

Шаблони для ведення лекцій за презентаціями (підготовлені для друку)

2. Алгоритм побудови недетермінованого скінченного автомата за регулярним виразом

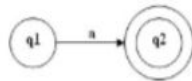
$$L = L(R_0) \quad L = L(M_0)$$

$$R_0 = e \quad M_0 = (\{q_0\}, \Sigma, \emptyset, q_0, \{q_0\}).$$



$$R_0 = a, a \in \Sigma \quad M_0 = (\{q_1, q_2\}, \Sigma, \delta_0, q_1, \{q_2\}).$$

$$\delta_0(q_1, a) = \{q_2\}$$



$$R_1 : M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$

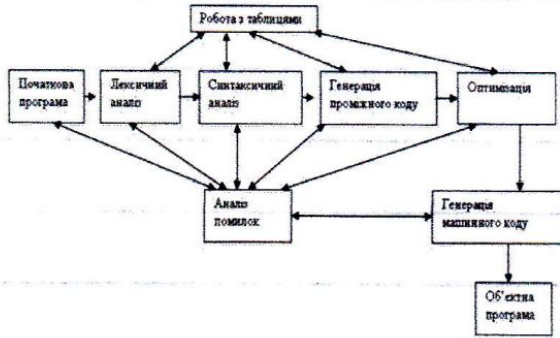
$$R_2 : M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

$$Q_1 \cap Q_2 = \emptyset$$

$$R_0 = R_1 + R_2 \quad (R_0 = R_1 | R_2) \quad R_0 = R_1 R_2 \quad R_0 = R_1^* \quad R_0 = R_1^+$$

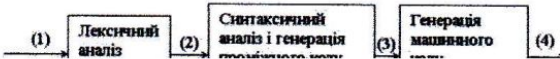
Приклад конспекту лекцій за презентаціями

2.6. Аналіз помилок компіляції та генерація машинного коду



Виконують всі такі задачі: Вибір

2.7. Взаємодія етапів компіляції, проходи компілятора.



Діт час роботи всім етапам роботи мовного процесора в 75- анкеті кожного

1. Якість лексичного аналізу (можливо утворити лексему)
2. Якість аналізу дугаєвості код та синтаксичного аналізу
3. Можливо виконати код (опис джерела не відбувся (приклад))

Бажано видавати всі помилки із поясненнями; рекомендаціями

іноді відбувається зрив машинного коду в об'єкті коду; коли неможливо отримати проміжний код.

наприклад, керує в для обчислення, ...

Моделі мовного процесора

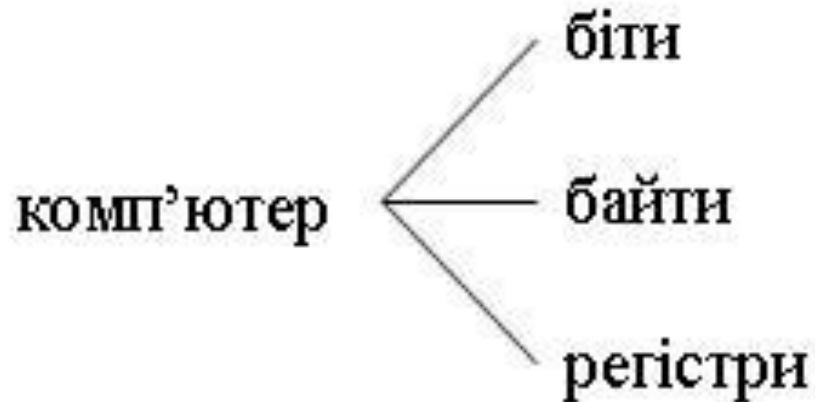


на різних прог. різних етапів вик. на кожному і на різних етапах виконання

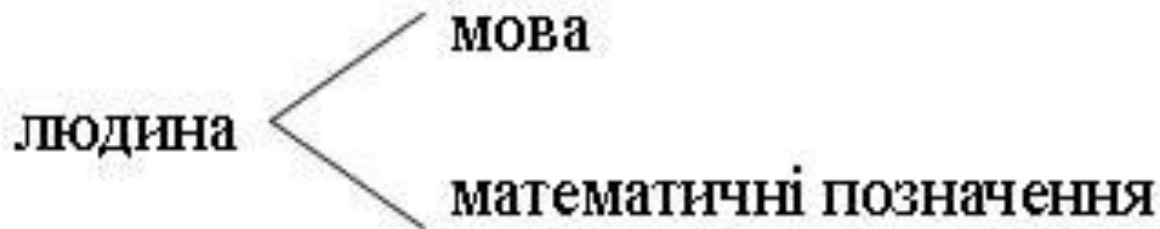
Тема 1: Розробка мовних процесорів мов програмування

- 1. Поняття мовного процесора, типи мовних процесорів.**
- 2. Основні фази мовного процесора, спрощена модель компілятора.**
 - 2.1. лексичний аналіз програм на мові високого рівня**
 - 2.2. робота з таблицями (хеш-таблиці)**
 - 2.3. синтаксичний аналіз програми**
 - 2.4. генерація проміжного коду**
 - 2.5. оптимізація проміжного коду**
 - 2.6. аналіз помилок компіляції та генерація машинного коду**
 - 2.7. взаємодія етапів компіляції, проходи компілятора.**

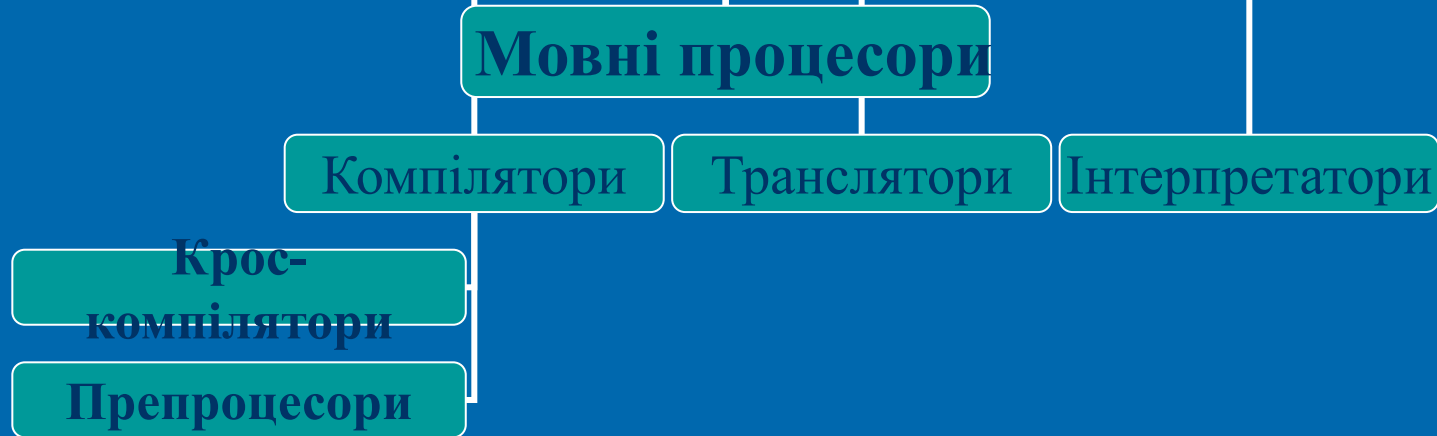
1. Поняття мовного процесора, ТИПИ МОВНИХ ПРОЦЕСОРІВ



