

Технології розробки алгоритмів розв'язання інженерних задач

Лекція №2

Викладач: Дрєєв Олександр Миколайович

2. Правила оформлення програми:

2.1. Структурна, функціональна, блок- схеми.

Принцип написання від коментарів до програми.

2.2. Оформлення програми для зручності читання. Вирівнювання тексту. Уособлення функцій, умов, циклів.

2.3. Принцип універсальності використання. Потоки вводу-виводу, стандартизація.

Правила оформлення програми

Причини введення правил оформлення

Програми пишуться на РОКИ — основна частина в роботі програміста є не створення програмного продукту, а його вдосконалення та супроводження. Через місяць Ваш код прирівнюється до чужого, але помилки виправляти потрібно, потрібно доповнювати функціональність, тощо.

Правила оформлення програми

Структурна, функціональна, блок- схеми.

Постанова задачі:

1. Вхідні данні та способи їх отримання
 2. Вихідні данні та способи їх виведення
 3. Вимоги до часу виконання
 4. Вимоги до використання пам'яті
 5. Вимоги до розміру програми
 6. Строки виконання
 7. Методика перевірки функціональності
 8. Доведення правильності
-
-

Правила оформлення програми

Структурна схема

На прикладі сортування вибіркою

З ЧОГО СКЛАДАЄТЬСЯ

Для програми виділяють:

1. Блок отримання масиву
 2. Блок звернення до елементів масиву
 3. Блок отримання найбільшого елемента в вказаному діапазоні
 4. Блок формування під діапазонів
 5. Блок перестановки елементів
 6. Блок виводу результату
-
-

Правила оформлення програми

Функціональна схема

Будується функціональна взаємодія:

ЯК ПРАЦЮЄ

1. Цикл підготовки масиву
 2. Цикл визначення підмасивів, видає перший номер
 3. Цикл пошуку номеру максимального елементу
 4. Зміна елементів місцями
 5. Цикл виведення результату
-
-

Введення до предмету

Схема потоків інформації

ВИДІЛЕННЯ ПОТОКІВ

1. Потік введення
 2. Потік індексів початку пошуку
максимального
 3. Потік значень елементів для пошуку
максимального значення
 4. Потік індексів максимальних елементів
 5. Потоки переміни місцями елементів
 6. Потік виведення
- Визначення інтенсивних потоків
-
-

Правила оформлення програми

Аналіз алгоритму

За структурою: максималізація відокремлення блоків керування інформацією та контейнерами інформації

