**Іспит з ОС і СП**

Студента ПМІ-42

Зелінського Олександра

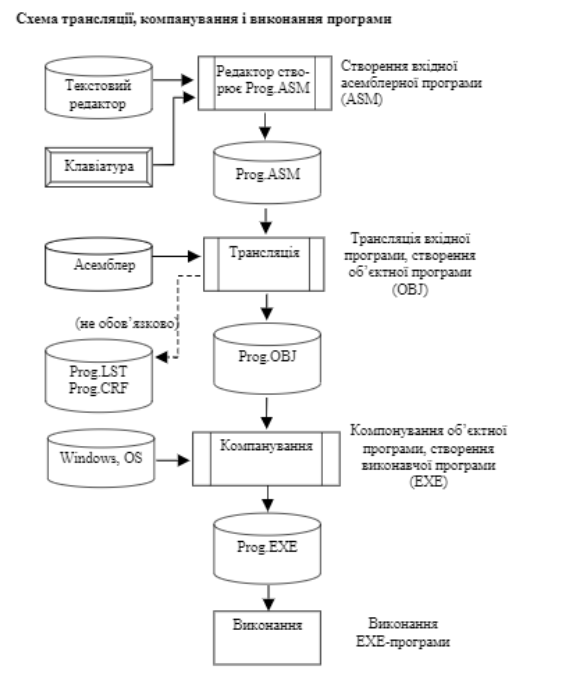
**Асемблер**

**Запитання:**

Схема трансляції, компонування і виконання програми.

**Відповідь:**

З текстового редактора або клавіатури символи потрапляють до редактора, де створюється вхідна асемблерна програма. Після цього відбувається трансляція вхідної програми та створення об’єктної програми (obj) та за потреби створюються (Prog.LST (source listing) , Prog.CRF (cross-reference)). Дальше об’єктна програма компонується та створюється виконуваний файл (exe), якщо об’єктних файлів декілька вони все одно компонуються в один exe-файл. В кінці відбувається виконання exe-програми. Схема процесу зображена на рисунку нижче.



**Бібліотеки DLL**

**Запитання:**

Стандартні типи даних Windows та їх еквіваленти в мові C.

**Відповідь:**

|  |  |  |
| --- | --- | --- |
| **Windows** | **Означення** | **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 восьми бітне без знакове ціле число | unsigned char |
| DWORD, UINT | 32-бітове без знакове 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); використовують 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 |

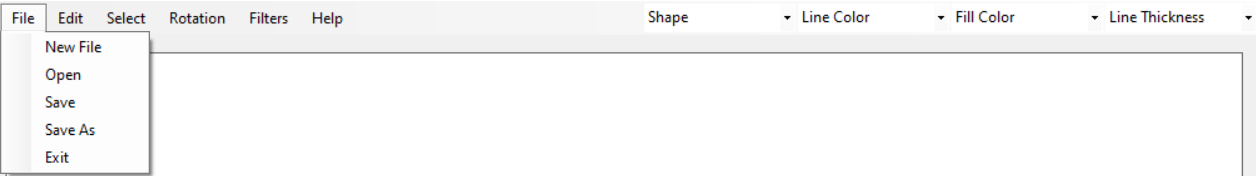
**Графічний редактор**

**Запитання:**

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.

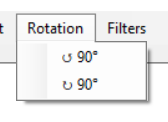
**Відповідь:**

1. Розбиття множини усіх припустимих для редактора команд на окремі логічні групи. Перемикання між групами.

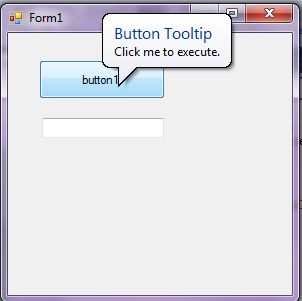


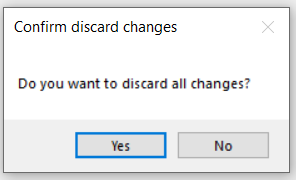
Розділення на групи відбувається переважно по функціоналу. З рисунку видно що в меню, яке реалізоване за допомогою компоненти MenuStrip з Windows Forms простір візуально розділений на 2 частини: списки команд поділені по підгрупах, що створені у вигляді випадаючого під меню, а також випадайок, що використовуються безпосередньо при малюванні.

1. Збільшення малюнка (розтягання), зменшення (стиснення), а також повороти на кут *±n(π/2)*, n=1,2,3.