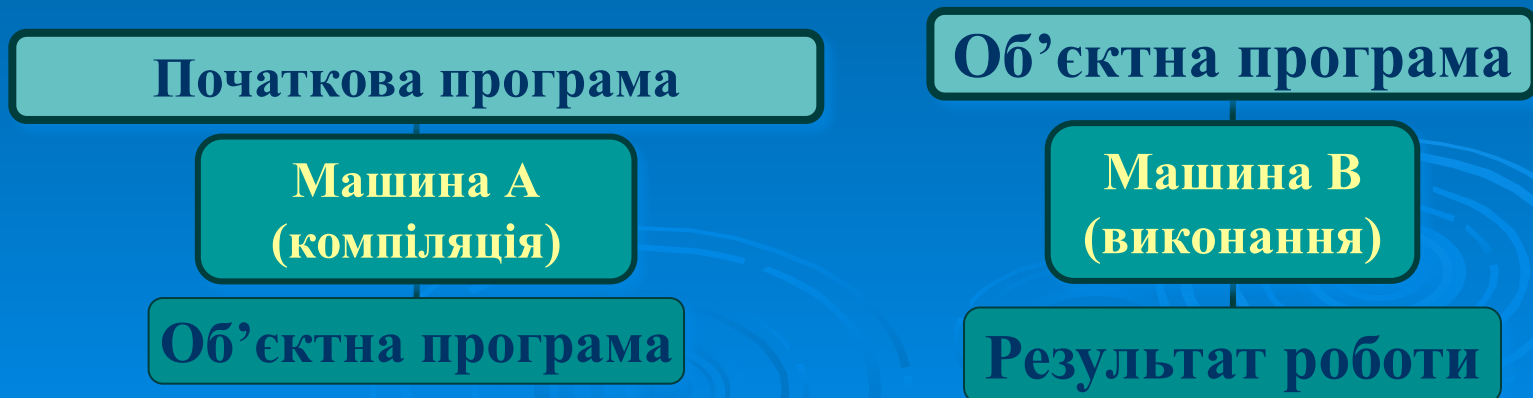
штучні мови



Типи мовних процесорів



Етапи виконання програми



2. Основні фази мовного процесора, спрощена модель компілятора.

Основні фази: лексичний аналіз; побудова таблиць; синтаксичний аналіз; генерація проміжного коду; оптимізація; генерація машинного коду

2.1. Лексичний аналіз програм на мові високого рівня

Приклад 1:

`cost := (price + tax)*0,98 (1)`

Позначення:

- ідентифікатори <ід>
- дійсне число <дч>
- присвоєння <пр>

`<ід, р1><пр>(<ід, р2> + <ід, р3>) * <дч, q1>`

2.2. Робота з таблицями (хеш-таблиці)

Integer cost, tax, price

$\text{cost} := (\text{price} + \text{tax}) * 0,98 \quad (1)$

Номер	Ідентифікатор	Значення
1	cost	p1
2	tax	p2
3	price	p3

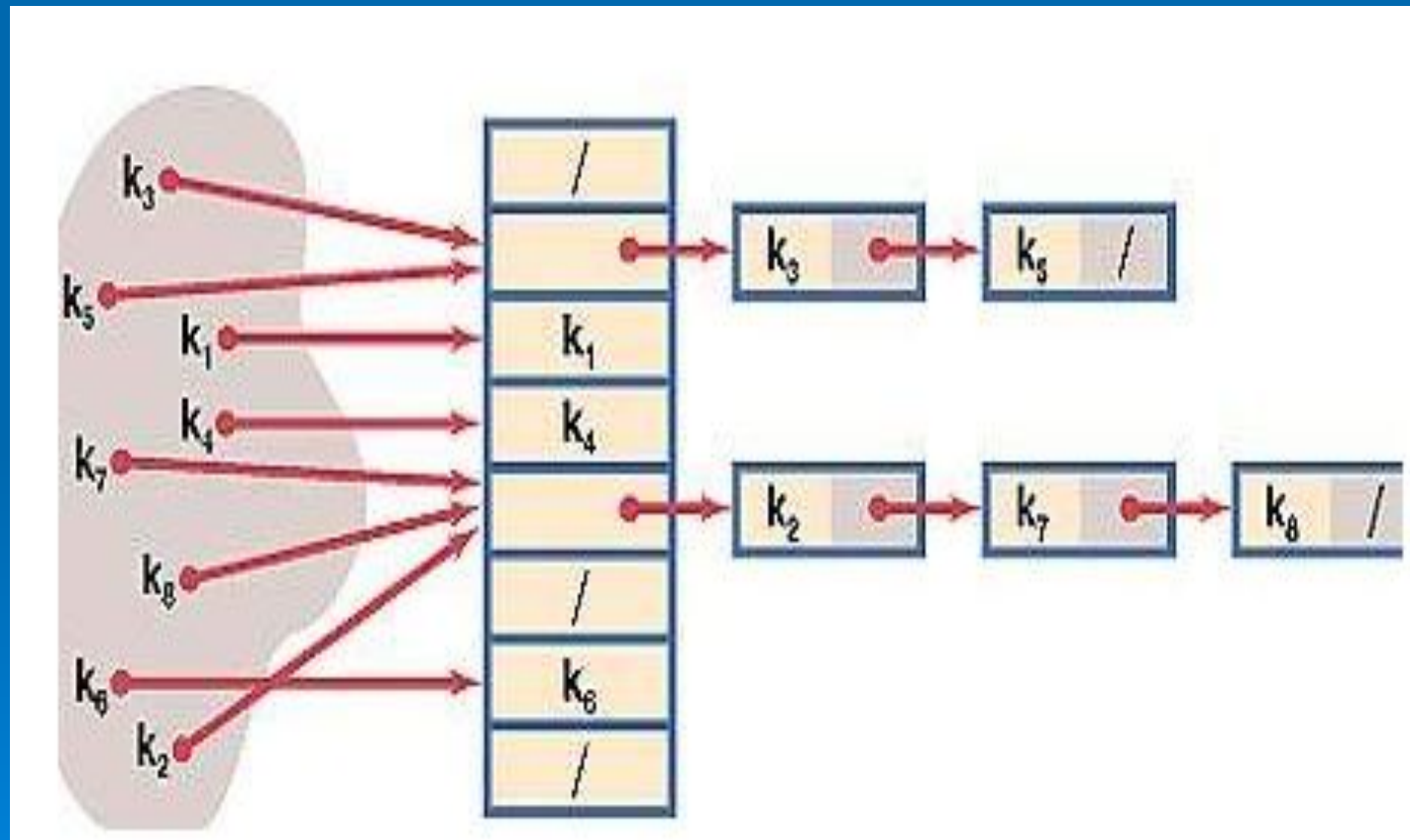
- 1) швидкий пошук інформації про ідентифікатор
- 2) швидке додавання ідентифікаторів у таблицю