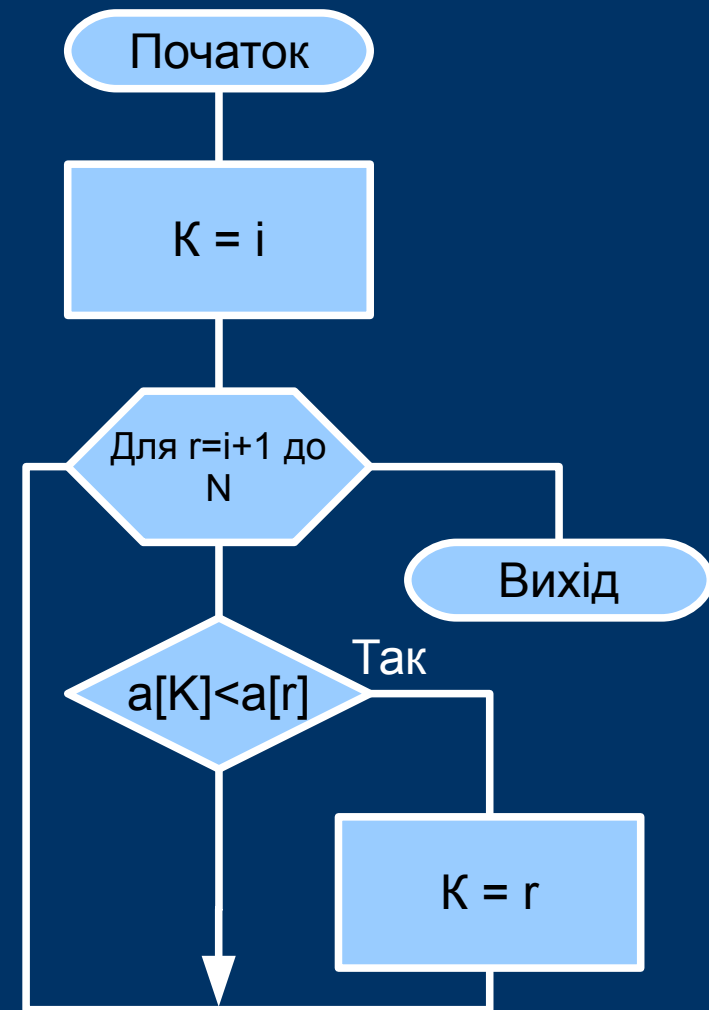
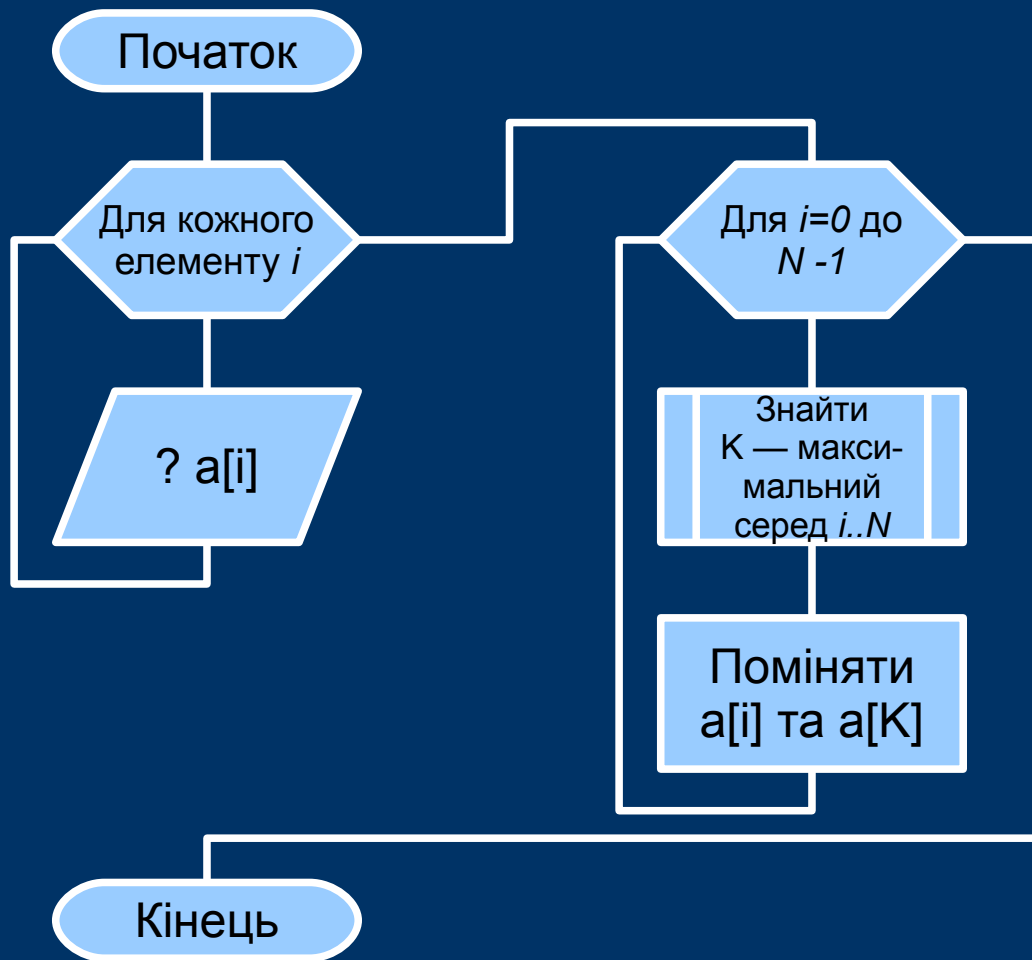
За функціональністю: мінімалізація кількості взаємодійних зв'язків модулів програми без врахування руху інформації

