінії Width, більшій, ніж 1,

всередину фігури від заданої координати

**Клас Brush**

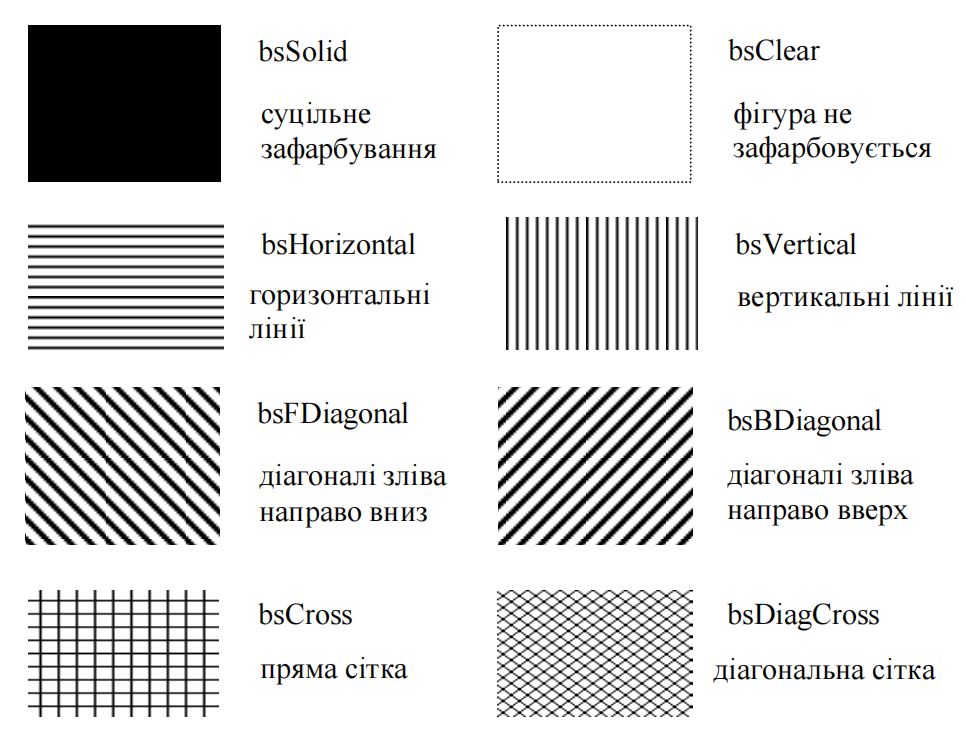
Об’єкти класу Brush (пензлі) використовують для заповнення внутрішнього простору замкнених фігур. Властивості класу можуть бути такими:

|  |  |
| --- | --- |
| property Color | Колір пензля |
| property Style | Стиль пензля – спосіб зафарбування внутрішньої ділянки фігури (див. нижче) |
| property Bitmap | Визначає зовнішнє растрове зображення, котре буде використане пензлем для заповнення. Якщо ця властивість визначена, властивості *Color* і *Style* ігноруються |
| property Handle | Дескриптор пензля. Використовується при безпосередньому звертанні до АРІ-функцій Windows |

Тип стилю пензля визначається так:

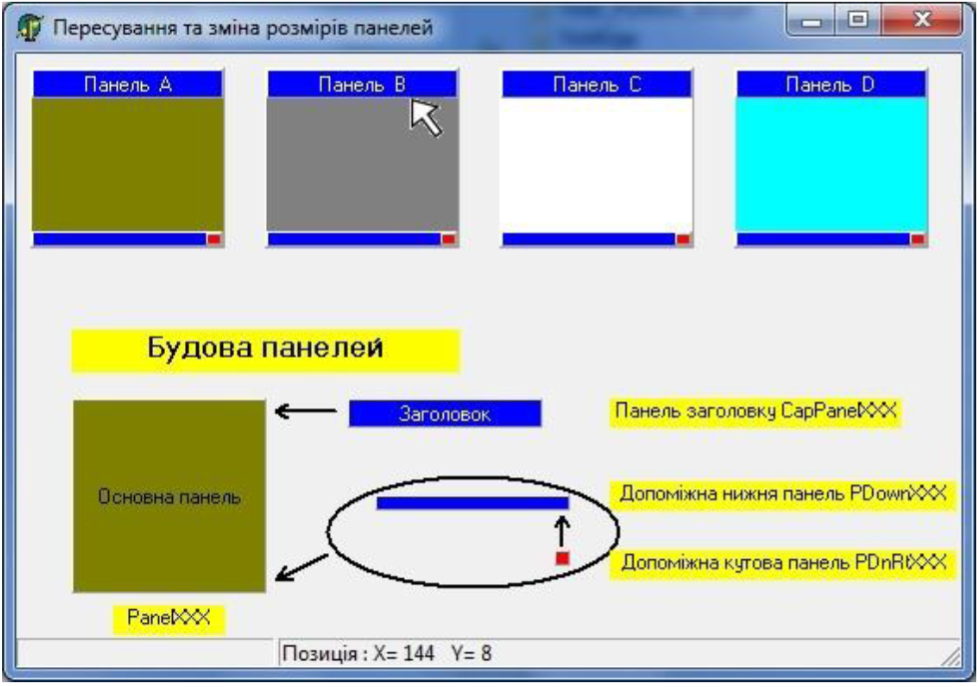
enum TBrushStyle = { bsSolid, bsClear, bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal, bsCross, bsDiagCross };