Таблиці розміщення (хеш-таблиці)

Традиційна класифікація	Класифікацією Ахо, Хопкрофта, Ульмана
хешування з ланцюжками (зі списками)	відкрите хешування (багатовимірне)
хешування з відкритою адресацією	закрите хешування (одновимірне)

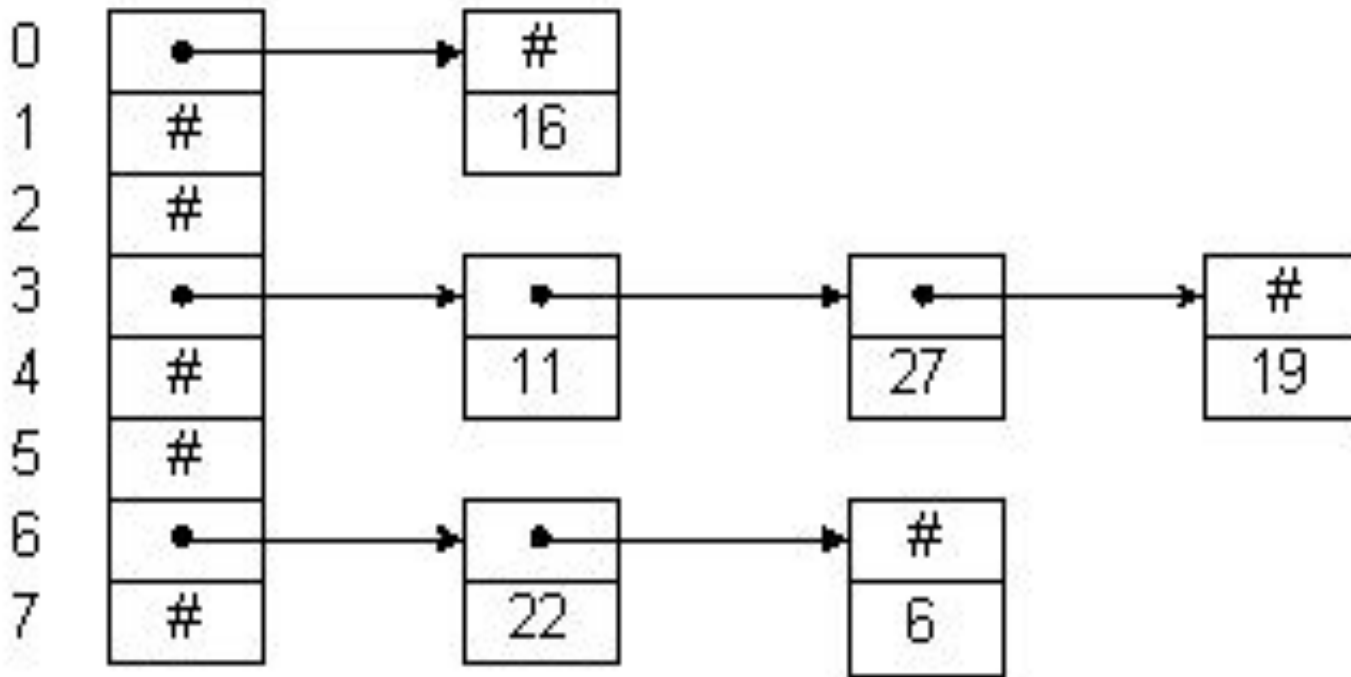
Схеми хешування

Хешування з ланцюжками (зі списками)



Приклад хешування зі списками

HashTable



Хешування з відкритою адресацією (одновимірне)

