Шаблони для ведення лекцій за презентаціями (підготовлені для друку)

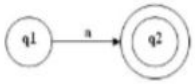
2. Алгоритм побудови недетермінованого скінченного автомата за регулярним виразом

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

$R_0 = e \qquad M_0 = (\{q_0\}, \Sigma, \varnothing, q_0, \{q_0\}).$

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

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



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

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

$Q_1 \cap Q_2 = \varnothing$

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

Приклад конспекту лекцій за презентаціями

Під час роботи всіх фаз роботи мовного процесора відб. аналіз помилок

1. Помилки лексичного аналізу (неможливість утворити лексему)
2. Помилки які будуються під час синтаксичного аналізу
3. Неможливість виконати код (опис дійсної не відбувся (приклад))

Бажано видавати всі помилки із поясненнями і рекомендаціями

Іноді будується зразу машинний код особливо коді коли немає оптимізації проміжного коду.

Виокремлюються окремі задачі: Вибір помилок, редисерів для обчислення, ...

Модель мовного процесора

на різних прод. різні фази вик. по кількому і по різному взаємодіють

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

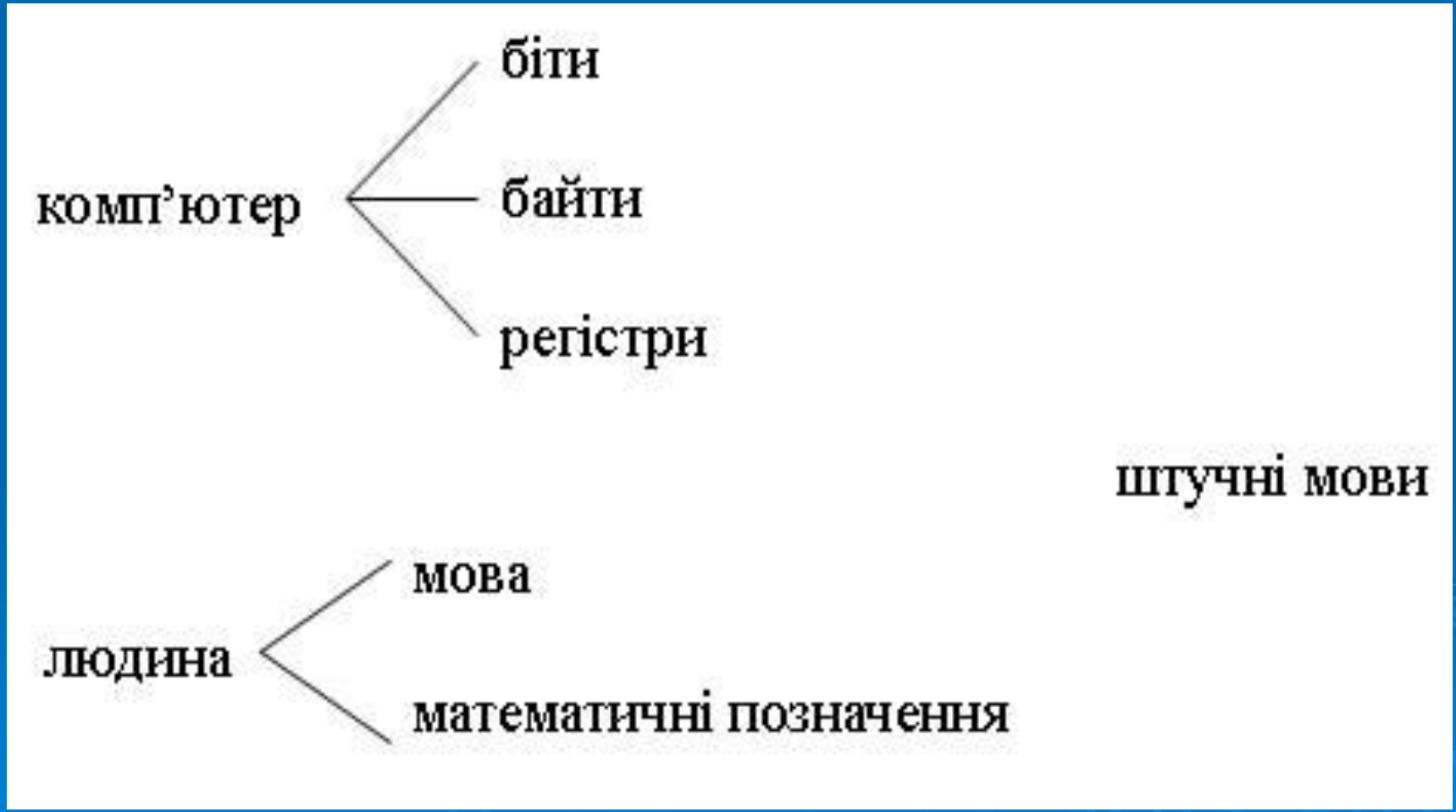
2.4. генерація проміжного коду

2.5. оптимізація проміжного коду

2.6. аналіз помилок компіляції та генерація машинного коду

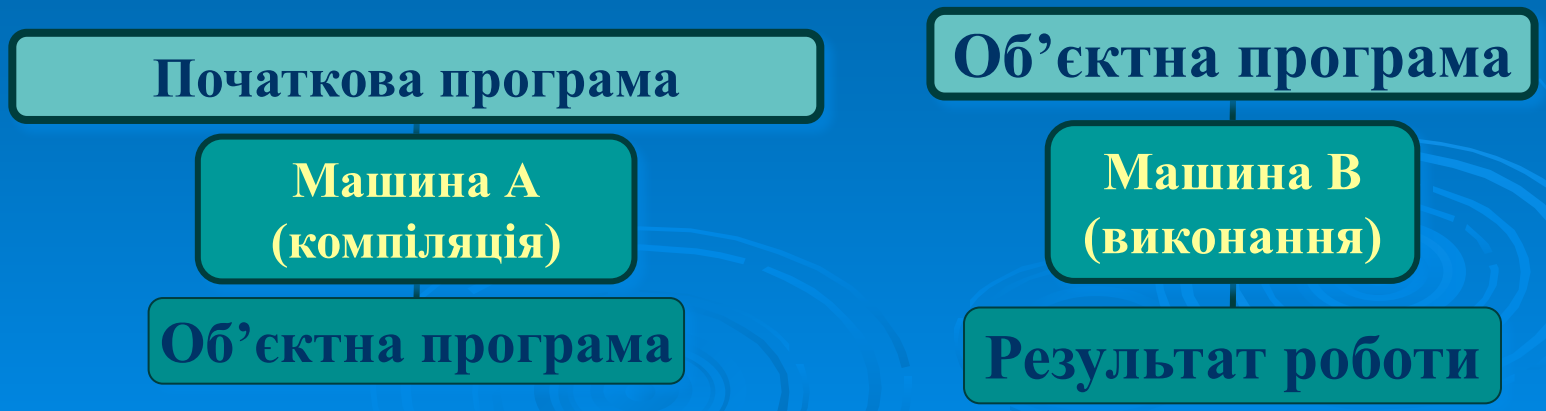
2.7. взаємодія етапів компіляції, проходи компілятора.

1. Поняття мовного процесора, типи мовних процесорів



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, p1><пр>(<ід, p2> + <ід, p3>) * <дч, q1>

2.2. Робота з таблицями (хеш-таблиці)

