МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

ІНЖЕНЕРНО-ТЕХНІЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА КОМП’ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

**КУРСОВА РОБОТА**

з дисципліни «Системне програмування**»**

на тему:

«**Програмна реалізація алгоритму сортування Шелла»**

Студента 2-го курсу

Напряму підготовки 6.050102 –

«Комп’ютерна інженерія»

Яремка Віталія

Керівник: ст.викл. Самусь Є.І

Національна шкала\_\_\_\_\_\_\_\_\_\_\_

Кількість балів:\_\_\_Оцінка:ECTS\_\_\_\_

Члени комісії: \_\_\_\_\_ ст.викл. Самусь Є.І

\_\_\_\_\_ ст.викл. Далекорей А.В

м.Ужгород – 2016 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

Інженерно-технічний факультет

Кафедра комп’ютерних систем та мереж

Напрям підготовки 6.050102 – «Комп’ютерна інженерія»

**Індивідуальне технічне завдання для виконання курсової роботи**

**з дисципліни «Програмування» студента**

­­­­­­­­­­­­­(прізвище, імя, по батькові)

**Тема:** Програмна реалізація алгоритму сортування – Shell Sort(Сортування Шелла)

**Мета:** Розробити програму на мові Асемблер та отримати необхідні навички програмування на даній мові.

**Вимоги до виконання роботи:**

1. Розробити алгоритм отриманий згідно варіанта.
2. Розробити процедури для алгоритму
   1. Зчитування даних з клавіатури та файлу;
   2. Реалізація алгоритму;
   3. Виводу результатів на екран та запису в файл;
3. Мова розробки –Асемблер.
4. Наявність користувацького інтерфейсу.
5. Передбачити перевірку коректності введених даних.
6. Передбачити перевірку коректності даних, які вводяться з клавіатури.
7. Передбачити можливість вибору початкових даних із файлу.
8. Реалізувати можливість зберігання даних у файл.

**Особливі вимоги:** Розробити меню, що забезпечує інтерактивну взаємодію користувача з комп'ютером при виборі відповідної динамічної структури (або підструктури) даних і операцій, що виконуються над нею.

**Графічна документація:** Структурні та блок-схеми алгоритмів.

**Календарний план виконання курсової роботи**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | Назва етапів курсової роботи | Строк виконання етапів роботи | Примітка |
| **1** | Вивчення предмету дослідження | 20.03.16 |  |
| **2** | Пошук та вивчення літератури з питань курсової роботи | 02.04.16 |  |
| **3** | Формування макету курсової роботи | 22.04.16 |  |
| **4** | Розробка алгоритму розв’язання задачі, вибір структур та методів | 30.04.16 |  |
| **5** | Розробка програмного забезпечення | 03.05.16 |  |
| **6** | Налагодження та тестування програми | 08.05.16 |  |
| **7** | Написання та оформлення пояснювальної записки | 20.05.16 |  |
| **8** | Попередній захист | 25.05.16 |  | |
| **9** | Захист | 03.06.16 |  | |

Студент\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище,ініціали)

Керівник­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище,ініціали)

‘\_\_’\_\_\_\_\_\_\_\_\_\_20\_\_\_р.

## РЕФЕРАТ

Пояснювальна записка цієї роботи містить: зміст, реферат, вступ, висновки, список використаних літературних джерел та 2 додатки, 46 сторінок, 14 ілюстрацій.

Мета роботи – створення програмного додатку на мові асемблер який дозволяє реалізувати запис та зчитування з файлу з використанням структур відповідно до предметної області, вказаної у технічному завданні, закріпити та узагальнити знання з набуті під час вивчення курсу, набути навичок використання основ алгоритмізації та програмування на мові асемблер.

Методи дослідження – методи теорії алгоритмів та мови програмування. У пояснювальній записці курсової роботи наведено опис обраної структури та обрано шлях розв’язання поставленої задачі. Для сортування даних використовується метод Дональда Шелла. Представлено ефективність створеної структури та роботу реалізованих методів, виявлено їх переваги та недоліки.

**ЗМІСТ**

[РЕФЕРАТ 4](#_Toc452275500)

[ВСТУП 5](#_Toc452275501)

[1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ 7](#_Toc452275502)

[2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ШЛЯХІВ РОЗВ’ЯЗУВАННЯ ЗАДАЧІ 9](#_Toc452275503)

[3 ОПИС МОВИ ПРОГРАМУВАННЯ АСЕМБЛЕР 11](#_Toc452275504)

[4 ОПИС АЛГОРИТМУ СОРТУВАННЯ ШЕЛЛА 13](#_Toc452275505)

[5 РОЗРОБКА ПРОГРАМИ 15](#_Toc452275506)

[5.1 Перелік реалізованих функцій 15](#_Toc452275507)

[5.2 Опис інтерфейсу користувача 15](#_Toc452275508)

[5.4 Тестування програми і результати її виконання 23](#_Toc452275509)

[ВИСНОВКИ 24](#_Toc452275510)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 25](#_Toc452275511)

[ДОДАТКИ 26](#_Toc452275512)

[Додаток А. Блок-схема алгоритму сортування Шелла. 26](#_Toc452275513)

[Додаток Б. Лістинг програми 27](#_Toc452275514)

## 

## ВСТУП

Метою даної курсової роботи є розробка програми яка буде сортувати масив чисел за алгоритмом сортування Шелла, особливими вимогами до виконання цієї задачі є розробка меню, яка забезпечує інтерактивну взаємодію користувача з комп'ютером. Програму необхідно розробити на мові Асемблер та отримати необхідні навички програмування на даній мові.

Мова асемблер – це мова, за допомогою якої зрозумілим для людини чином пишуться команди процесора. Насправді процесор розуміє не команди асемблера, а послідовності з нулів і одиниць. Перетворенням команд асемблера в команди процесора займається компілятор в нашому випадку це компілятор *TASM*.

Алгоритми сортування мають такі основні характеристики : час необхідний на впорядкування n-елементного масиву і додаткова пам'ять необхідна для впорядкування. Крім цих двох характеристик, сортування буває [стабільним](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) чи нестабільним, з використанням додаткової інформації про елементи, чи без використання.

Для значної кількості алгоритмів середній і найгірший час впорядкування n-елементного масиву є , це пов'язано з тим, що в них передбачені перестановки елементів, що стоять поряд. Такі алгоритми зазвичай є стабільними, хоча і не ефективними для великих масивів.

Інший клас алгоритмів здійснює впорядкування за час . В цих алгоритмах використовується можливість обміну елементів, що знаходяться на будь-якій відстані один від одного.