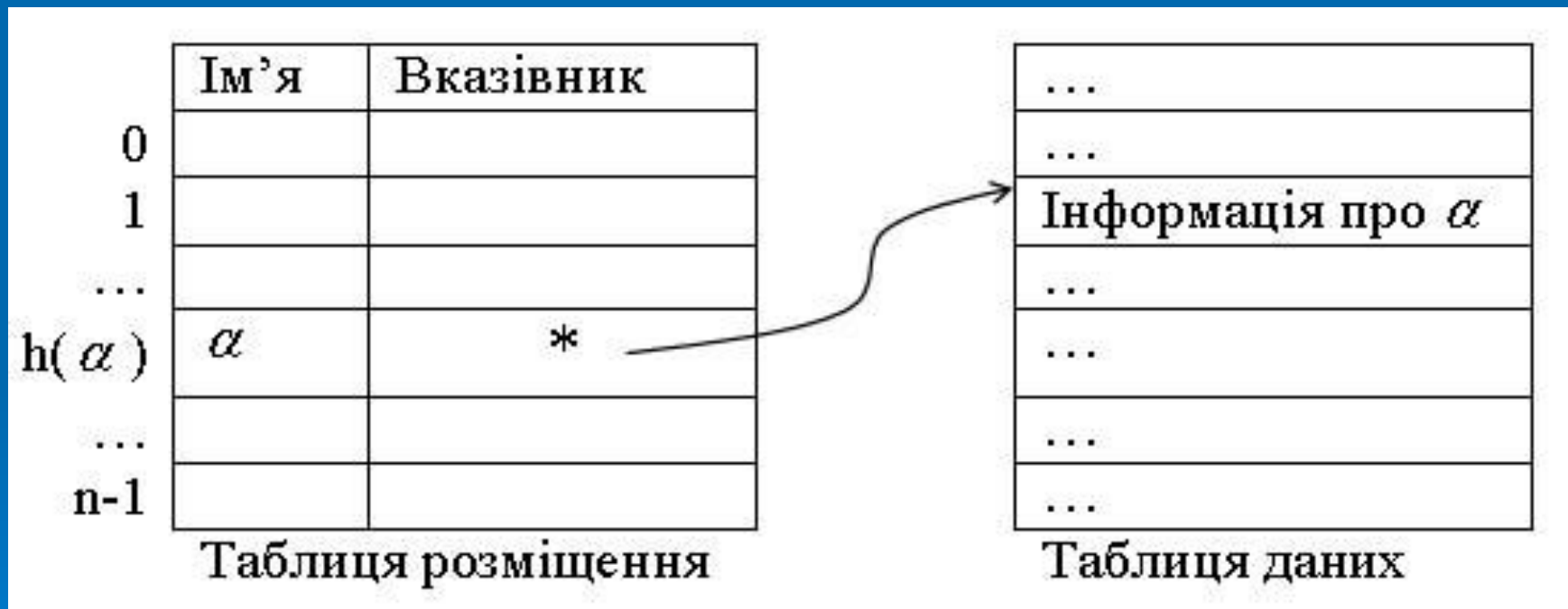


Схема пам'яті

хешування cost, tax і price

0	tax	p2 *	інформація про cost інформація про tax інформація про price
1	price	p3 *	
2			
3	cost	p1 *	
4			
5			

$$h_0(\alpha) = \text{CODE}(\alpha) \bmod 6$$

$$h_i(\alpha) = (\text{CODE}(\alpha) + i + 2) \bmod 6, \quad i = 1, \dots, 5$$

$$\text{CODE}(\text{cost}) = 3 + 15 + 19 + 20 = 57$$

$$h_0(\text{cost}) = \text{CODE}(\text{cost}) \bmod 6 = 57 \bmod 6 = 3$$

$$\text{CODE}(\text{tax}) = 20 + 1 + 24 = 45$$

$$h_0(\text{tax}) = \text{CODE}(\text{tax}) \bmod 6 = 45 \bmod 6 = 3$$

$$h_1(\text{tax}) = 48 \bmod 6 = 0$$

$$\text{CODE}(\text{price}) = 16 + 18 + 9 + 3 + 5 = 51$$

$$h_0(\text{price}) = \text{CODE}(\text{price}) \bmod 6 = 51 \bmod 6 = 3$$

$$h_1(\text{price}) = 54 \bmod 6 = 0$$

$$h_2(\text{price}) = 55 \bmod 6 = 1$$

$$\text{CODE}(a) = 1 \quad \text{CODE}(p) = 16$$

$$\text{CODE}(c) = 3 \quad \text{CODE}(r) = 18$$

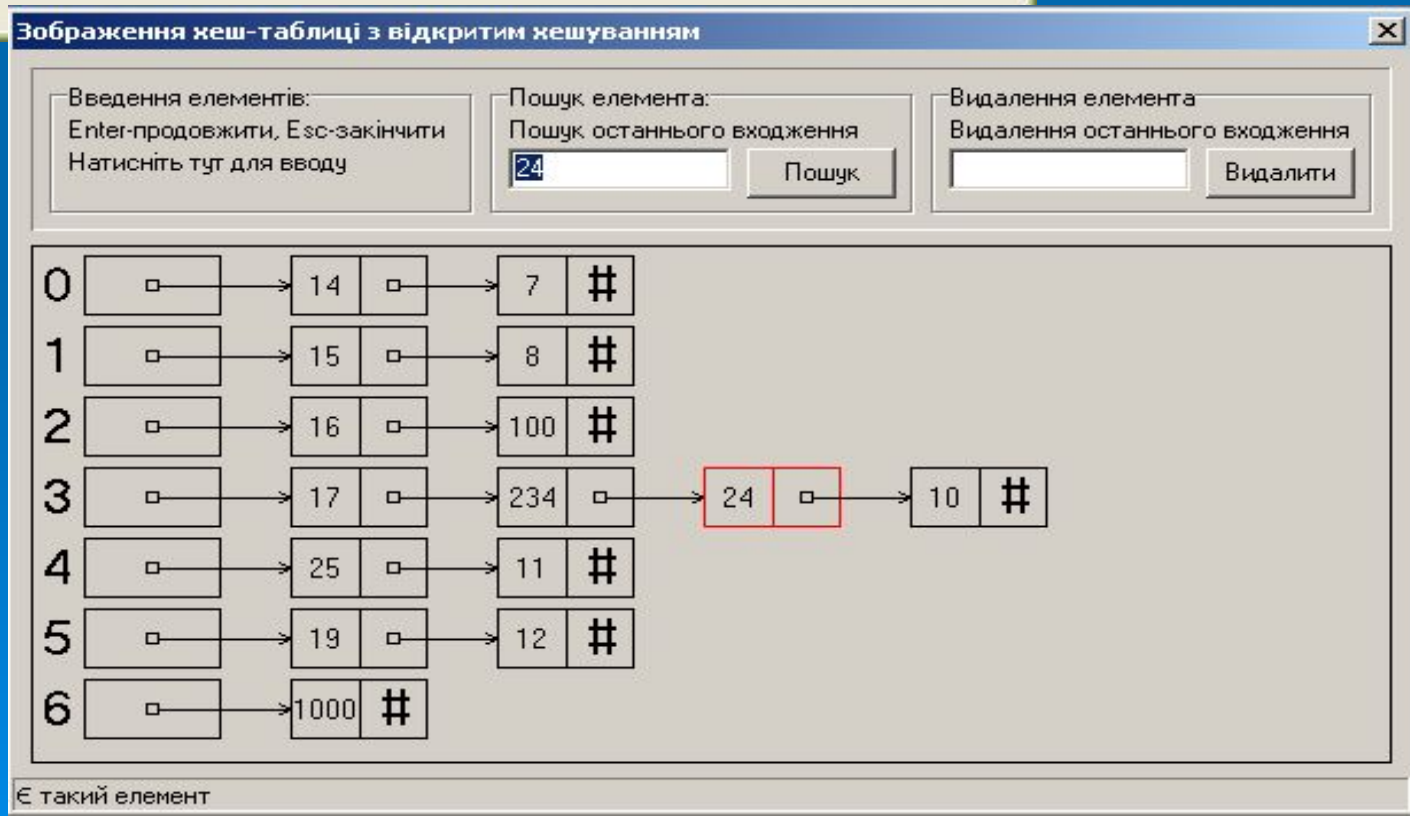
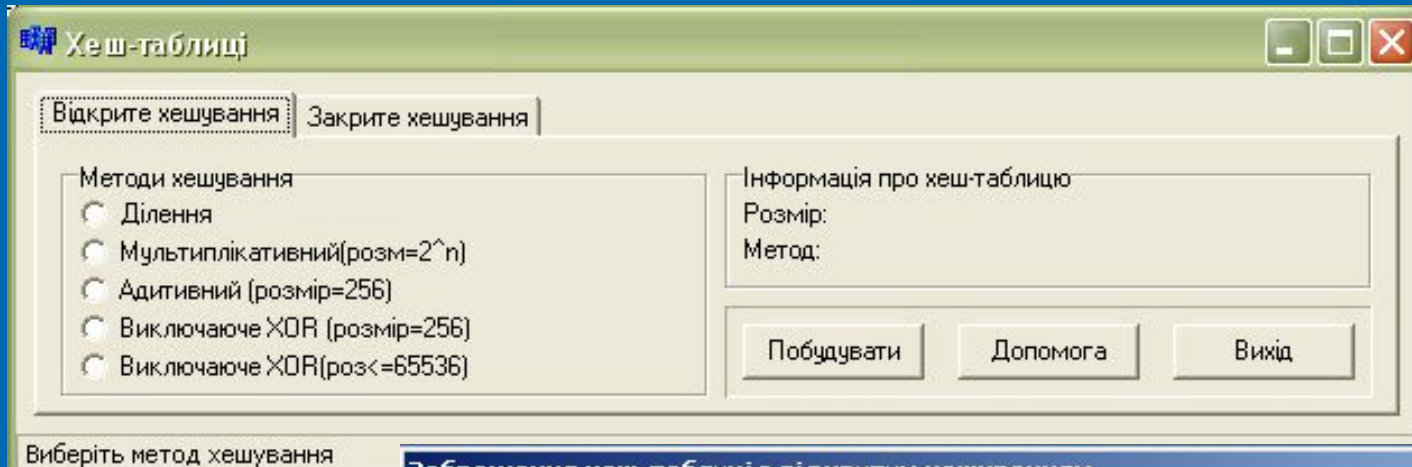
$$\text{CODE}(e) = 5 \quad \text{CODE}(s) = 19$$