За потоками інформації: мінімілізація руху інформації

Корекція схем

Правила оформлення програми

Створення блок-схеми алгоритму



Правила оформлення програми

Принцип написання від коментарів до коду

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

```
/* Функція для переданого масиву А починаючи з номера і  
 * шукає максимальний елемент та повертає його номер */
```

```
int findMaxFromI(int* A, int start, int length);
```

```
int main(int N, char* Param[])
```

```
{ /*Опис масиву та змінна для його довжини*/
```

```
    int Massiv[256], k;
```

```
    int i, Tmp, M; /*Лічильник, тимчасова, номер max */
```

```
    /*Цикл введення масиву до кінця файлу або до k=256 */
```

```
    while( !feof(stdin) && k<256 )
```

```
    {
```

```
        scanf("%i", &Massiv[k]);
```

```
        k++;
```

```
    }
```

Правила оформлення програми

Принцип універсальності використання, потоки

```
/* Для кожного елементу введеного масиву виконати:*/  
for (i=0 ; i<k ; i++)  
{ /* Визначимо номер максимального */  
    M=findMaxFromI (Massiv, i) ;  
    /* Поміняємо максимальний з теперішнім елементом */  
    Tmp=Massiv [M] ;  
    Massiv [M] = Massiv [i] ;  
    Massiv [i] = Tmp ;  
} /* Масив відсортовано, приступимо до виводу */  
for (i=0 ; i<k ; i++)  
{  
    printf ("%i\n", Massiv [i]) ;  
}  
}
```

Правила оформлення програми

Оформлення програми для зручності

```
int findMaxFromI(int* A, int start, int length)
{
    int i,M;
    M = start; /*Перший елемент вважаємо найбільшим*/
    for(i=start; i<length; i++)
    { /*Для кожного з дальших елементів перевіримо, чи він
       *більший*/
        if( A[i]>A[M] )
        { /*Якщо дійсно більший, то змінимо номер найбільшого*/
            M = i;
        }
    }
    return M; /*Повертаємо значення найбільшого елементу*/
}
```

Основні принципи проектування алгоритмів

Лекція №3

3. Основні принципи проектування алгоритмів.

3.1. Принцип виконання алгоритму на процесорі. Приклад.

3.2. Метод покрокової деталізації.

3.3. Запис алгоритму.

3.4. Числа, посилання, масиви, структури.

Основні принципи проектування алгоритмів

Принцип виконання алгоритму на процесорі

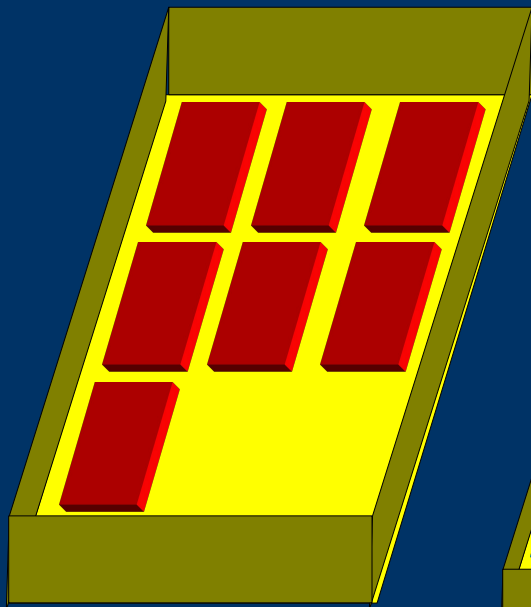
Приклад. Пошук менших.

1. Вхідні данні: коробка з перевернутими картками, на яких написано число.
 2. Вихідні дані: Картки з числом меншим за число на першій картці
 3. Умови виконання: однією рукою виконавець може дістати або повернути картку в коробку, виконавець може побачити число на картці лише на килимку-“пристрої виконання” де вміщується лише дві картки.
-
-