Ефективність сортування Шелла в певних випадках забезпечується тим, що елементи « швидше » встають на свої місця ( в простих методах сортування вставками або бульбашкою кожна перестановка двох елементів зменшує кількість інверсій в списку максимум на 1, при сортуванні Шелла це число може бути більше), але це сортування не краще, тому що все одно залишається повільним .

# 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

Потрібно розробити програму на мові Асемблер та отримати необхідні навички програмування на даній мові, до цього завдання є ряд вимог які необхідно виконати:

1 Розробити алгоритм отриманий згідно варіанта.

2 Розробити процедури для алгоритму

* 1. Зчитування даних з клавіатури та файлу;
  2. Реалізація алгоритму;
  3. Виводу результатів на екран та запису в файл;

3 Мова розробки –Асемблер.

4 Наявність користувацького інтерфейсу.

5 Передбачити перевірку коректності введених даних.

6 Передбачити перевірку коректності даних, які вводяться з клавіатури.

7 Передбачити можливість вибору початкових даних із файлу.

8 Реалізувати можливість зберігання даних у файл.

Розробити меню, що забезпечує інтерактивну взаємодію користувача з комп'ютером при виборі відповідної динамічної структури (або підструктури) даних і операцій, що виконуються над нею.

Аналізуючи вимоги до виконання курсової роботи можна сказати, що найбільш трудомісткими частинами програми будуть такі частини як: зчитування даних з файлу та клавіатури, запис результату програми у файл, виводу даних на екран, реалізація користувацького інтерфейсу.

Також потрібно виділити особливу увагу на перевірку коректності введених даних. Так як дані можна буде передавати в програму з файлу і вводити їх безпосередньо з клавіатури в самій програмі, то перевірку потрібно реалізувати як і для введених даних з клавіатури так і для даних зчитаних з файлу.

Програму не можна буде запустити без додаткових засобів на версіях *Windows* новіших за *XP*. Тому потрібно використовувати, або *DOSBox*, або віртуальну машину, або ще якісь аналогічні емулятори.

В даній курсовій роботі буде використовуватись *DOSBox*, поставлений на операційну систему *Windows 10*.

# 2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ШЛЯХІВ РОЗВ’ЯЗУВАННЯ ЗАДАЧІ

Дану задачу швидше і простіше було б реалізувати на іншій мові програмування, наприклад С++ але в поставленій задачі потрібно використовувати мову програмування асемблер, і це не дарма так як на цій мові програма буде швидше працювати так як вона працює не з пам’яттю комп’ютера, а з регістрами процесора які набагато швидше працюють із процесором а ніж пам'ять, яка знаходиться на носію, також розроблену програму легше оптимізувати для більш коректної роботи і швидкодії.

Задача побудована на мові програмування асемблер. Переважно використовують цю, а не іншу мову програмування для таких задач:

* Які вимагають максимальної швидкості виконання: основні компоненти комп'ютерних ігор, ядра операційних систем реального часу і просто критичні ділянки програм;
* Які взаємодіють із зовнішніми пристроями: драйвери, програми, що працюють безпосередньо з портами, звуковими і відео платами;
* Які використовують повністю можливості процесора: ядра багатозадачних операційних систем, DPMI-сервери і взагалі будь-які програми, які переводять процесор в захищений режим;
* Які повністю використовують можливості операційної системи: віруси і антивіруси, захисту від несанкціонованого доступу, програми, які оминають цей захист, і програми, які захищаються від цих програм;

Як виявляється, багато з того, що зазвичай пишуть на мовах високого рівня, краще, простіше і швидше написати на асемблері.

Програма сортування даних писалася з допомогою таких матеріалів:

***Turbo Assembler*** *(****TASM****)* —  [програмний пакет](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8%D0%B9_%D0%BF%D0%B0%D0%BA%D0%B5%D1%82) компанії [*Borland*](https://uk.wikipedia.org/wiki/Borland), призначений для розробки програм [мовою асемблера](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%B0%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0) для архітектури [*x*86](https://uk.wikipedia.org/wiki/X86). Крім того, *TASM* може працювати спільно з [трансляторами](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) з мов високого рівня фірми *Borland*, такими як *Turbo C* і [*Turbo Pascal*](https://uk.wikipedia.org/wiki/Turbo_Pascal).

***Notepad++*** — [текстовий редактор](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D0%B8%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80), призначений для програмістів і тих, кого не влаштовує скромна функціональність Блокнота, що входить до складу *Windows*. *Notepad++* базується на компоненті [*Scintilla*](https://uk.wikipedia.org/w/index.php?title=Scintilla&action=edit&redlink=1) (потужному компоненті для редагування), написаному на [*C++*](https://uk.wikipedia.org/wiki/C%2B%2B)з використанням тільки [*Windows API*](https://uk.wikipedia.org/wiki/Windows_API) і [*STL*](https://uk.wikipedia.org/wiki/STL), що забезпечує максимальну швидкість роботи при мінімальному розмірі програми.

Дана задача буде розглядатись в емуляторі *DOSBox*.

***DOSBox*** — емулятор програмного забезпечення, що емулює [*IBM PC* -сумісний комп'ютер](https://uk.wikipedia.org/wiki/IBM_PC-%D1%81%D1%83%D0%BC%D1%96%D1%81%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80) під управлінням старої операційної системи [*MS-DOS*](https://uk.wikipedia.org/wiki/MS-DOS). Це дає можливість запускати старі програми та ігри для [*DOS*](https://uk.wikipedia.org/wiki/MS-DOS) на сучасних комп'ютерах. *DOSBox* є [вільним програмним забезпеченням](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F), написаним в основному на [*C*](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), яке розповсюджується під [*GNU General Public License*](https://uk.wikipedia.org/wiki/GNU_General_Public_License).

Можливості такого емулятора:

1. *DOSBox* частково емулює середовище [*MS-DOS*](https://uk.wikipedia.org/wiki/MS-DOS), вектори переривань [*BIOS*](https://uk.wikipedia.org/wiki/BIOS) і апаратну частину [*IBM PC*](https://uk.wikipedia.org/wiki/IBM_PC), при цьому не потрібний ні [x86](https://uk.wikipedia.org/wiki/X86)-процесор, ні копія оригінальної *MS-DOS*.
2. Динамічне ядро ​​процесора: на системах, які мають набір команд i386, використовується динамічна трансляція інструкцій.
3. На системах, не сумісних з [x86](https://uk.wikipedia.org/wiki/X86), проводиться повна емуляція, яка веде до істотного уповільнення роботи системи.
4. Емуляція файлової системи дає можливість підключати папку хост-системи як *HDD* гостьової системи, що не доступно в жодному іншому емуляторі *PC*.