При цьому зафарбування буде таким:



## Поняття про пересування та зміну розмірів видимих елементів вікна під час виконання програми.+

Розглянемо деякі прийоми програмування щодо пересування та зміни розмірів видимих елементів форми програми, стандартні для Windows. Мова йде про пересування та зміну розмірів під час виконання програми за нашими командами, наприклад, від мишки. Для програмування треба вибрати за основу деякі події. У випадку мишки найкраще скористатись подіями MouseDown, MouseMove, MouseUp. Нехай, наприклад, маємо проєкт форми програми, як показано на рисунку.



Розташовуємо на формі для прикладу 4 панелі поряд (PanelA, PanelB, ...). Зверху на кожну панель накладаємо іншу - меншу, яка відіграватиме роль заголовка (CapPanelA, CapPanelB, ...). Знизу на кожну панель також накладаємо вузеньку допоміжну панель (PDownA, PDownB, ...), а на цю допоміжну справа - ще одну допоміжну кутову, за допомогою якої змінюватимемо розміри основної панелі (PDnRtA, PDnRtB, ...). Колір заголовної панелі - синій, підпис на ній - білий, тоді така панель є дуже подібною до заголовка звичайного вікна Windows. Допоміжні панелі знизу мають бути невеликого розміру по висоті, їхній колір можна обрати довільно. Важливо для допоміжної кутової панелі визначити вид курсора як діагональної двоспрямованої стрілки, тоді під час виконання програми будемо бачити момент можливої зміни висоти та ширини основної панелі.

Зв'язування панелей між собою можна здійснити так: під час проектування розташувати їх на формі де завгодно, а на початку виконання програми (за подією, наприклад, OnCreate форми) визначити для всіх допоміжних панелей відповідні їм батьківські панелі (Parent) та спосіб вирівнювання Align щодо батьківської. В такому разі події одних елементів (дочірніх, надбудованих, зверху) можна передавати (транслювати) до батьківських елементів, а вже батьківські елементи будуть реагувати і виконувати потрібні операції. На формі знизу розташуємо панель статусу StatusBar класу TStatusBar, з вирівнюванням alBottom, де записуватимемо позицію панелі, яка пересувається чи змінює розміри. А до панелі StatusBar додаємо через Object Inspector і властивість Panels два елементи класу TStatusPanel – окремі поля цілої панелі для різних даних.

## Події миші і клавіатури, які можна використати для пересування і зміни розмірів видимих елементів вікна під час виконання програми. Параметри подій.+

Для програмування треба вибрати за основу деякі події. У випадку мишки найкраще скористатись подіями MouseDown, MouseMove, MouseUp.

|  |  |
| --- | --- |
| [MouseDown](https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.control.mousedown) | Ця подія відбувається при натисканні користувачем кнопки миші, коли курсор миші знаходиться на елементі управління. Оброблювач цієї події бере аргумент типу MouseEventArgs. |
| [MouseMove](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.mousemove) | Ця подія виникає при переміщенні вказівника миші на елемент керування. Оброблювач цієї події бере аргумент типу MouseEventArgs. |
| [MouseUp](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.mouseup) | Ця подія виникає, коли вказівник миші знаходиться на елементі управління і користувач відпускає кнопку миші. Оброблювач цієї події бере аргумент типу MouseEventArgs. |
| [MouseWheel](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.mousewheel) | Ця подія виникає, коли користувач обертає коліщатко миші, коли фокус знаходиться на елементі управління. Оброблювач цієї події бере аргумент типу MouseEventArgs. Для визначення того, наскільки прокручено коліщатко миші, можна використовувати властивість Delta елемента MouseEventArgs. |