Основні принципи проектування алгоритмів

Принцип виконання алгоритму на процесорі

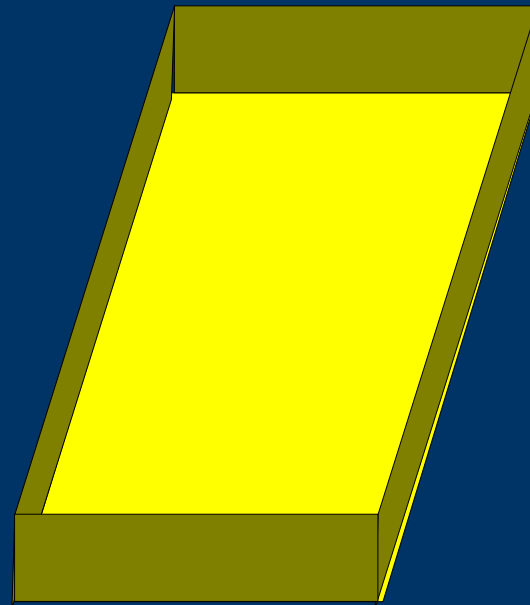
Приклад. Пошук менших.



ОЗП
Пам'ять



ЦП
Регістри+
логічний пр.



ПЗП
Пам'ять

Основні принципи проектування алгоритмів

Покрокова деталізація

Покрокова деталізація:

1. Огляд задачі в цілому
2. Визначення, “що я повинен вміти, щоб розв’язати цю задачу”.
3. Побудова алгоритму для розв’язання основної задачі
4. Повторити розв’язання для поставлених підзадач у пункті 2.

ПРИКЛАД: Сортування по спаданню.

Основні принципи проектування алгоритмів

Покрокова деталізація

Задача: Відсортувати масив.

1. Нехай у масиві елементи $0..N-1$ вже відсортовані від більшого до меншого. Як додати до такого масиву ще один елемент?
 2. Є загальний масив. Перший елемент є впорядкованим масивом. До нього можна додати наступний, але так, щоб масив з новим елементом зберіг сортування. Дію повторювати і сортований масив почне збільшуватись.
-
-

Основні принципи проектування алгоритмів

Покрокова деталізація

3. Алгоритм:

- а) Перший елемент вважаємо початковим відсортованим підмасивом
- б) Наступний елемент додаємо до підмасиву зі збереженням сортованості
- г) Поки є не сортовані елементи, переходимо до пункту б)

Підзадача: останній елемент відсортованого масиву не сортований, пересунути його на своє місце

Основні принципи проектування алгоритмів

Покрокова деталізація

4. Підзадача:

Останній елемент відсортованого масиву не сортований, пересунути його на своє місце.

а) Якщо попередній елемент від не сортованого менший, то поміняти їх місцями.

б) Повторити пункт а) доки не досягнемо початку масиву, або не було обміну місцями

5	3	2	4
5	3	4	2
5	4	3	2

Основні принципи проектування алгоритмів

Запис алгоритму

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

```
void addElement(int* A, int length);
```

```
void main()
```

```
{
```

```
    int M[256], lengthM, i;
```

```
    ... /* Прочитати масив */
```

```
    for( i=1; i<lengthM; i++)
```

```
    { /*Збільшення відсортованої частини*/
```

```
        addElement(M,i)
```

```
    }
```

```
}
```

Основні принципи проектування алгоритмів


Запис алгоритму

```
/*Збільшення відсортованої частини*/  
void addElement(int* A, int length)  
{  
    int k, tmp;  
    for( k=length; k>0; k--)  
    { /*З кінця підмасива до його початку:*/  
        if( A[k]>A[k-1] ) /*Якщо наш елемент більший*/  
        { /*Зміна елементів місцями*/  
            tmp = A[k]; A[k] = A[k-1]; A[k-1] = tmp;  
        } else  
        { /*Елемент вже на місці, вертаємося*/  
            return;  
        }  
    }  
}
```

Основні принципи проектування алгоритмів

Числа, посилання, масиви, структури

- **Число** — значення яке зберігається в пам'яті комп'ютера.
- **Посилання** — значення адреси пам'яті за якою записані дані (число).



№	0	1	2	3	4	5	6	7	8
Зн.	0	6	?	?	?	?	48	?	?

Основні принципи проектування алгоритмів

Числа, посилання, масиви, структури

- **Масив** — значення які зберігаються в пам'яті у послідовних адресах.
- **Структура** — декілька значень які записані послідовно у визначеному порядку.

№	0	1	2	3	4	5	6	7	8
Зн.	0	?	34	23	22	1	48	?	?