# 3 ОПИС МОВИ ПРОГРАМУВАННЯ АСЕМБЛЕР

Асемблер - це мова, за допомогою якої зрозумілим для людини чином пишуться команди процесора. Насправді процесор розуміє не команди Асемблера, а послідовності з нулів і одиниць. Перетворенням команд Асемблера в команди процесора займається компілятор (наприклад, *TASM*, *MASM, WASM*). Зазвичай одна команда Асемблера відповідає одній команді процесора, хоча не завжди - іноді для спрощення виду програм під однією командою маскується кілька. Програми, написані на Асемблері не поступаються в якості і швидкості програмами, написаним на машинній мові, так як компілятор просто переводить мнемонічні позначення команд в послідовності байтів (нулів і одиниць).

Людина не може одночасно втримувати в голові велику кількість чисел, тим більше з нулів і одиниць. Якщо б людна писала програму навіть всього з 100 команд, вона змушена була би вручну для себе позначати як-небудь ці команди, робити пояснення. Але так як ця робота нескладна, але втомлива, то краще її доручити компілятору, а програміст нехай краще займається більш корисними справами. Програми, написані на Асемблері можуть бути на порядок складнішими, ніж програми, написані на машинній мові, так як людина більше думає про алгоритм, і менше відволікається на такі дрібниці, типу як кодується команда складання регістра EAX і числа 2. Існують мови високого рівня, на яких при одному і тому ж рівні розуму і пам'яті програміста він зможе писати більш складні програми. Це в основному досягається за рахунок використання змістовних назв змінних, розбиття програми на процедури і модулі, також мова високого рівня приховує від людини багато дій, які його можуть відволікати від суті (наприклад, програміст може не задумуватись, про те що для організації циклу може вимагатися кілька команд Асемблера, він може не задумуватись, про те що дозволяє, а що не дозволяє конкретна архітектура процесора. Наприклад, немає такої команди в Асемблері, яка б могла пересилати значення з однієї комірки пам'яті в іншу, а в мовах високого рівня це взагалі не проблема). Крім того, програма на мові високого рівня зазвичай коротша, ніж відповідна програма на Асемблері, і більше піддається зміні і корекції. Але все-таки, якщо би все було так добре, не було б сенсу використовувати Асемблер. Звичайно, програма на мові високого рівня працюватиме повільніше, ніж програма на мові асемблера. Високій швидкості алгоритму при реалізації його на Асемблері можна домогтися в основному за рахунок утримання основних змінних (які використовуються найчастіше) в регістрах і використання особливостей конкретного процесора. Звернення до регістрів відбуваються швидше раз в десять, ніж звертання до комірок пам'яті, тому варто по можливості уникати зберігання основних змінних в осередках пам'яті.

# 4 ОПИС АЛГОРИТМУ СОРТУВАННЯ ШЕЛЛА

Суть методу полягає в тому, що сортуванню піддаються не всі під ряд елементи послідовності, а тільки ті які стоять один від одного на визначеній відстані *h*. На кожному кроці значення *h* змінюється, поки не стане(обов’язково) рівним 1. Важливо правильно підібрати значення *h* для кожного кроку. Від цього залежить швидкість роботи алгоритму. В якості *h* можна використовувати наступні числові послідовності: (4,2,1), (8,4,2,1) і також (7,5,3,1). Послідовності чисел можна вираховувати аналітично. За допомогою такого співвідношення :

в результаті виходить послідовність зміщень:1, 4, 13, 40, 121…

в результаті виходить послідовність зміщень:1, 3, 7, 15, 31…

Подібне аналітичне отримання послідовності зміщень дає можливість розробляти програму, яка динамічно підлаштовується під конкретний масив який буде сортуватись.

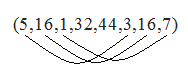
Можна відмітити, що коли *h*=1, то алгоритм сортування Шелла перетворюється в сортування прямими вставками.

**Приклад сортування**

Проілюструймо роботу алгоритму на вхідному масиві А із заданими зміщеннями *h*.

*A* = (5,16,1,32,44,3,16,7), *h* = (5,3,1).

1. Перший крок сортування.



Порівнюються числа 5 і 3, 16 і 16, 1 і 7.

Масив після впорядкування з кроком в 5: (3,16,1,32,44,5,16,7) — зроблено 1 обмін місцями помінялись числа 5 і 3.

1. Другий крок.

Порівнюються числа 7 і 44, 16 і 32, 5 і 1,44 і 16,32 і 3



Масив після впорядкування з кроком 3: (3,7,1,16,16,5,32,44) — зроблено 3 обміна місцями помінялись 44 і 7, 16 і 7, 32 і 16.

1. Третій крок.

Порівнюються всі числа як в сортуванні вставками, тобто останній крок завжди буде сортуванням вставками.



Масив після впорядкування з кроком 1: (1,3,5,7,16,16,32,44) — зроблено 5 обмінів.

Отже, весь масив впорядковано за 9 операцій обміну.

# 5 РОЗРОБКА ПРОГРАМИ

## 5.1 Перелік реалізованих функцій

– Функція зчитування даних з файлу

* Функція введення даних з клавіатури
* Функція виведення даних на екран
* Функція перевірки коректності введених даних
* Функція зберігання даних у файл
* Функція створення нового текстового файлу

## 5.2 Опис інтерфейсу користувача

Інтерфейс користувача будувався так щоб бути якнайбільш зручним і зрозумілим користувачу.

Щоб запустити програму необхідно скопіювати всі файли з носія, у робочу папку програми *DosBox* . Всі файли розробленої програми, і також файли з масивами які в подальшому будуть сортуватись програмою повинні знаходитись в одні папці, в нашому випадку в папці *Tasm 1.4 / Tasm* програми *DOSBox*.

Потім запускаємо емулятор, і вводимо наступний рядок:

*start.bat*

*=> Enter*

Після чого відкриється наступне меню (див. рисунок 1) у якому доступні наступні варіанти:

*File input* *press(F)* − читання масиву даних з файлу із розширенням *.txt*

*Console input press(C)* − введення масиву з клавіатури.

*Exit -> press(ESC)* – вихід із програми.