$$\text{CODE}(i) = 9 \quad \text{CODE}(t) = 20$$

$$\text{CODE}(o) = 15 \quad \text{CODE}(x) = 24$$

Адреса у локальній мережі:

Програмна група Арт \Math_02 \Zavd_Lab \SystProg \Проекти \Хеш таблиці



Функції розміщення

// Ділення

```
typedef int HashIndexType;
const int HashTableSize=7;

HashIndexType Hash(int Key)
{
    return Key % HashTableSize;
}
```

// Адитивний метод

```
typedef
unsigned char HashIndexType; //8
bit
const int HashTableSize=256;

HashIndexType Hash(char *str)
{
    HashIndexType h = 0;
    while (*str) h+= *str++;
    return h;
}
```

Методи вирішення конфліктів

Побудова вторинних функцій

h_1, \dots, h_m

$h_j(\alpha) \neq h_i(\alpha)$ для всіх $i \neq j$, $m = n-1$.

□ $h_i(\alpha) = (h_0(\alpha) + i) \bmod n$, (1)

□ $h_i(\alpha) = (h_0(\alpha) + r_i) \bmod n$, (2)

□ $h_i(\alpha) = (i(h_0(\alpha) + 1)) \bmod n$, (3)

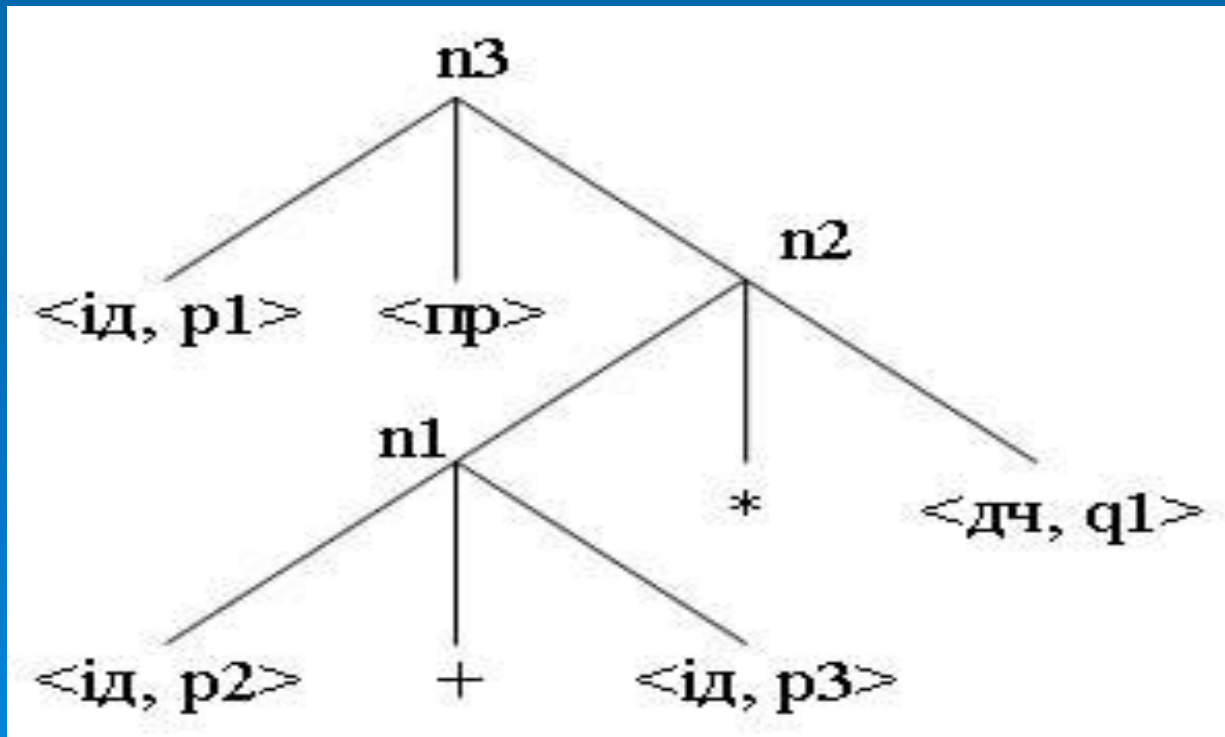
□ $h_i(\alpha) = (h_0(\alpha) + ai^2 + bi) \bmod n$, (4)

$i=1, 2, \dots, n-1$

2.3. Синтаксичний аналіз програми

`cost := (price + tax)*0,98` (1)

$\langle \text{ід, } p1 \rangle \langle \text{пр} \rangle (\langle \text{ід, } p2 \rangle + \langle \text{ід, } p3 \rangle) * \langle \text{дч, } q1 \rangle$



2.4. Генерація проміжного коду

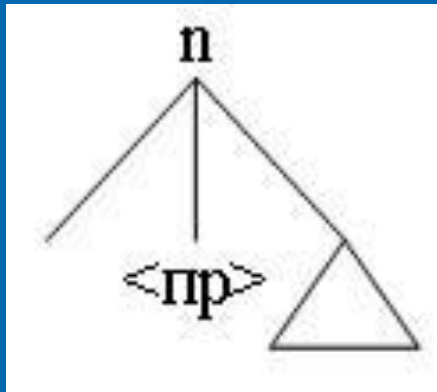
Введемо позначення:

$R(m)$ – містиме комірки m .