У випадку клавіатури:

|  |  |  |
| --- | --- | --- |
| [KeyDown](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.keydown) | Ця подія виникає, коли користувач натискає фізичну клавішу. | Оброблювач KeyDown отримує:    *Параметр* KeyEventArgs, який надає властивість KeyCode (вказує на фізичну клавішу клавіатури).  *Властивість* Modifiers (SHIFT, CTRL або ALT).  *Властивість* KeyData (яке об'єднує код клавіші і модифікатор).  Параметр KeyEventArgs також надає:  *Властивість* Handled, яке може бути задано для запобігання отримання коду клавіші базовим елементом управління.  *Властивість* SuppressKeyPress, яке може використовуватися для придушення подій KeyPress і KeyUp для даного натискання клавіші. |
| [KeyPress](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.keypress) | Ця подія виникає якщо в результаті натискання клавіші або клавіш виходить символ. Наприклад, користувач натискає клавіші SHIFT і малу "a", в результаті виходить символ великої літери "A". | KeyPress виникає після KeyDown.    Оброблювач KeyPress отримує:  *Параметр* KeyPressEventArgs, який містить код символу натиснутоюклавіші. Цей код є унікальним для кожної комбінації клавіш символу і модифікатора.    Наприклад клавіша "A" створить    - код символу 65, якщо вона натиснута, утримуючи клавішу "SHIFT"  - Або клавіша CAPS LOCK поверне код 97, якщо вона натиснута сама по собі,  - І код 1, якщо вона натиснута спільно з клавішею CTRL. |
| [KeyUp](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.keyup) | Ця подія виникає, коли користувач відпускає фізичну клавішу. | Оброблювач KeyUp отримує:    *Параметр* KeyEventArgs,  - який надає властивість KeyCode (вказує на фізичну клавішу клавіатури).  *- Властивість* Modifiers (SHIFT, CTRL або ALT).  *- Властивість* KeyData (яке об'єднує код клавіші і модифікатор). |

## Об'єкт пересування та зміни розміру, зміщення об'єкта в процесі пересування, зовнішні розміри об'єкта, поточна канва малювання.+

Щоб перемістити об'єкт:

Використовуйте клавіші зі стрілками на клавіатурі, щоб перемістити об’єкт горизонтально або вертикально. Об'єкт рухатиметься на 1 піксель у напрямку натиснутої клавіші зі стрілкою. Ви також можете виконати ці дії, утримуючи клавішу Shift. Це призведе до переміщення об’єкта з кроком 10 пікселів у обраному напрямку.

Клацніть на об'єкт і, утримуючи ліву кнопку миші, перетягніть об'єкт на нове місце. Щоб перемістити об’єкт у нове положення, яке знаходиться безпосередньо над, під або з будь-якої сторони його поточного положення, утримуючи клавішу Shift, перетягуючи елемент у нове місце. Це призведе до того, що об’єкт рухатиметься прямо, вгору, вниз, вліво або вправо.

Вручну встановіть положення об’єкта, змінивши положення X та Y на вкладці Позиція та розмір властивостей об’єкта.

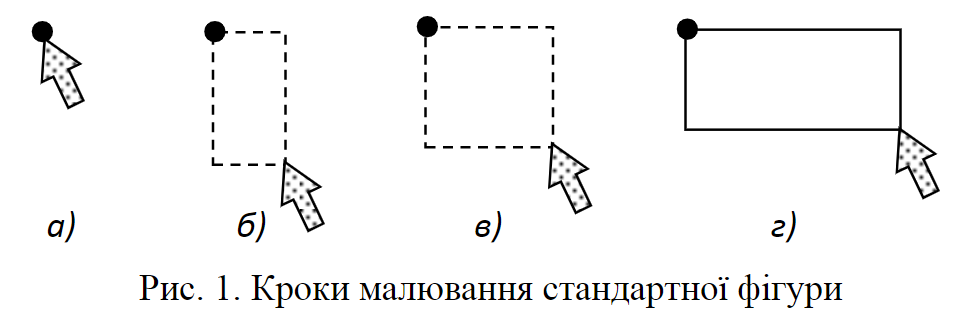
Щоб змінити розмір об’єкта:

Клацніть на будь-який кут або сторону об'єкта, і, утримуючи ліву кнопку миші, перетягніть кут або сторону об'єкта, щоб змінити його розмір. Утримання клавіші Shift на клавіатурі під час виконання цих кроків забезпечить пропорційний розмір об’єкта.

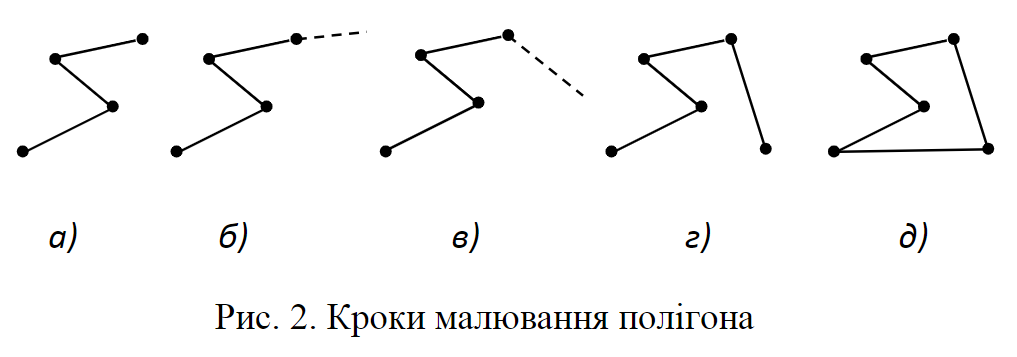
Встановіть розмір об’єкта вручну, змінивши Ширину та Висоту об’єкта на вкладці Позиція та Розмір властивостей об’єкта.