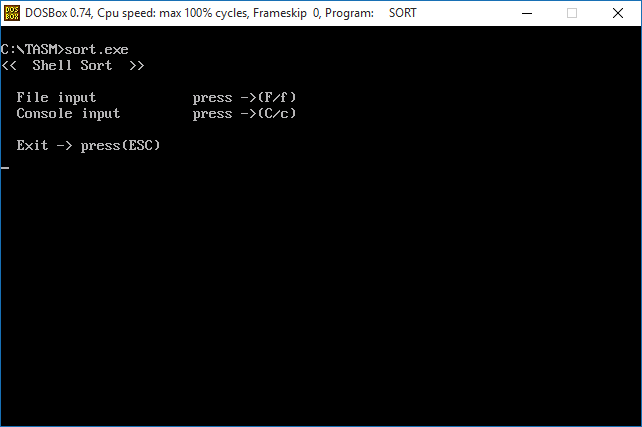


Рисунок 5.2.1 – Початкове меню програми

**Далі розглянемо різні випадки використання програми:**

1. **Випадок коли дані зчитуються з файлу**

Після вибору *File input* натисканням клавіші (*F*) даного меню, користувач повинен ввести назву файлу наприклад: *array.txt* , в якому він завчасно ввів масив чисел. В першому рядку файлу потрібно вказати який розмір матиме масив, в другому рядку безпосередньо ввести масив чисел через пробіл.

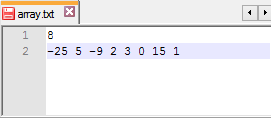
****

Рисунок 5.2.2 – Вміст файлу *array.txt*

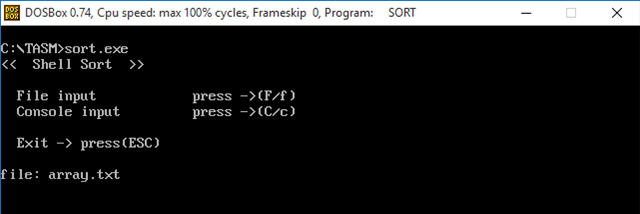


Рисунок 5.2.3 – Введення назви файлу

Після натискання клавіші Enter відбудеться виведення на екран розміру масиву, початкового масиву і відсортованого масиву.

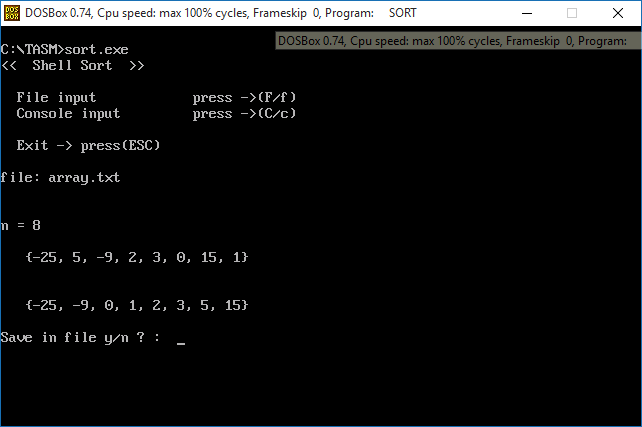


Рисунок 5.2.4 – Виведення на екран даних

Користувачу буде запропоновано зберегти відсортований масив у новому файлі, якщо потрібно зберегти масив то натисканням клавіші *Y* підтверджується зберігання масиву у файл і відкриється новий пункт в якому необхідно вказати назву файлу і розширення *.txt* , якщо не потрібно зберігати відсортований масив то натиснувши клавішу *N* – відбудеться вихід із програми. Відсортований маси буде збережено в папку в якій знаходиться дана програма.

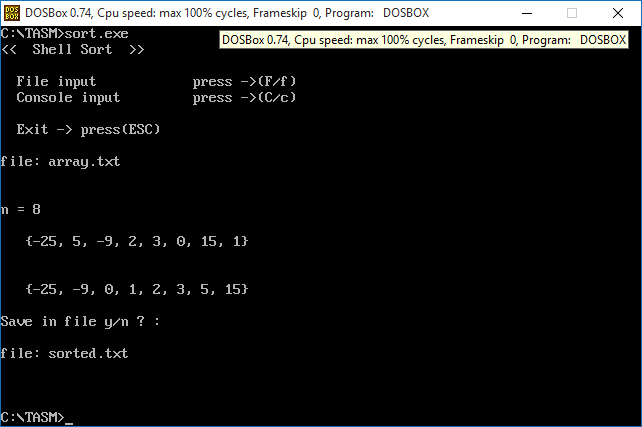


Рисунок 5.2.4 – Збереження відсортованого масиву файлу.

1. **Випадок коли дані вводяться з клавіатури**

Отже користувач вибрав пункт *Console input* , йому буде запропоновано ввести розмір масиву(див.рисунок 5).

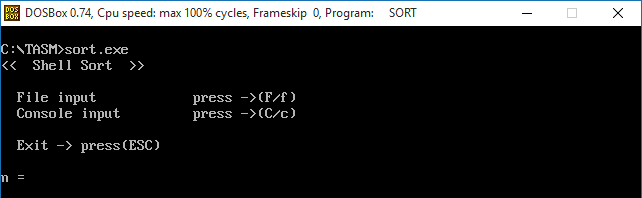


Рисунок 5.2.5 – Введення розміру масиву

Після того коли користувач ввів розмір масиву, йому потрібно буде ввести бажаний масив з клавіатури.

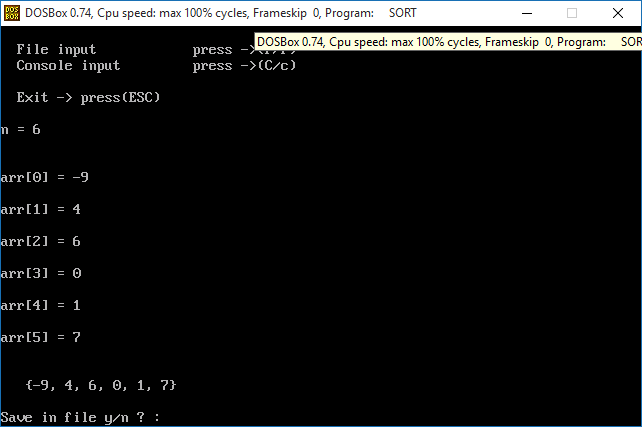


Рисунок 5.2.6 – Введення масиву

Після того коли масив був введений, він відобразиться на екрані. Користувачу буде поставлене питання чи бажає він зберегти цей масив у файл, натиснувши клавішу *Y*  потрібно буде дати назву файлу (обов’язково потрібно вказувати назву файлу з розширенням, наприклад : *save.txt*) який буде створений і в який буде записано масив, після введення назви файлу на екран виведеться відсортований масив, і користувачу знову буде запропоновано зберегти масив. В іншому випадку коли користувач натиснув клавішу *N*, на екран зразу ж виведеться відсортований масив, і буде запропоновано зберегти цей відсортований масив.