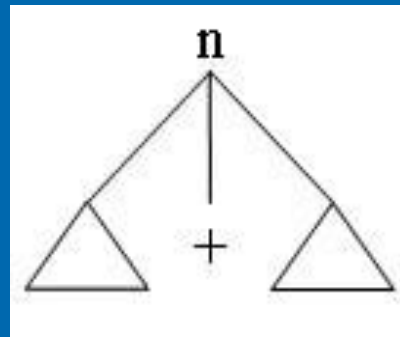
$=m$ – числове значення m .

Команда	Значення
Load m	$R(m) \rightarrow \text{суматор}$
Add m	$\text{суматор} + R(m) \rightarrow \text{суматор}$
Mpy m	$\text{суматор} * R(m) \rightarrow \text{суматор}$
Store m	$\text{суматор} \rightarrow R(m)$
Load $=m$	$m \rightarrow \text{суматор}$
Add $=m$	$\text{суматор} + m \rightarrow \text{суматор}$
Mpy $=m$	$\text{суматор} * m \rightarrow \text{суматор}$

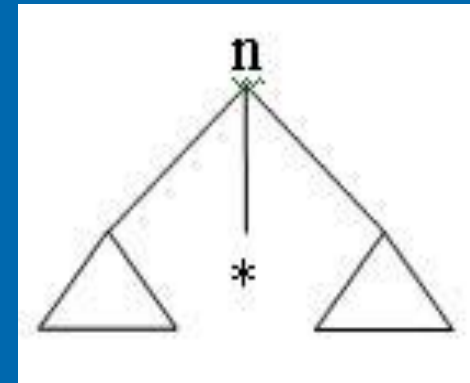
2.4. Генерація проміжного коду



a)



б)



в)

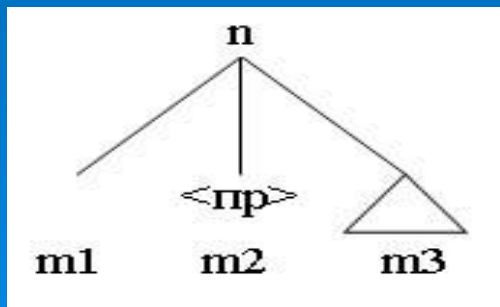
Введемо позначення:

$C(n)$ – частина проміжного коду, що відповідає вершині .
 $l(n)$ – рівень вершини .

$$l(n) = \begin{cases} 0, & \text{якщо } n \text{ немає нащадків} \\ \max_{1 \leq i \leq k} l(n_i) + 1, & n_i - \text{нащадки вершини } n \end{cases}$$

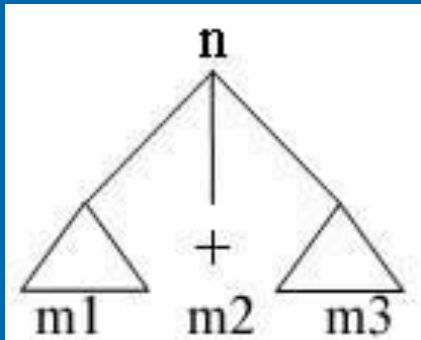
Визначимо код , для кожного типу вершини:

- 1)
 - Якщо n – лист, який відповідає ідентифікатору, то $C(n)$ – це ім'я змінної, яке відповідає ідентифікатору(cost).
 - Якщо n – лист, який відповідає дійсному числу, то $C(n)$ – дійсне число(=0.98).
 - Якщо n – лист, який відповідає лексемам $+$, $*$, $\langle пр \rangle$, то їм не відповідає ніякий код.
- 2) Якщо n – вершина типу а), $m1$, $m2$, $m3$ – нащадки, тоді код цієї вершини :



$$C(n) \rightarrow \begin{matrix} Load & C(m3) \\ Store & C(m1) \end{matrix}$$

3) Якщо n – вершина типу б), m_1, m_2, m_3 – нащадки, то вершині відповідає такий



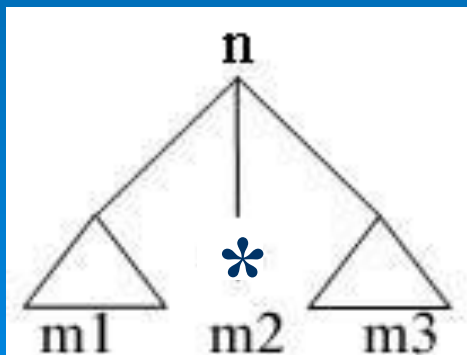
$C(n) \rightarrow C(m_3)$

Store $\$l(n)$

Load $C(m_1)$

Add $\$l(n)$

4) Якщо n – вершина типу в), **m_1, m_2, m_3** – нащадки, тоді код цієї вершини :