## Особливості малювання многокутників. Фіксування вершин полігону. Відображення проміжних полігонів. Динаміка малювання ребер полігона. Події, придатні до малювання полігона і відображення динаміки малювання.+

Розглянемо кроки малювання стандартних фігур на прикладі прямокутника. Початок фігури фіксується у момент натискання на клавішу мишки (рис.1,а). При пересуванні мишки з натисненою клавішею малюємо прямокутник від точки початку фігури до точки поточного розташування мишки (рис.1,б). При подальшому пересуванні мишки стираємо попередній прямокутник і малюємо його у новому місці (рис.1,в) – така операція виконується багатократно, при кожній зміні позиції мишки. Остаточний прямокутник фіксується у момент відпускання клавіші мишки (рис.1,г).



У загальному випадку замкнений многокутник (в іншій термінології –полігон) можна малювати описаним вище методом. Однак усі проміжні полігони до остаточного вибору не показують перспективи продовження шляхом додавання нових ланок і дають неправильну уяву про кінцевий результат. Тому варто у процесі побудови малювати лише окремі ребра, які вже вибрано (Polyline). Необхідно домогтися фіксування вершин полігона. З цією метою можна використати подію OnClick. Наприклад, на рис.2,а зазначено вже побудовану частину полігона. Наступне ребро для продовження полігона малюємо подібно, як звичайну лінію, описаним вище методом (рис.2,б,в), лише пересувати мишку можна й без натисненої клавіші. Наступне ребро фіксується подальшим натисненням клавіші мишки (подія OnClick) (рис.2,г). Закінчення побудови полігона – за подією OnDblClick, при цьому остання вершина сполучається з найпершою (рис.2,д).



## Масштабування зображень в процесі малювання. Проблеми, які можуть виникати при ручному програмуванні масштабування.+

У процесі редагування можна подавати малюнок у збільшеному масштабі з метою точного малювання дрібних елементів малюнка. Виглядає доцільним обмежитись фіксованим переліком збільшення масштабу в 2, 4 чи 8 разів. Крім того, масштабування кратне степеням 2 дозволяє зберігати якість растрового формату при зменшенні назад, бо непотрібна апроксимація пікселів. Проєкт форми для вибору масштабу подано на рис.7. Варто зробити важливе застереження. Збільшення масштабу малюнка в *n* разів може вимагати збільшення пам’яті комп’ютера в *n2* разів для його зберігання та опрацювання (за растровими форматами) – залежно від способу зберігання. Тому до збільшення масштабу треба підходити виважено, передусім для великих за розмірами малюнків.



## Опрацювання зображень способом фільтрування. Приклади фільтрів та їх формули. Застосування фільтрів до частини загального зображення.+

Фільтрування – спеціальний спосіб опрацювання малюнка шляхом перетворення кольорів. Перетворення виконується за певними формулами, що дають змогу отримати різні ефекти. При перетворенні виконується аналіз сусідніх ділянок малюнка (пікселів) довкола кожної точки обраної ділянки малюнка. Крім того, перетворення виконується окремо для кожного з трьох складових кольорів стандартного пікселя: червоного, зеленого, синього.

Операція фільтрування "Рисунок → Фільтр" застосовується до виокремленого прямокутного фрагмента малюнка. Якщо виокремлення немає – тоді до цілого малюнка.

***Фільтр «Яскравість / контрастність»***

Для зміни яскравості на N відсотків використовується наступна формула:

I = I + N • 128/100 (1)

де I - відповідно R, G, B канали кожної точки зображення.

Зменшення контрастності на N відсотків:

I = (I • (100 - N) + 128 • N) / 100 (2)

Збільшення контрастності на N відсотків:

I = (I • 100 - 128 • N) / (100 - N) (3)

Якщо нове I не потрапляє у діапазон 0..255 - то його слід урізати.

***Фільтр «Баланс кольору»***

Для зміни колірного балансу по одному з каналів R, G, B на N відсотків слід обчислити нове значення колірного каналу за формулою:

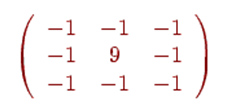
I = I + N • 128/100, де I - це R, G або B кожної точки зображення. (4)

Якщо нове I не влучає у діапазон 0..255 - то його слід урізати.

***Фільтри «Підвищити різкість» і «Розмити»***

Ці фільтри реалізуються на основі ядра згортки. Елемент зображення отримує нове значення на основі групи елементів, що примикають до даного. Областю примикання є квадратна матриця, розмірність якої збігається з розміром обраного ядра згортки, і центром в оброблюваному елементі.