## 5.4 Тестування програми і результати її виконання

Результати роботи програми можна побачити на наступних рисунках. На рисунку 1 видно що програма чудово відсортувала масив з 40 - ка елементів. Далі ми приступим до тестування програми з від’ємними числами рисунок 5.4.3. На рисунку 5.5.4 видно що програма відмінно працює з масивом розмір якого 120 від’ємних елементів.

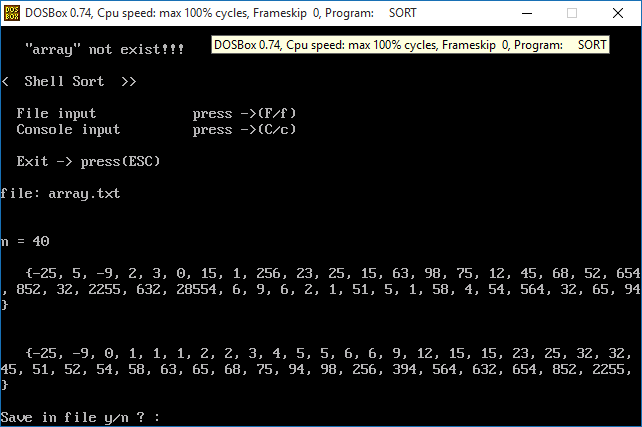


Рисунок 5.4.1 – Тестування програми при розмірі масиву 40 елементів

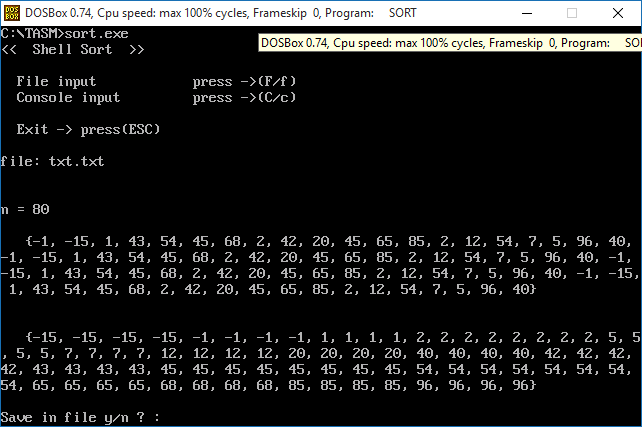


Рисунок 5.4.2 – Тестування програми при розмірі масиву 80 елементів

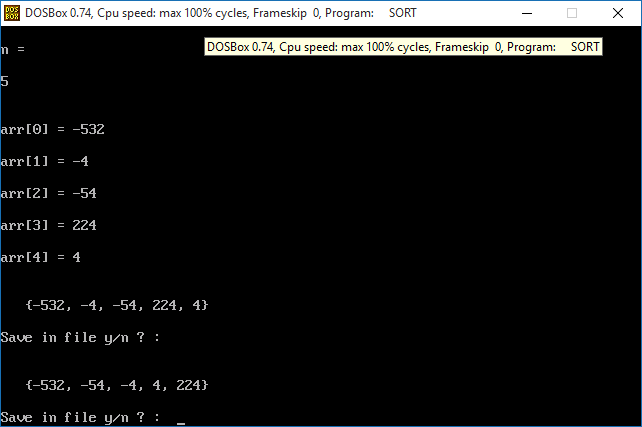


Рисунок 5.4.3 – Тестування програми з від’ємними числами

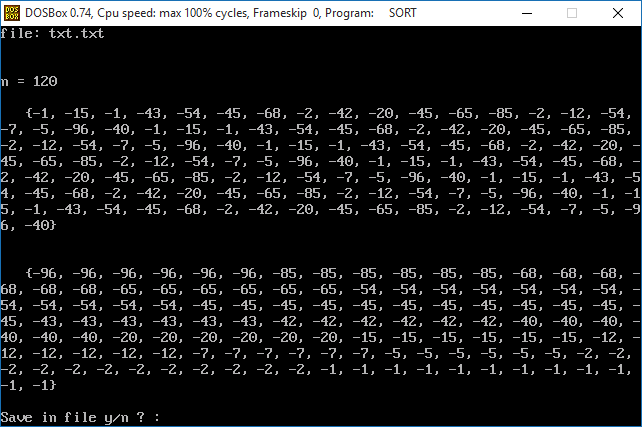


Рисунок 5.4.4 – Тестування програми з від’ємними числами при розмірі масиву 120

В програмі передбачено корекцію введення даних, тому коли користувач введе дані які б викликали некоректну роботу програми, програма не пропустить ці дані і виведе повідомлення про помилку. На рисунках 5.4.5 , 5.4.6, 5.4.7 показано спробу помилкового введення даних.