Integer cost, tax, price

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

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

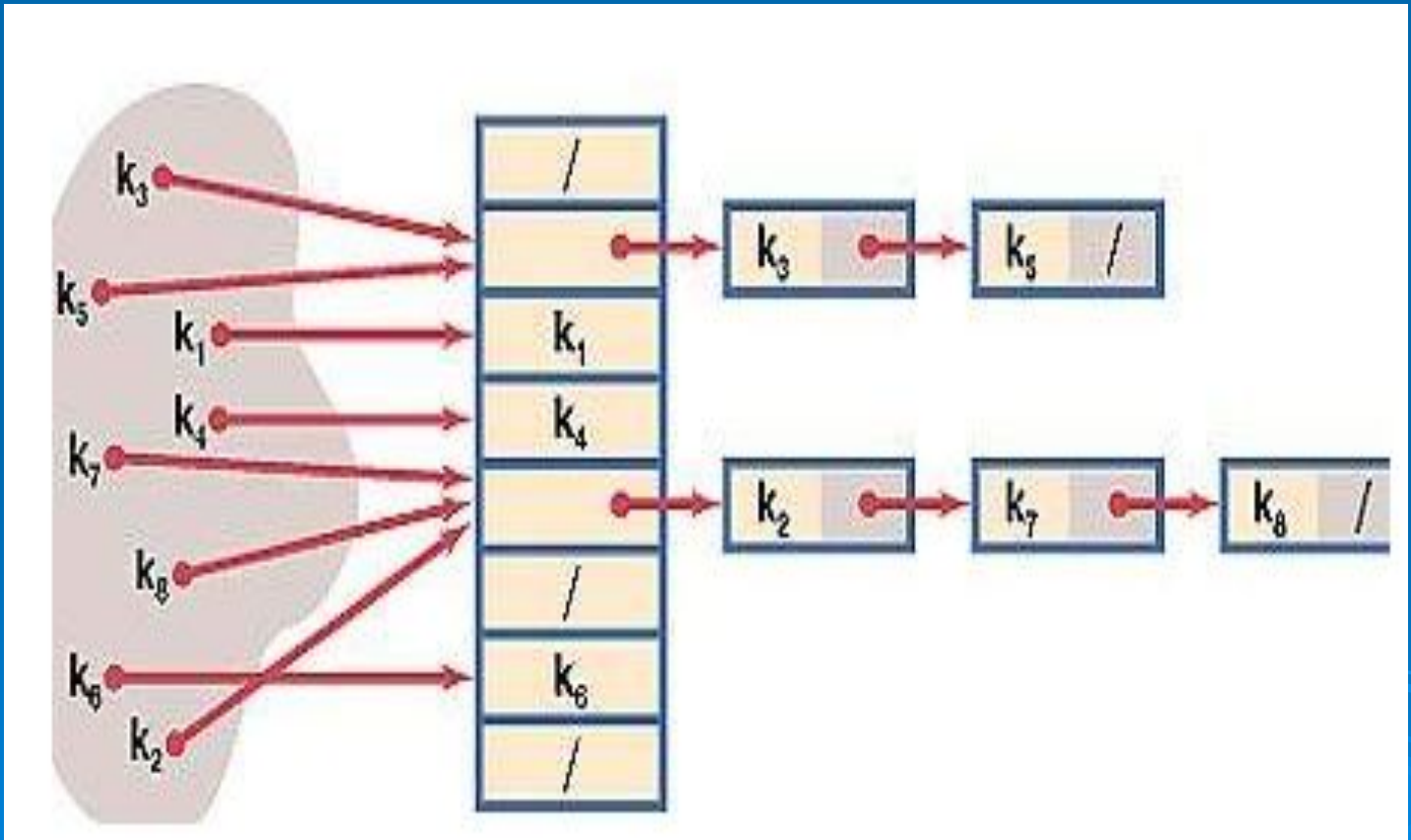
1) **швидкий пошук інформації про ідентифікатор**
2) **швидке додавання ідентифікаторів у таблицю**

Таблиці розміщення (хеш-таблиці)