$C(n) \rightarrow C(m_3)$

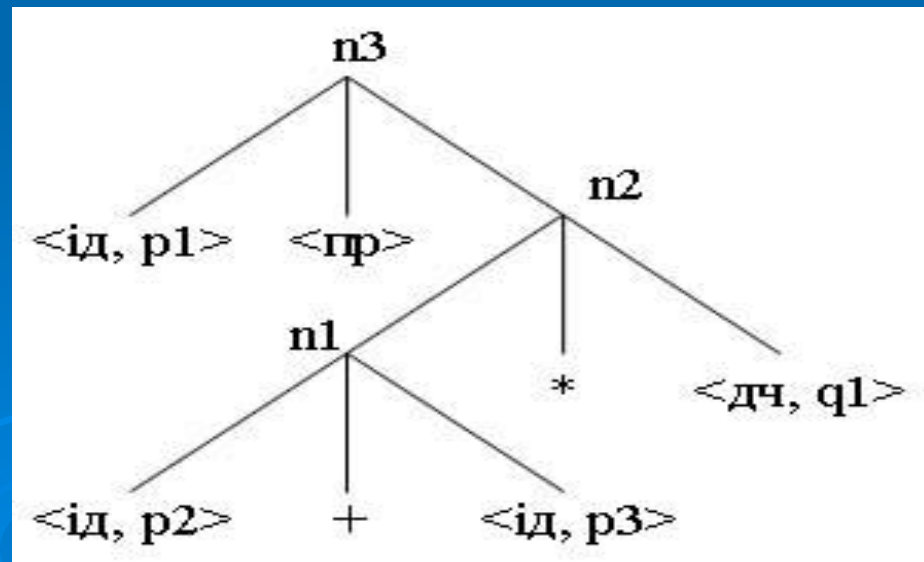
Store $\$l(n)$

Load $C(m_1)$

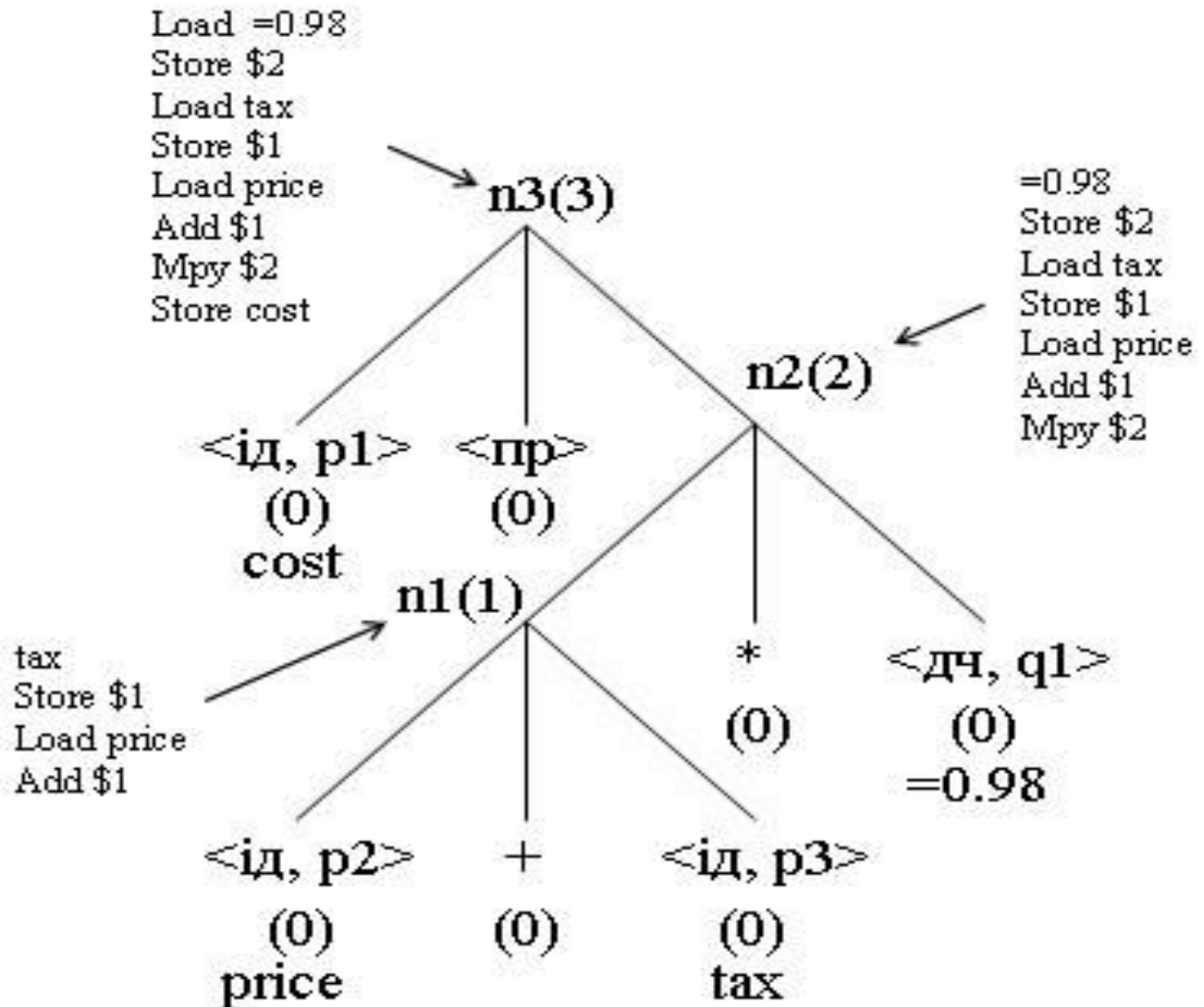
Mpy $\$l(n)$

$$\text{cost} := (\text{price} + \text{tax}) * 0,98 \quad (1)$$

$C(n1)$: *tax; Store \$1; Load price; Add \$1*
 $C(n2)$: *:= 0.98; Store \$2; Load C(n1); Mpy \$2*
 $C(n3)$: *Load C(n2); Store Cost*