При використанні алгоритму **збільшення різкості** підкреслюються відмінності між кольорами суміжних пікселів і виділяються непомітні деталі. В ядрі різкості центральний коефіцієнт більше 1, а оточений він від’ємними числами, сума яких на одиницю менша центрального коефіцієнта. Таким чином, збільшується будь-який існуючий контраст між кольором пікселя і кольорами його сусідів. При обробці кожного пікселя в зображенні використовується ядро різкості розмірами 3 × 3:



При використанні алгоритму **розмиття** в зображенні перерозподіляються кольори і пом'якшуються різкі границі. Ядро згладжування складається з сукупності коефіцієнтів, кожен з яких менший 1, а їх сума становить 1. Це означає, що після фільтрації кожен піксель поглине щось з кольорів сусідів, але повна яскравість зображення залишиться незмінною. При обробці зображення використовується наступне ядро згортки:



## 18. Контекстні підказки про хід виконання графічних операцій. Перемикання контекстом підказки.

# Відповіді до питань до розділу "Електронні таблиці"

## 1. Означення термінів "активна комірка", "впливаюча комірка", "залежна комірка", "ряд даних", "властивості комірки". Алгоритм розпізнавання статусу комірки. +-

Активна комірка - це комірка, з якою працюють у даний момент часу; у яку вводяться дані.

(https://sites.google.com/view/hmcinformatic/інформаційні-технології/кухар/3-розряд/табличний-процесор)

Впливаючі комірки - це комірки, на які посилаються формули. ()

Залежні комірки містять формули, що посилаються на адреси комірок електронної таблиці.

(<https://elearning.sumdu.edu.ua/free_content/lectured:c5dfdb13db13a6099b7e1489d805156fd10127f0/20200921190435//1782889/index.html>)

Ряд даних — це група взаємопов'язаних певних чином між собою елементів даних діаграми.

## 2. Формули в комірках ЕТ. Структура формули на рівні змістового призначення. Типи можливих синтаксичних і семантичних помилок в формулах. Критерії визначення помилки. +

Для точного визначення будови деякої синтаксичної конструкції приймемо такі позначення. Їх використання є загальноприйнятим, і має основою формули БНФ:

- нетермінали позначаємо звичайним ідентифікатором, наприклад number;

- термінали позначаємо лапками, наприклад "+", "(", "sqrt";

- круглі дужки використовуємо подібно, як в алгебрі, для групування інших конструкцій, наприклад ( "+" | "-" | "\*" | "/" ); самі дужки в такому записі не є частиною конструкції;

- вертикальна риска | означає "або", тобто вибір однієї з альтернатив;

- знак зірочки \* означає повторення нуль або більше разів конструкції, записаної перед зірочкою, наприклад cipher \* ; сама зірочка не є частиною конструкції;

- квадратні дужки [ ] означають або відсутність, або однократне входження в синтаксичний запис, наприклад [ prompt ] ;

- знак ::= читаємо "за означенням є".

Сканер фактично повинний розпізнавати лише такі групи лексем у формулах:

* однолітерні розділювачі (знаки);
* ідентифікатори (імена функцій і комірок);
* числа (різних форматів);
* літерали в одинарних лапках;
* літерали в подвійних лапках.

Все решту вважається помилкою запису формули. Сканер може виявляти помилки, які стосуються окремих лексем, і у випадку помилки повідомляє код помилки і лексему з помилкою. У випадку успішного сканування можна прочитати по черзі всі закодовані лексеми (метод GetToken).

## 3. Граматики для розпізнавання типу комірки. Алгоритм розпізнавання. +

Розглянемо тепер записи формул, виконані відомими алгебраїчними правилами. Наприклад, маємо синтаксичне означення правил запису формули з врахуванням загальних алгебраїчних правил:

arith\_expr ::= term ( ( "+" | "-" ) term ) \*

term ::= factor ( ( "\*" | "/" ) factor ) \*

factor ::= number | "(" arith\_expr ")"

number ::= cipher cipher \*

cipher ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Реалізацію граматики виразів і розпізнавання виразу реалізуємо методом нисхідного граматичного розбору за алгоритмом рекурсивного спуску. З теорії формальних мов і граматик відомо, що для реалізації методу рекурсивного спуску потрібно переписати граматичні правила у вигляді, де немає ліворекурсивних правил. Для нашого випадку ця вимога виконана.

За методом рекурсивного спуску до кожного нетермінального символа будують окрему функцію, яка повинна розпізнати праву частину граматичного правила. При цьому функція може викликати в потрібні моменти інші функції для розпізнавання частини конструкції. Наприклад, функція для arith\_expr буде викликати функцію для term, функція для term буде викликати функцію для factor, і т.д. Врешті за такою схемою можливі непрямі рекурсії, наприклад, arithexpr - term - factor - (arithexpr) - … .

## 4. Поняття лексеми формули. Перелік лексем на прикладі граматики формули. Формальне визначення лексем. +

(<https://uk.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B5%D0%BC%D0%B0> визначення звідси:)

Лексема — послідовність машинних символів вихідного коду програми, що мають певне сукупне значення.