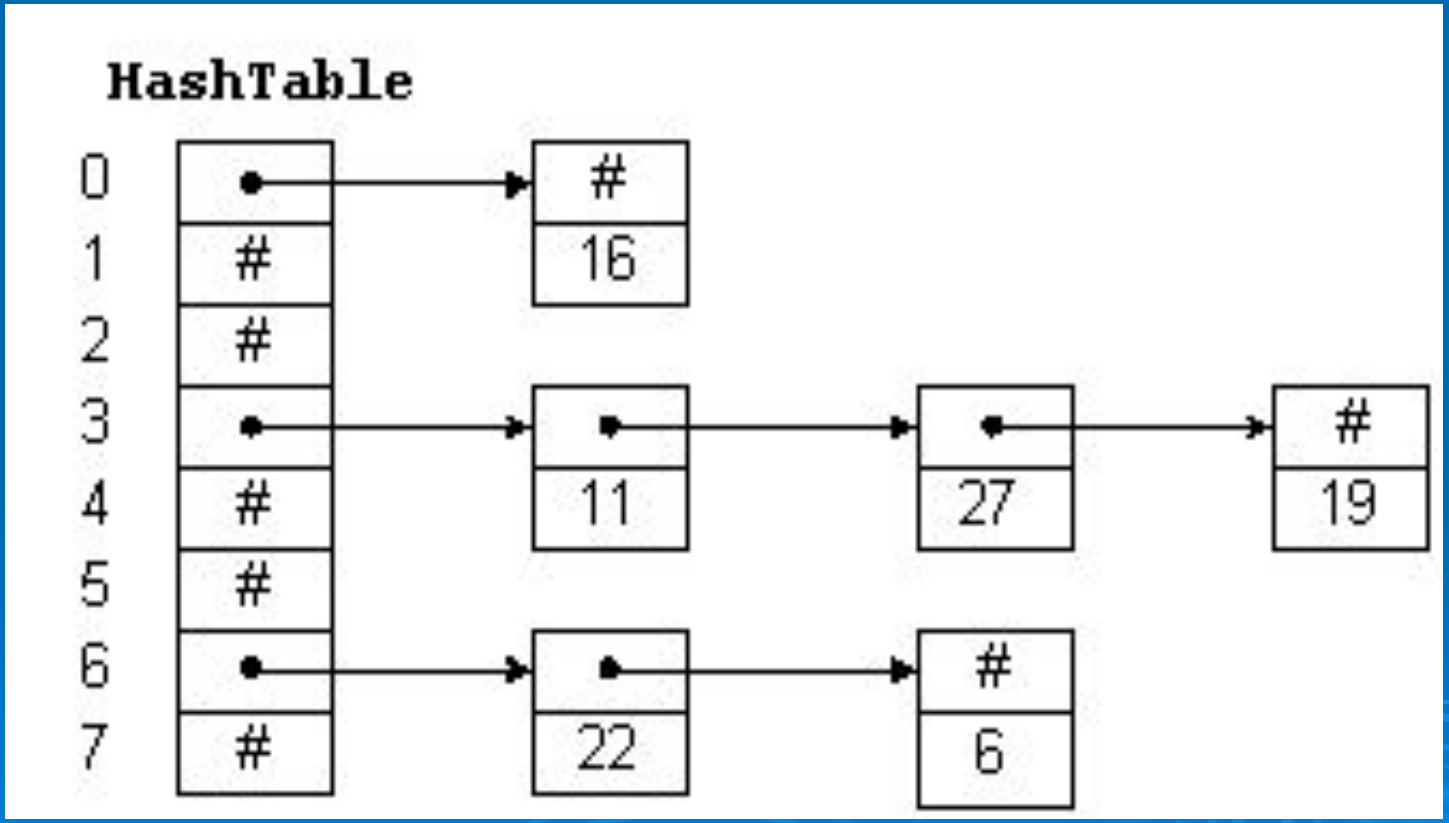
Традиційна класифікація	Класифікацією Ахо, Хопкрофта, Ульмана
хешування з ланцюжками (зі списками)	відкрите хешування (багатовимірне)
хешування з відкритою адресацією	закрите хешування (одновимірне)

Схеми хешування

Хешування з ланцюжками (зі списками)



Приклад хешування зі списками



Хешування з відкритою адресацією (одновимірне)