Проміжний код

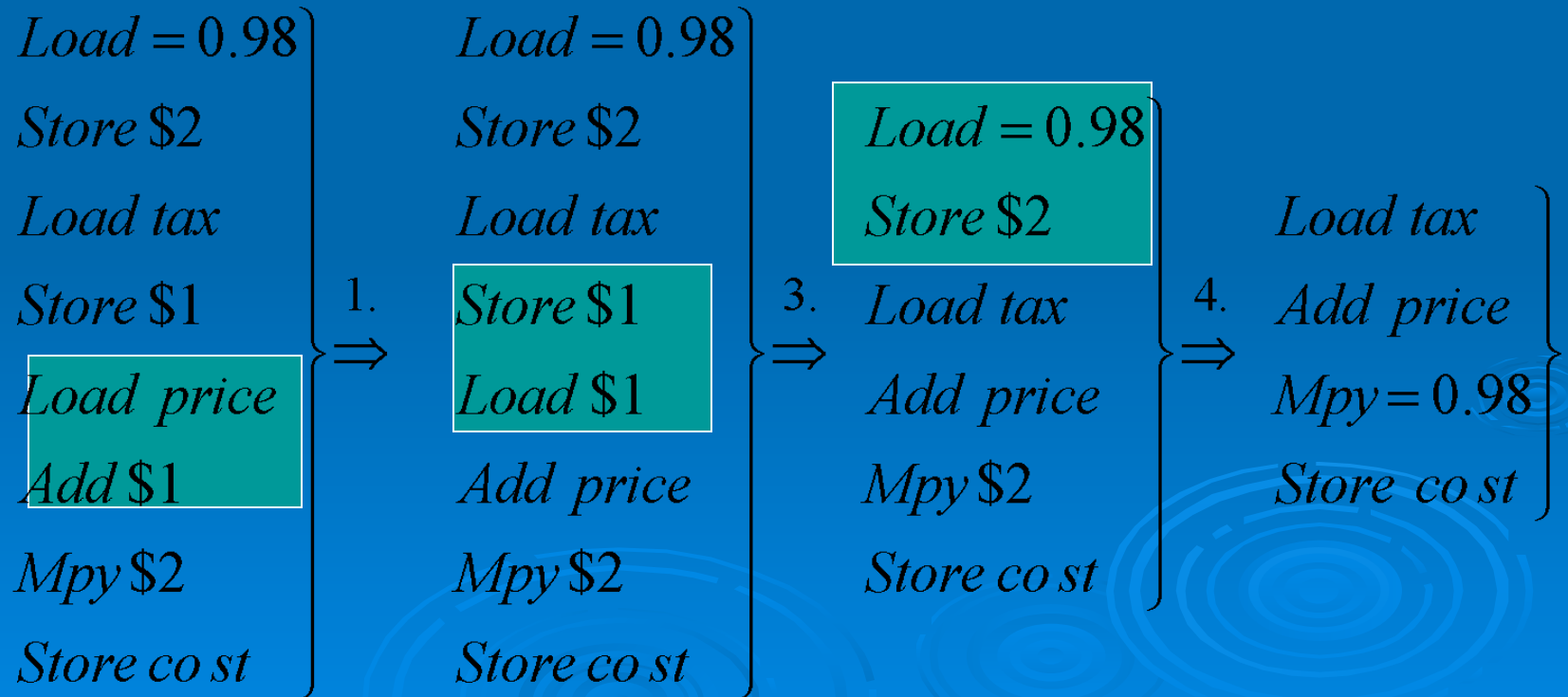


2.5. Оптимізація проміжного коду

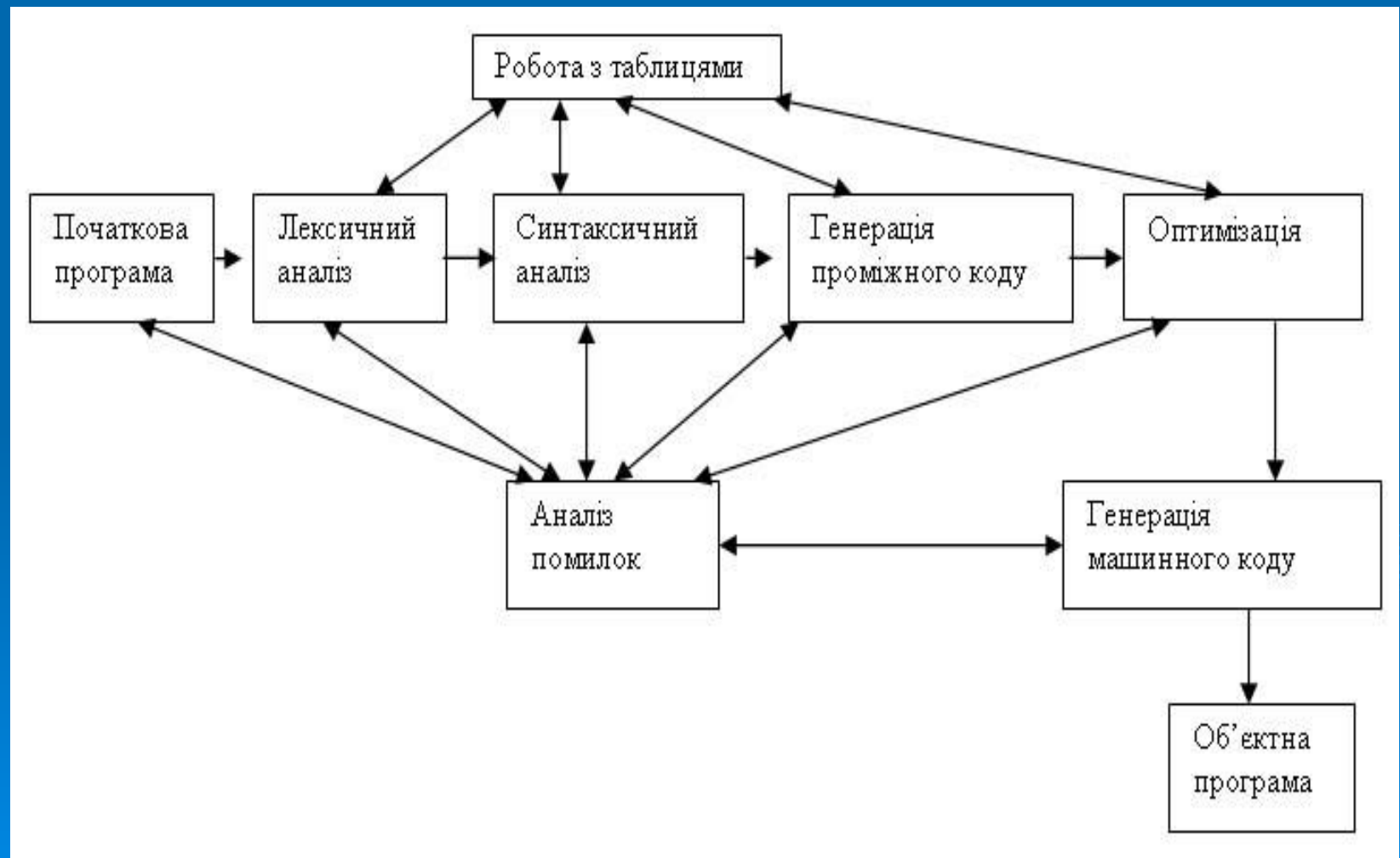
- 1)
$$\left. \begin{array}{l} \textit{Load } a \\ \textit{Add } b \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{l} \textit{Load } b \\ \textit{Add } a \end{array} \right.$$
 операція «+» є комутативною в тому випадку, коли на *Add b* не передавалось управління.
- 2)
$$\left. \begin{array}{l} \textit{Load } a \\ \textit{Mpy } b \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{l} \textit{Load } b \\ \textit{Mpy } a \end{array} \right.$$
 операція «*» є комутативною .
- 3)
$$\begin{array}{l} \textit{Store } a \\ \textit{Load } a \end{array}$$
 можна вилучити при умові, що надалі комірка *a* не буде використовуватись або буде заповнена безпосередньо перед використанням.
- 4)
$$\begin{array}{l} \textit{Load } a \\ \textit{Store } b \end{array}$$
 можна вилучити при умові, що за нею слідує інший оператор *Load* і немає переходу до *Store b*. Наступні входження *b* замінюються на *a* до того моменту, поки знову не з'явиться оператор *Store b*.

2.5. Оптимізація проміжного коду

$$\mathbf{cost := (price + tax) * 0,98}$$



2.6. Аналіз помилок компіляції та генерація машинного коду



2.7. Взаємодія етапів компіляції, проходи компілятора.



(1) – початкова програма.

(2) – послідовність лексем.

(3) – проміжний код.

(4) – машинний код.