Для більшості мов програмування актуальні такі класи лексем:

* зарезервовані слова
* ідентифікатори
* числові константи
* літерні константи
* рядкові константи
* коди операторів
* коментарі, які безпосередньо не несуть інформації щодо структури програми; транслятором не сприймаються, синтаксичному аналізатору не передаються.
* дужки й інші елементи програми.

Розглянемо тепер записи формул, виконані відомими алгебраїчними правилами. Наприклад, маємо синтаксичне означення правил запису формули з врахуванням загальних алгебраїчних правил:

arith\_expr ::= term ( ( "+" | "-" ) term ) \*

term ::= factor ( ( "\*" | "/" ) factor ) \*

factor ::= number | "(" arith\_expr ")"

number ::= cipher cipher \*

cipher ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Вхідними даними для програми обчислення виразу є масив лексем, побудований сканером. Арифметичний вираз може мати довільний вигляд в межах записаних вище синтаксичних правил. Тому доцільно підготувати список лексем в формі, зручній для синтаксичного розбору. Кожну лексему можна подати у вигляді (код, значення), де код – числове позначення лексеми кожного виду. Приймемо такі позначення лексем:

|  |  |  |
| --- | --- | --- |
| number | 1 | значення числа |
| openbracket | 3 | "(" |
| closebracket | 4 | ")" |
| add | 5 | "+" |
| subtract | 6 | "-" |
| multiply | 7 | "\*" |
| divide | 8 | "/" |

Список лексем легко отримати, переглядаючи синтаксичні правила визначення формули.

## 5. Загальна схема сканера. Визначення структур або класів для розпізнавання лексем сканером. Перелік параметрів лексем для їх опрацювання. +

Сканер – це програма, яка текст однієї формули ділить на окремі лексеми і кодує лексеми певним способом, наприклад, цілими числами. Отже, замість тексту формули будемо мати масив цілих чисел, що дозволить більш ефективно будувати програми перетворення і обчислення формул.

Наприклад, якщо маємо формулу = B5+B6-SUM(C4:C10)/2

то поділ на лексеми мав би виглядати так:



Щоб програмувати сканер, потрібно укласти повний список всіх допустимих лексем і визначити таблицю кодування лексем. В основі таблиці кодування має бути деякий принцип чи загальний підхід. Наприклад, для кожної лексеми можна надати три специфікатори: перший визначає тип лексеми, а один або два наступні уточнють зміст лексеми. Отже, для однієї лексеми можна визначити такі структури:

struct ETindex // індекс таблиці

{

int row, col; // індекси рядка і стовпця електронної таблиці

};

struct Ttoken // визначення лексеми

{

int token; // тип лексеми

union// спільна ділянка пам'яті (накладання)

{

ETindex index; // індекси рядка і стовпця

double cellsvalue; // або значення числа в комірці

char \* lit; // текстовий рядок (літерал)

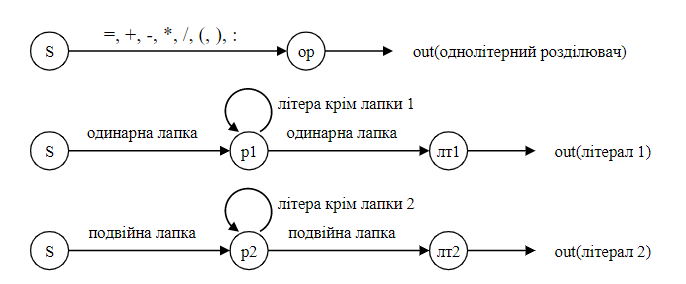
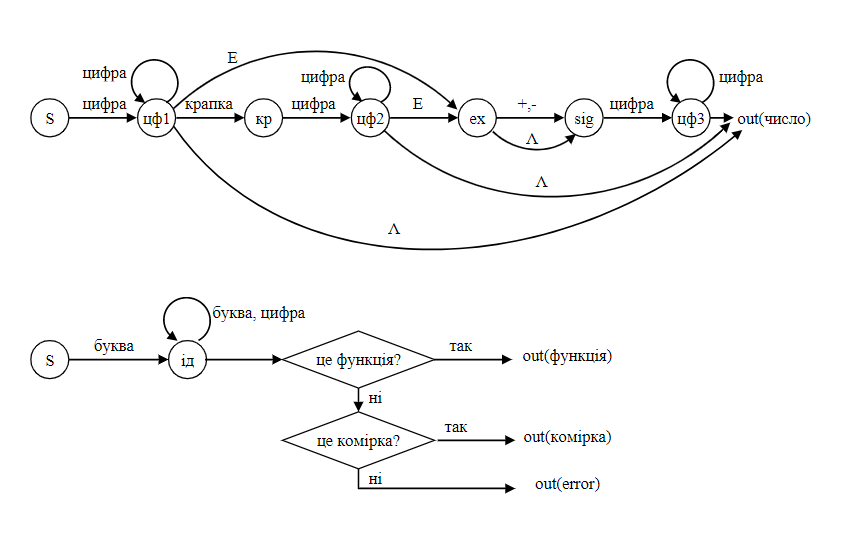
};