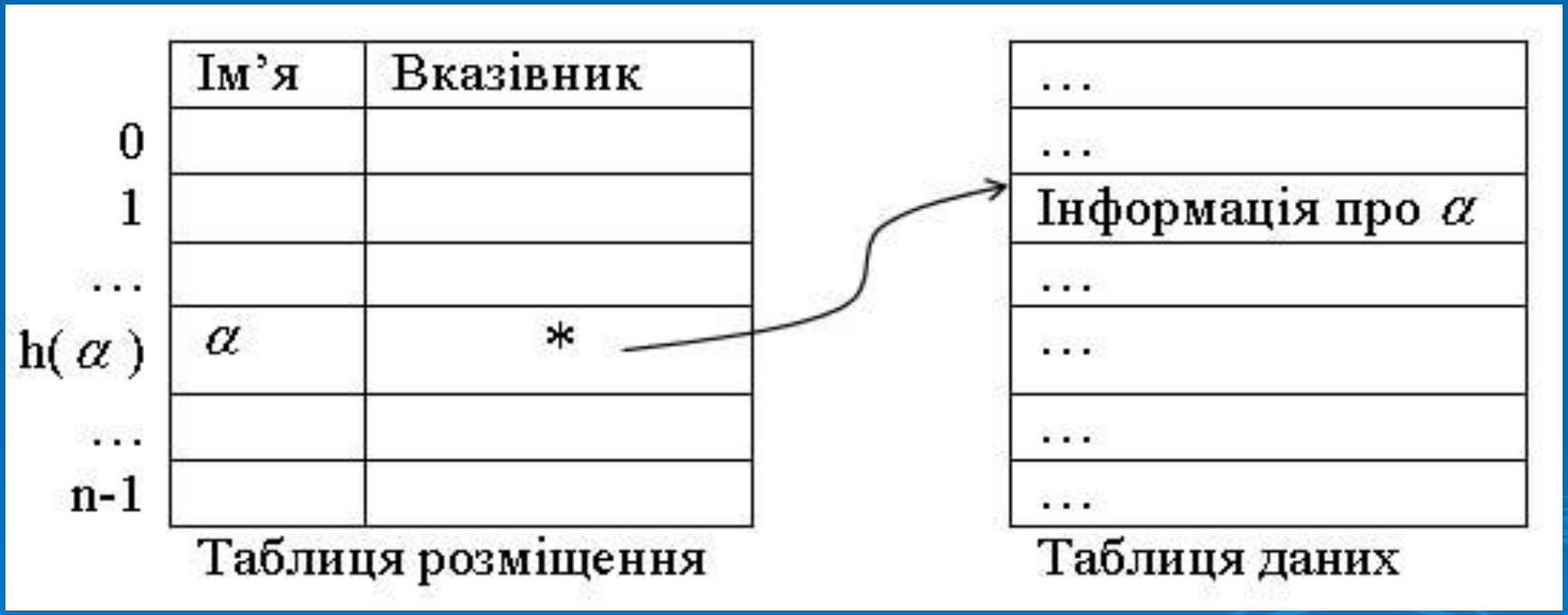
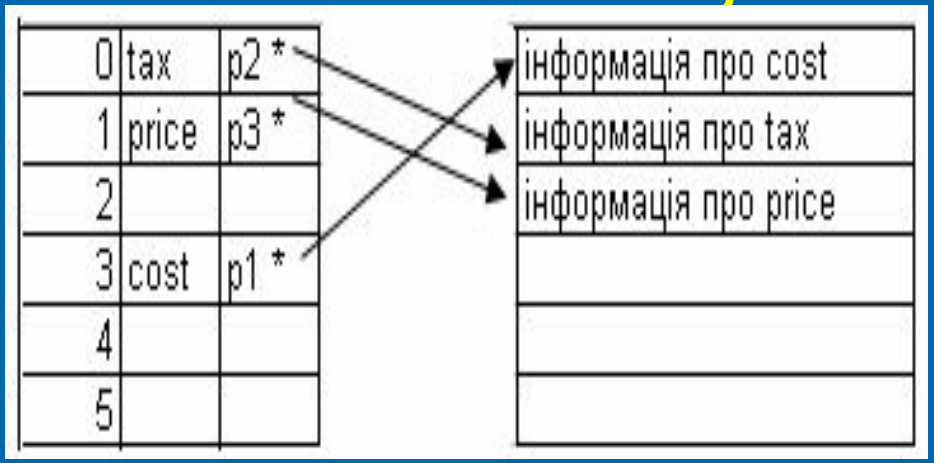