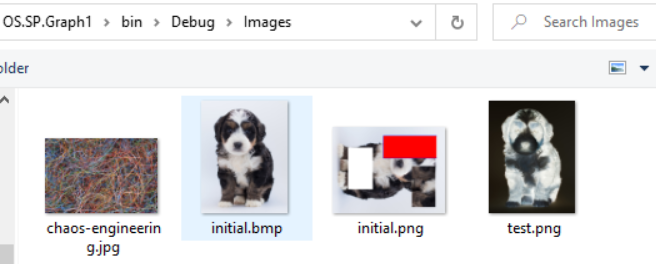


1. Масштабування малюнка з метою точного виконання графічних операцій.
2. Реалізація контекстного (наприклад, за клавішею F1) і випливаючого меню (наприклад, за правою клавішею мишки). Вибір та виконання команд через меню.
3. Підказки про події на екрані. (У Windows Forms клас ToolTip та MessageBox)

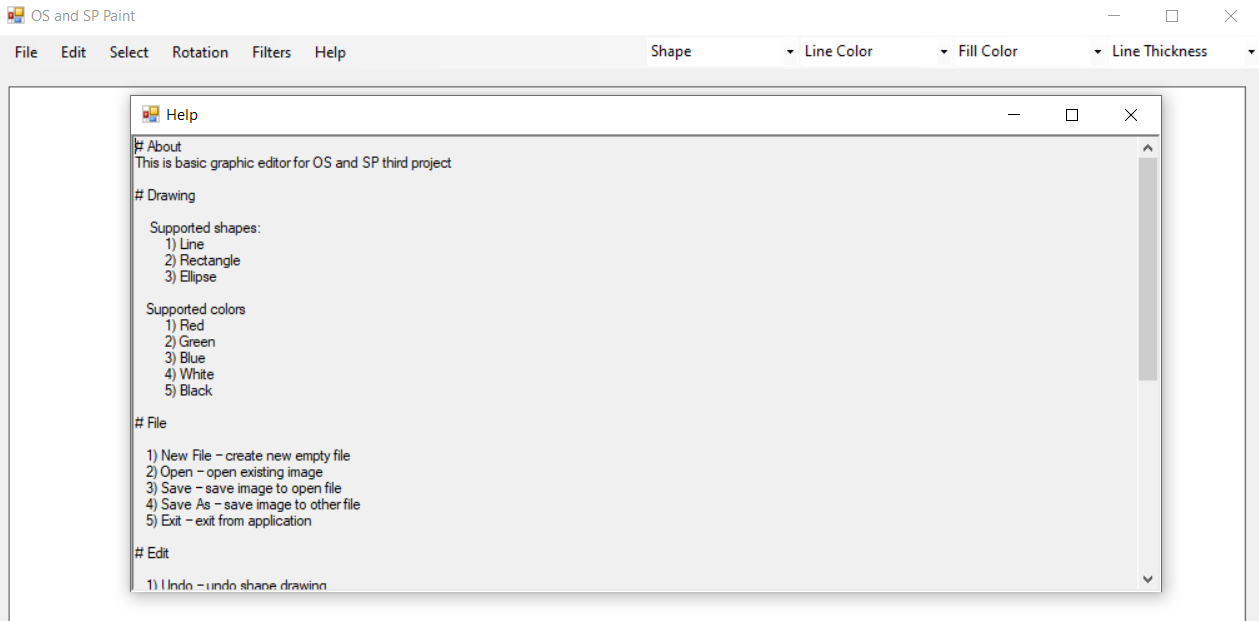




1. Згладжування ліній та інші подібні методи геометричного опрацювання.
2. Згладжування межі зміни кольору (плавний перехід одного кольору в інший) та інші способи фільтрування малюнка.
3. Формування у файлах набору демонстраційних малюнків.



1. Наявність використання. навчальної підсистеми – опис редактора, команд, особливостей. (Опис редактора можна реалізовувати за допомогою форми зі звичайним тестовим полем, або за допомогою Help.ShowHelp())



**Електронні таблиці, інтерпретатори.**

**Запитання:**

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

**Відповідь:**

Звичний для нас запис (інфіксна форма) математичних виразів має вигляд:

У такому випадку обчислення відбувається згідно звичайних математичних правил та пріоритетів. Існують також інші форми запису математичних виразів: префіксна та постфіксна. У постфіксній формі попередній приклад має вигляд:

Для обчислення виразів у такій формі в процесі інтерпретації зручно використовувати стек. Стек – структура даних, що функціонує за принципом LIFO (last in first out – останній прийшов, перший пішов).

*Алгоритм:*

Для всіх символів виконуємо такі дії:

* Якщо **Аі** число, то вкласти його у стек;
* Якщо **Аі** оператор, то:
* Витягуємо зі стека два числа;
* Виконуємо дію із числами і результат вкладаємо в стек;
* Якщо **Аі** є функцією то:
* Витягуємо зі стека одне число;
* Визначаємо значення функції із відповідним аргументом та поміщаємо результат у стек;
* В кінці роботи в стеку знаходитиметься результат виразу.

Порядок дій над ним буде такий:

|  |  |  |
| --- | --- | --- |
| **Крок** | **Елемент** | **Стек** |
| 1 | 9 | 9 |
| 2 | 3 | 3 9 |
| 3 | / | 3 |
| 4 | 2 | 2 3 |
| 5 | 4 | 4 2 3 |
| 6 | \* | 8 3 |
| 7 | 7 | 7 8 3 |
| 8 | 6 | 6 7 8 3 |
| 9 | - | 1 8 3 |
| 10 | + | 9 3 |
| 11 | \* | 27 |