};

Величина token є головною і визначає тип лексеми. Величини row і col, які входять в структуру ETindex, є допоміжними і визначають уточнюючі параметри лексеми, а саме – номер рядка і номер стовпця комірки електронної таблиці, на яку є посилання в даній лексемі. Величина cellsvalue визначає безпосередньо число, якщо воно є значенням лексеми. З метою економії пам'яті використовуємо накладання величин (union) index і cellsvalue, враховуючи ту обставину, що лексема не може бути одночасно адресою комірки і числом. Величина lit є вказівником на текстовий рядок, що є значенням лексеми, у випадку, якщо лексема є літералом.

## 6. Алгоритми розпізнавання лексем формул у вигляді діаграми станів скінченного автомата. +

Оскільки для розпізнавання лексем потрібно аналізувати по черзі кожну окрему літеру формули, то алгоритми розпізнавання лексем доцільно подати у формі діаграм станів детермінованого скінченного автомата. Нижче подано відповідні діаграми станів. Позначення станів є умовними. Знаком "Λ" позначають пусту літеру, іншими словами, це є перехід без розпізнавання будь-якої літери за умови, що інші переходи неможливі. Out означає вихід з відповідною розпізнаною лексемою.



## 7. Схема основного циклу сканування формули і кодування лексем. +

// цикл сканування і кодування лексем;

//аналіз виконуємо за поточною літерою letter

while (letter != '\0' && !error)

{

Init(); // знайти першу значущу літеру від поточної позиції

if( error|| letter== '\0') break; // помилка функції Init()

//або кінець формули

// за першою літерою розпізнаємо групу, до якої належить лексема

if(letter>='0' && letter<='9') // це є число

{

// цифри перед крапкою

while(isdigit(letter)) { AddLetterToToken(); GetNextChar(); }

if(letter == '.') // може бути крапка

{

AddLetterToToken(); GetNextChar(); // сама крапка

// цифри після крапки

if( ! isdigit(letter)) // після крапки немає цифри

{codeerror=1002; error=true; break; // після крапки немає цифри}

else // прочитати цифри після крапки

while(isdigit(letter)) { AddLetterToToken(); GetNextChar}

} // if(letter == '.')

// далі за текстом може бути записаний порядок числа

if(letter=='E') // порядок присутній

{

AddLetterToToken(); GetNextChar(); // дописати букву E

if(letter=='+' || letter=='-') // може бути знак порядку

{ AddLetterToToken(); GetNextChar(); } // дописати знак

// далі мають бути цифри порядку

if( ! isdigit(letter)) // після букви E немає цифри

{

codeerror=1003; error=true; break; // після букви E немає цифри

}

// дописати цифри порядку

while(isdigit(letter)) { AddLetterToToken(); GetNextChar(); }

} // порядок присутній

// перетворити число з текстової форми в double і записати лексему

double x = atof(tx.c\_str());

toklist[N].token=number\_et; toklist[N].cellsvalue=x;

} // це є число

else

if(letter>='A' && letter<='Z') // це комірка або функція

{

// спочатку прочитати всі букви

while(letter>='A' && letter<='Z') {AddLetterToToken(); GetNextChar();}

// далі можуть бути цифри і букви в довільному порядку

while(isdigit(letter) || letter>='A' && letter<='Z'){ AddLetterToToken(); GetNextChar(); }

// перевіримо, чи це є іменем функції

int fnc = FunctionNameCode(tx);

if(fnc) { toklist[N].token=fnc; } // знайшли ім'я функції

else // перевіримо, чи це правильне ім'я комірки

{

// розділити лексему на букви і цифри

int i=0;

string tempcol="";

for(; i<tx.length() && tx[i]>='A'&&tx[i]<='Z'; i++) tempcol+= tx[i];string temprow="";for(; i<tx.length() && isdigit(tx[i]); i++) temprow+= tx[i];

int numrow=atoi(temprow.c\_str());

Int numcol=ConvertA1toR1C1(tempcol);

if(numrow>=1 && numrow<=99 && numcol>=1 && numcol<=256&& i>=tx.length())

{ // правильна лексема

toklist[N].token=cell\_et;

toklist[N].index.col=numcol;

toklist[N].index.row=numrow;

}

else // помилка

{

codeerror=1004; error=true; break;

// неправильне ім'я комірки або функції

}

} // перевірка правильності імені комірки

} // це комірка або функція

else

if(GetCodeTok1L(letter)>0) // це однолітерний розділювач

{

toklist[N].token=GetCodeTok1L(letter);GetNextChar (); // до наступної літери формули