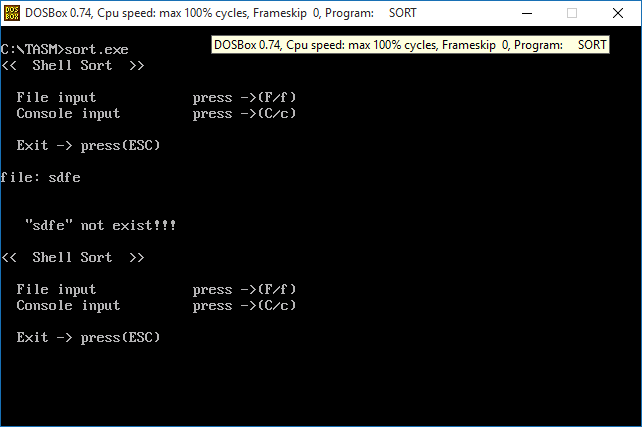


Рисунок 5.4.5 – Введення не існуючого файлу

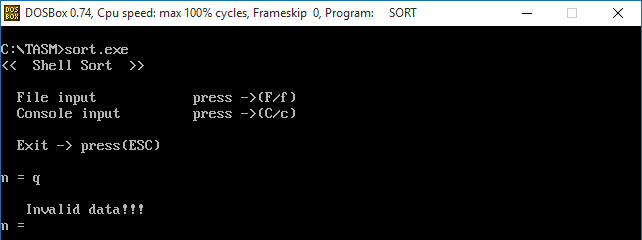


Рисунок 5.4.6 – Введення не числа в поле розміру масиву

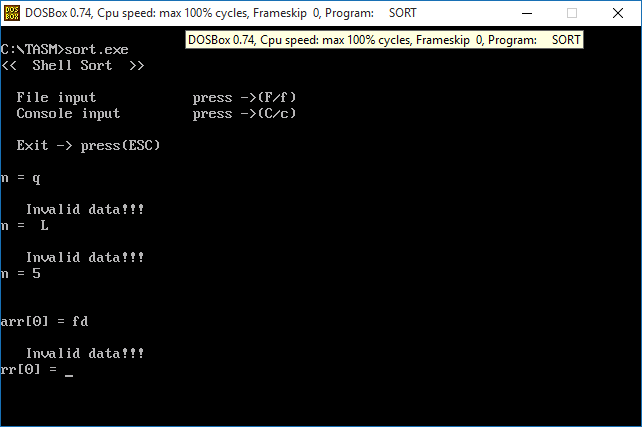


Рисунок 5.4.7 – Введення не числа в сам масив

# ВИСНОВКИ

В даній курсовій роботі було розроблено програму сортування даних. Дані сортуються за допомогою алгоритму сортування який опублікував Дональд Шелл у 1959 році. Під час виконання цієї роботи було освоєно відповідні навички програмування на низькорівневій мові програмування Асемблер, і розібрано роботу алгоритму сортування Шелла.

Метою роботи було створення програмного додатку на мові асемблер який дозволяє реалізувати запис та зчитування даних з файлу з використанням структур відповідно до предметної області, вказаної у технічному завданні, та реалізувати в даному програмному додатку перевірку коректності введених,зчитаних даних.

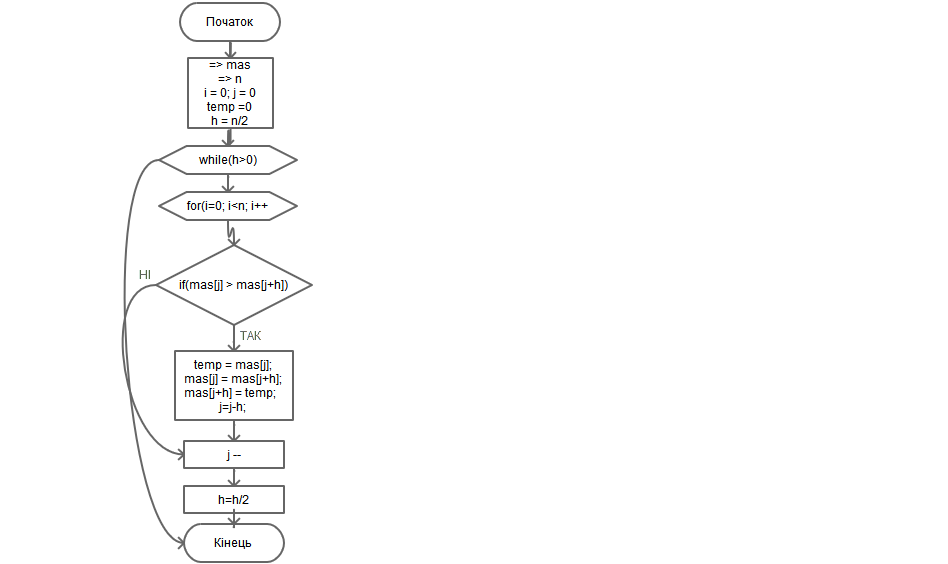
Отже, розробивши програму сортування методом Шелла можна сказати, що ефективність, алгоритму сортування Шелла, вища за деякі інші алгоритми але все одно цей алгоритм майже не використовується тому, що на даний момент вже є багато більш ефективних алгоритмів сортування, хоч алгоритм Шелла порівнює числа не попорядку, а з пеним інтервалом який щоразу зменшується він поступається сьогоденним алгоритмам. В найгіршому випадку швидкодія даного алгоритму – , в найліпшому випадку – .

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Юров В.И Assembler Учебник для вузов. 2-е изд.— СПб.: Питер, 2003. — 637 с.: ил.
2. Юров В.И Assembler. Практикум. 2-е изд. – СПб.: Питер, 2006. – 399с.:ил.
3. Рудольф Марек. Ассемблер на примерах. Базовий курс. – СПб: Наука и Техника, 2005. – 240с.:ил.
4. Львов М.С., Співаковський О. В. Основи алгоритмізації та програмування Херсон, 1997.
5. Калашников О. А. - Ассемблер — это просто. Учимся программировать. – БХВ-Петербург, 2011–336с.:ил
6. Пилыциков В. Н. – Программирование на языке ассемблера IBM PC. - М.: "ДИАЛОГ- МИФИ", 1999. - 288 с.

# ДОДАТКИ

## Додаток А. Блок-схема алгоритму сортування Шелла.



## **Додаток Б.** Лістинг програми

**Основна частина коду**

.model small

.386

.stack 3000h

.data

head db "<< Shell Sort >>", endl, 0

str\_fin db ' File input press ->(F/f)', endl, 0

str\_cin db ' Console input press ->(C/c)', endl, 0

str\_n db 'n = ', 0

str\_id db ' Invalid data!!!', endl, 0

str\_nf db ' Not found!!!', endl, 0

exit\_tab db endl, ' Exit -> press(ESC) ', endl, 0

str\_arr\_beg db 'arr[', 0

str\_arr\_end db '] = ', 0

str\_value db 'value = ', 0

file\_err db 'not exist!!!', endl, 0

prompt db 'file: ', 0

str\_save db 'Save in file y/n ? : ', 0

len dw ?

n dw 0

value dw 0

arr dw ?

handle dw ?

file\_name db 80 dup(0)

arr\_out db 80 dup(0)

tmp db 80 dup(0)

.code

LOCALS

include arr\_io.inc

;============================

cmpnum proc c

arg @a, @b

uses bx, si

mov si, @a

mov ax, word ptr[si]

mov si, @b

mov bx, word ptr[si]

cmp ax, bx

je @@ecv

jl @@less

mov ax, 1

jmp @@end

@@ecv:

xor ax, ax

jmp @@end

@@less:

mov ax, -1

@@end:

ret

cmpnum endp

;============================

;============================

exchange proc c

arg @a, @b, @n

uses ax, cx, si, di

mov cx, @n

mov si, @a

mov di, @b

@@cycle:

cmp cx, 0

je @@end

mov al, [di]

mov ah, [si]

mov [di], ah

mov [si], al

inc si

inc di

dec cx

jmp @@cycle

@@end:

ret

exchange endp

;============================

;============================

shell\_sort proc c

arg @arr, @arr\_len, @size\_elem, @comparator

uses ax, bx, cx, dx, si, di

local @d, @i, @j

mov di, @arr\_len

mov bx, @arr

@@cycle1:

shr di, 1

cmp di, 0

je @@end

mov cx, di

@@cycle2:

cmp cx, @arr\_len

je @@ns0

mov dx, cx

@@cycle3:

cmp dx, di

jl @@ns1

mov ax, dx

sub ax, di

mov si, bx

add si, ax

add si, ax

push si

mov si, bx

add si, dx

add si, dx

push si

call @comparator

add sp, 4

cmp ax, 0

jge @@ns1

push @size\_elem

push si

sub si, di

sub si, di

push si

call exchange

add sp, 6

sub dx, di

jmp @@cycle3

@@ns1:

inc cx

jmp @@cycle2

@@ns0:

jmp @@cycle1

@@end:

ret

shell\_sort endp

;============================

;============================

file\_save proc c

uses ax

puts str\_save

@@fsave:

call \_getch

cmp al, 'y'

je @@save

cmp al, 'Y'

je @@save

cmp al, 'n'

je @@end

cmp al, 'N'

je @@end

jmp @@fsave

@@save:

putc endl

putc endl

puts prompt

push offset file\_name

call gets

add sp, 2

push WO

push offset file\_name

call fopen

add sp, 4

mov handle, ax

push 10

push offset tmp

push n

call itoa

add sp, 6

push handle

push offset tmp

call fputs

add sp, 4

fputc endl, handle

push handle

push n

push arr

call arr\_ifout

add sp, 6

push handle

call fclose

add sp, 2

@@end:

putc endl

putc endl

ret

file\_save endp

;============================

;============================

main proc

mov ax, @data

mov ds, ax

@@beg\_app:

; menu

puts head

putc endl

puts str\_fin

puts str\_cin

puts exit\_tab

@@check\_input:

call \_getch

cmp al, 31h

je @@fin

cmp al, 'f'

je @@fin

cmp al, 'F'

je @@fin

cmp al, 32h

je @@cin

cmp al, 'c'

je @@cin

cmp al, 'C'

je @@cin

cmp ah, 1

jne @@check\_input

\_exit

; file input

@@fin:

putc endl

puts prompt

push offset file\_name

call gets

add sp, 2

putc endl

putc endl

push RO

push offset file\_name

call fopen

add sp, 4

test ax, ax

jne @@ns2

putc ' '

putc ' '

putc ' '

putc '"'

puts file\_name

putc '"'

putc ' '

puts file\_err

putc endl

jmp @@beg\_app

@@ns2:

mov handle, ax

push offset tmp

push handle

call ifin

add sp, 4

cmp bl, 0

je @@ns3 ; file open

; error

puts str\_id

jmp @@beg\_app

@@ns3:

mov n, ax

add ax, ax

mov len, ax

sub sp, ax

mov arr, sp

puts str\_n

puts tmp

putc endl

putc endl

push handle

push n

push arr

call arr\_ifin

add sp, 6

push ax

putc endl

pop ax

cmp al, 0

je @@alg

; error

putc endl

puts str\_id

; memory free

add sp, len

; goto @@beg\_app

jmp @@beg\_app

; console input

@@cin:

putc endl

push offset str\_n

call icin

add sp, 2

mov n, ax

; reserved memory

add ax, ax

mov len, ax

sub sp, ax

mov arr, sp

putc endl

; input arr

push n

push arr

call arr\_icin

add sp, 4

putc endl

push n

push arr

call arr\_icout

add sp, 4

putc endl

call file\_save

@@alg:

putc endl

push offset cmpnum

push 2

push n

push arr

call shell\_sort

add sp, 8

push n

push arr

call arr\_icout

add sp, 4

putc endl

call file\_save

@@end:

; memory free

add sp, n

add sp, n

@@end\_app:

\_exit

main endp

;============================

end main

**Модуль введення виведення масиву.**

; module for input/output array

include io.inc

;============================

str\_arr proc c

arg @str, @num, @beg, @end

uses ax

push @beg

push @str

call strcpy

add sp, 4

push 10

push offset tmp

push @num

call itoa

add sp, 6

push offset tmp

push @str

call strcat

add sp, 4

push @end

push @str

call strcat

add sp, 4

ret

str\_arr endp

;============================

;============================

arr\_icin proc c

arg @arr, @n

uses bx, cx

mov bx, @arr

xor cx, cx

@@cycle:

cmp cx, @n

je @@end

push offset str\_arr\_end

push offset str\_arr\_beg

push cx

push offset arr\_out

call str\_arr

add sp, 8

push offset arr\_out

call icin

add sp, 2

mov word ptr[bx], ax

add bx, 2

inc cx

jmp @@cycle

@@end:

ret

arr\_icin endp

;============================

;============================

arr\_ifin proc c

arg @arr, @n, @handle

uses bx, cx, dx, si

sub sp, 80

mov dx, sp

mov si, @arr

xor cx, cx

push dx

push @handle

putc ' '

putc ' '

putc ' '

putc '{'

@@cycle:

cmp cx, @n

je @@end

call ifin

cmp bl, 0

je @@ns

putc endl

add sp, 84

mov ax, 1

ret

@@ns:

mov [si], ax

puts dx

putc ','

putc ' '

add si, 2

inc cx

jmp @@cycle

@@end:

add sp, 84

putc 8

putc 8

putc '}'

putc endl

xor ax, ax

ret

arr\_ifin endp

;============================

;============================

arr\_icout proc c

arg @arr, @n

uses ax, bx, cx, dx

putc ' '

putc ' '

putc ' '

putc '{'

sub sp, 10

mov dx, sp

mov bx, @arr

mov cx, @n

@@cycle:

cmp cx, 0

je @@end

push 10

push dx

push word ptr[bx]

call itoa

add sp, 6

puts dx

putc ','

putc ' '

add bx, 2

dec cx

jmp @@cycle

@@end:

add sp, 10

putc 08h

putc 08h

putc '}'

putc endl

ret

arr\_icout endp

;============================

;============================

arr\_ifout proc c

arg @arr, @n, @handle

uses ax, bx, cx, dx

sub sp, 10

mov dx, sp

xor cx, cx

mov bx, @arr

push 10

push dx

@@cycle:

cmp cx, @n

je @@end

push word ptr[bx]

call itoa

add sp, 2

push @handle

push dx

call fputs

add sp, 4

fputc ' ', @handle

@@ns:

inc cx

add bx, 2

jmp @@cycle

@@end:

fputc endl, @handle

ret

arr\_ifout endp

;============================

**Підпрограма даного модуля**

; io

include src\stdio.inc

include src\ctype.inc

;============================

ifin proc c

arg @handle, @str

uses cx

push @handle

@@cycle1:

call fgetc

push ax

call isspace

mov bx, ax

pop ax

cmp bx, 0

jne @@cycle1

mov bx, @str

mov [bx], al

inc bx

@@cycle2:

call fgetc

push ax

call isspace

mov cx, ax

pop ax

cmp cx, 0

jne @@end\_fin

mov [bx], al

inc bx

jmp @@cycle2

@@end\_fin:

mov byte ptr[bx], 0

push @str

call atoi

add sp, 4

ret

ifin endp

;============================

;============================

icin proc c

arg @s

uses bx, dx

sub sp, 80

mov dx, sp

mov dx, sp

push dx

@@cycle:

mov bx, @s

puts bx

call gets

putc endl

call strlen

cmp ax, 0

je @@err

call atoi

cmp bl, 0

je @@end

@@err:

puts str\_id

jmp @@cycle

@@end:

add sp, 82

ret

icin endp

;============================

**Модуль контролю коректності введения даних**

; libasm - partial implementation of libc in tasm

; uses CDECL as calling convention

;int isupper(int c)

;=============================================

; if(c >= 'A' && c <= 'Z') ax = true;

; else ax = false;

;=============================================

isupper proc c

arg @c

mov ax, @c

cmp al, 'A'

jb @@false

cmp al, 'Z'

ja @@false

mov ax, 1

ret

@@false:

xor ax, ax

ret

isupper endp

;int islower(int c)

;=============================================

; if(c >= 'a' && c <= 'z') ax = true;

; else ax = false;

;=============================================

islower proc c

arg @c

mov ax, @c

cmp al, 'a'

jb @@false

cmp al, 'z'

ja @@false

mov ax, 1

ret

@@false:

xor ax, ax

ret

islower endp

;int isdigit(int c)

;=============================================

; if(c >= '0' && c <= '9') ax = true;

; else ax = false;

;=============================================

isdigit proc c

arg @c

mov ax, @c

cmp al, '0'

jb @@false

cmp al, '9'

ja @@false

mov ax, 1

ret

@@false:

xor ax, ax

ret

isdigit endp

;int isxdigit(int c)

;=============================================

; if((c >= '0' && c <= '9') ||(c >= 'A' && c <= 'F') ||

; (c >= 'a' && c <= 'f') ax = true;

; else ax = false;

;=============================================

isxdigit proc c

arg @c

mov ax, @c

cmp al, '0'

jb @@false

cmp al, '9'

jbe @@true

cmp al, 'A'

jb @@false

cmp al, 'F'

jbe @@true

cmp al, 'a'

jb @@false

cmp al, 'f'

ja @@false

@@true:

mov ax, 1

ret

@@false:

xor ax, ax

ret

isxdigit endp

;int isalpha(int c)

;=============================================

; if(isupper(c) || islover(c)) ax = true;

; else ax = false;

;=============================================

isalpha proc c

arg @c

push @c

call isupper

cmp ax, 0

jnz @@return

call islower

@@return:

add sp, 2

ret

isalpha endp

; int isalnum(int c) -> ax

;=============================================

; if(isalpha(c) || isdigit(c)) ax = true;

; else ax = false;

;=============================================

isalnum proc c

arg @c

push @c

call isalpha

cmp ax, 0

jnz @@return

call isdigit

@@return:

add sp, 2

ret

isalnum endp

;int isspace(int c)

;=============================================

; if(c == '\v' || c == '\n' || c == '\t' ||

; c == '\r' || c == '\f' || c == ' ') ax = true;

; else ax = false;

;=============================================

isspace proc c

arg @c

mov ax, @c

cmp al, 9h

jb @@false

cmp al, 0dh

jbe @@true

cmp al, ' '

jne @@false

@@true:

mov ax, 1

ret

@@false:

xor ax, ax

ret

isspace endp

;int iscntrl(int c)

;=============================================

; if(c <= 1fh) ax = true;

; else ax = false;

;=============================================

iscntrl proc c

arg @c

mov ax, @c

cmp al, 1fh

ja @@false

mov ax, 1

ret

@@false:

xor ax, ax

ret

iscntrl endp

;int ispunct(int c)

;=============================================

; if((c == {!"#$%&'()\*+,=./:;<=>?@[]^\_`{|}~) ax = true;

; else ax = false;

;=============================================

ispunct proc c

arg @c

mov ax, @c

cmp al, 21h

jb @@false

cmp al, 2fh

jbe @@true

cmp al, 3ah

jb @@false

cmp al, 40h

jbe @@true

cmp al, 5bh

jb @@false

cmp al, 60h

jbe @@true

cmp al, 7bh

jb @@false

cmp al, 7eh

ja @@false

@@true:

mov ax, 1

ret

@@false:

xor ax, ax

ret

ispunct endp

; int isgraph(int c) -> ax

;=============================================

; if(c >= '!' && c <= '~') ax = true;

; else ax = false;

;=============================================

isgraph proc c

arg @c

mov ax, @c

cmp al, '!'

jb @@false

cmp al, '~'

ja @@false

mov ax, 1

ret

@@false:

xor ax, ax

ret

isgraph endp

; int isprint(int c) -> ax

;=============================================

; if(c >= ' ' && c <= '~') ax = true;

; else ax = false;

;=============================================

isprint proc c

arg @c

mov ax, @c

cmp al, ' '

jb @@false

cmp al, '~'

ja @@false

mov ax, 1

ret

@@false:

xor ax, ax

ret

isprint endp

; char toupper(char c) -> al

toupper macro c

mov ax, c

cmp al, 'a'

jb @@end

cmp al, 'z'

ja @@end

sub al, 32

@@end:

endm

; char tolower(char c) -> al

tolower macro c

mov ax, c

cmp al, 'A'

jb @@end

cmp al, 'Z'

ja @@end

add al, 32

@@end:

endm

; int isblank(int ch)

;=============================================

; C99

; if(isspace(c)) ax = true;

; else ax = false;

;=============================================

isblank proc c

arg @c

push @c

call isspace

add sp, 2

ret

isblank endp

; char \*\_toupper(char \*s) -> ax

\_toupper proc c

uses bx, dx

arg @s

mov bx, @s

@@beg:

mov dl, [bx]

toupper dx

add sp, 2

mov [bx], al

inc bx

cmp byte ptr[bx], 0

jne @@beg

mov ax, @s

ret

\_toupper endp

; char \*\_tolower(char \*s) -> ax

\_tolower proc c

uses bx

arg @s

mov bx, @s

@@beg:

mov al, [bx]

tolower ax

mov [bx], al

inc bx

cmp byte ptr[bx], 0

jne @@beg

mov ax, @s

ret

\_tolower endp