} // це однолітерний розділювач

else

if(letter=='\'') // це літерал з одинарними апострофами

{ // читати до кінця літерала

GetNextChar(); // наступна літера після апострофа

while(letter != '\'' && letter != '\0')

{

AddLetterToToken(); GetNextChar(); }

toklist[N].token=lit1;

toklist[N].lit= new char[tx.length()+1]; // надати пам'ять

strcpy(toklist[N].lit,tx.c\_str()); // копіювати літерал

GetNextChar(); // перейти до наступної літери за апострофом

} // це літерал з одинарними апострофами

else

if(letter=='\"') // літерал з подвійними апострофами

{ // читати до кінця літерала

GetNextChar(); // наступна літера після апострофа

while(letter != '\"' && letter != '\0')

{

AddLetterToToken(); GetNextChar(); }

toklist[N].token=lit2;

toklist[N].lit= new char[tx.length()+1]; // надати пам'ять

strcpy(toklist[N].lit,tx.c\_str()); // копіювати літерал

GetNextChar(); // перейти до наступної літери за апострофом

} // літерал з подвійними апострофами

else // помилка -невизначена лексема

{

AddLetterToToken(); // запам'ятати недопустиму літеру в лексемі

codeerror=1001; // невизначена лексема (недопустима літера)

error=true;

} // помилка -невизначена лексема

} // while (letter != '\0' && !error)

## 8. Програмовані функції для кодування і декодування адресів комірок ЕТ. Зображення комірок як лексем.

## 9. Перетворення формул ЕТ з інфіксної форми у постфіксну. Правила перетворення. Алгоритми, необхідні для перетворення. +

В постфіксній формі операція має бути записана за операндами, до яких вона стосується. Знак "=", який є на початку формули, можна викреслити з постфіксної форми, бо він не є операцією, а лише ознакою формули в комірці таблиці. Дужки в постфіксній формі відсутні, бо всі операції виконують зліва направо в порядку записування.

Граматичні правила для формул запишемо в такому вигляді:

Формули:

<формула> ::= = <вираз>

<вираз> ::= <доданок> { (+ | -)<доданок>}

<доданок> ::= <множник> { ( \* | / ) <множник> }

<множник> ::= <число> | <комірка> | <функція> | '('<вираз> ')'

<комірка> ::= <букви><цифри>

<букви>::= буква | буква буква

<цифри> ::= цифра | цифра цифра

<функція> ::= ( SUM| MAX| MIN) '('<ряд> ')'

|( ABS| INT)'('<вираз > ')'

<ряд> ::= <комірка> : <комірка>

За методом рекурсивного спуску до кожного нетермінального символа будують окрему функцію, яка повинна розпізнати праву частину граматичного правила. При цьому функція може викликати в потрібні моменти інші функції для розпізнавання частини конструкції. Наприклад, функція для <вираз> буде викликати функцію для <доданок>, функція для <доданок> буде викликати функцію для <множник>, і т.д. Врешті за такою схемою можливі непрямі рекурсії, наприклад, <вираз>—<доданок>—<множник >—<вираз>—...

Вхідними даними для програми перетворення з інфіксної форми в постфіксну є масив лексем, побудований сканером. Отже, всі лексеми вже мають однаковий числовий формат, що значно спрощує програмування такого перетворення. Відповідно, вихідними даними цієї програми є перебудований масив лексем в тому самому числовому форматі. Головна ідея генерування постфіксної форми полягає в правильному визначенні послідовності записування елементів виразу в постфіксну форму. Постфіксна форма – це спільний масив (потік) для всіх функцій розпізнавання нетермінальних символів. Функції почергово записують в такий спільний масив елементи виразу, які вони розпізнали.

Загальну схему взаємодії і використання сканера і перетворювача формул до постфіксної форми можна подати такою діаграмою:



## 10. Побудова постфіксної форми виразів на основі граматики виразів. Нисхідний граматичний розбір виразів методом рекурсивного спуску.

## 11. Модифікація граматичних правил виразів для випадку розбору методом рекурсивного спуску. Питання ліво рекурсивних правил і факторизації.

## 12. Алгоритм взаємодії функцій розпізнавання елементів виразів для перетворення формул ЕТ з інфіксної форми у постфіксну.

## 13. Алгоритм фіксування помилок у формулах і друкування діагностики в процесі перетворення до постфіксної форми.

## 14. Інтерпретація формул ЕТ на основі постфіксної форми зображення. Структура основного алгоритму інтерпретації.

## 15. Алгоритм застосування стеку і правила опрацювання постфіксної форми виразів в процесі інтерпретації.

## 16. Алгоритм опрацювання зв'язаних формул в процесі інтерпретації (залежна-впливаюча).

## 17. Організація алгоритму інтерпретації прямо рекурсивної формули. Необхідні допоміжні параметри для коректної інтерпретації та їх використання