Схема пам’яті

хешування cost, tax і price



$h_0(\alpha)=CODE(\alpha) \mod 6$

$h_i(\alpha)=(CODE(\alpha)+i+2) \mod 6),\ i=1,\ldots,5$

CODE(a)=1 CODE(p)=16

CODE(c)=3 CODE(r)=18

CODE(e)=5 CODE(s)=19

CODE(i)=9 CODE(t)=20

CODE(o)=15 CODE(x)=24

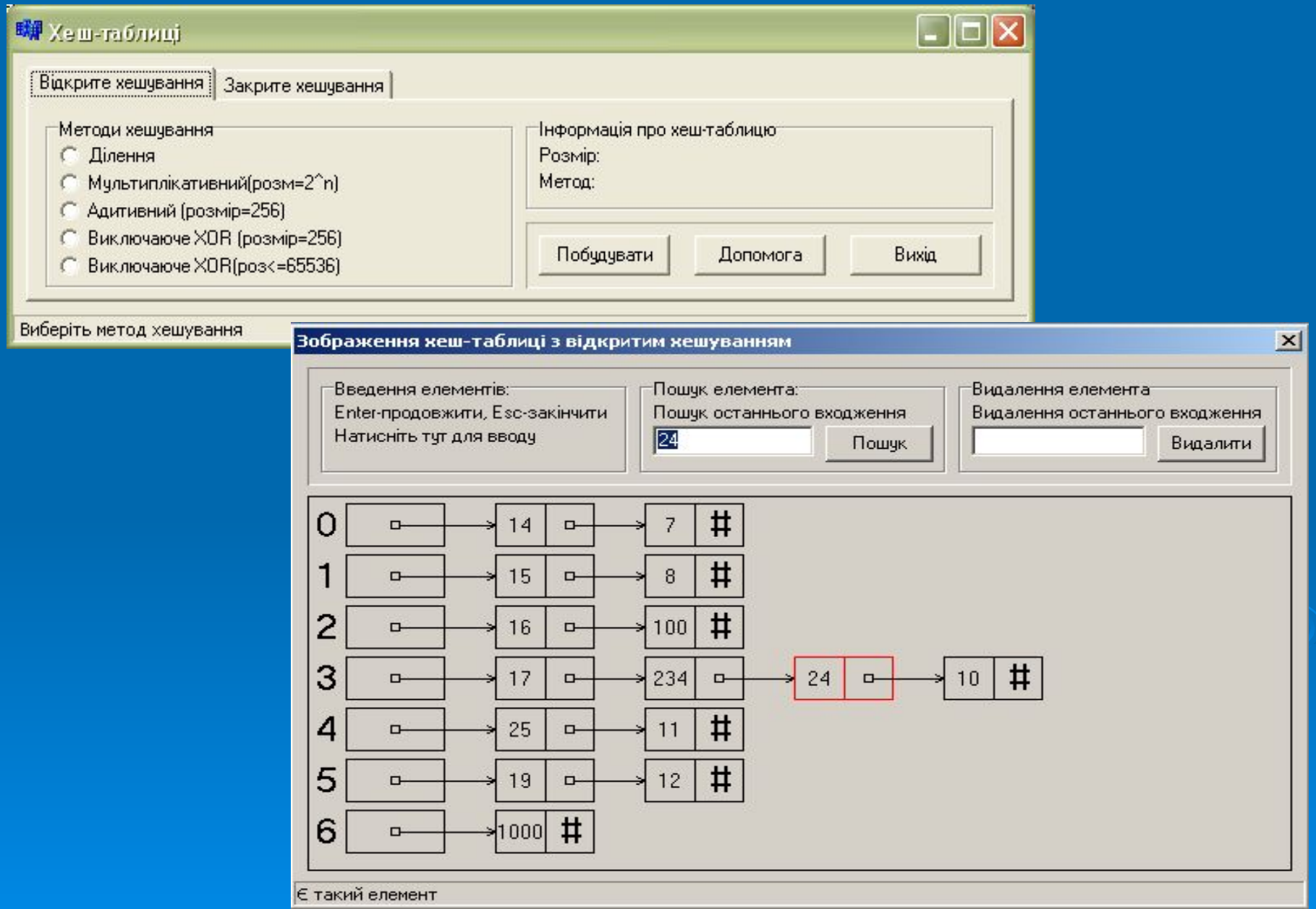
CODE (cost)=3+15+19+20=57
h_0(*cost*)= CODE(*cost*) mod 6= 57 mod 6 =3

CODE (tax)= 20+1+24=45
h_0(*tax*)= CODE(*tax*) mod 6 = 45 mod 6 = 3
h_1(*tax*)= 48 mod 6 = 0

CODE (price)=16+18+9+3+5=51
h_0(*price*)= CODE (*price*) mod 6 = 51 mod 6 = 3
h_1(*price*)= 54 mod 6 =0
h_2(*price*)= 55 mod 6 =1

Адреса у локальній мережі:

Програмна група Apm \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 $h_1,\ldots,h_m$

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

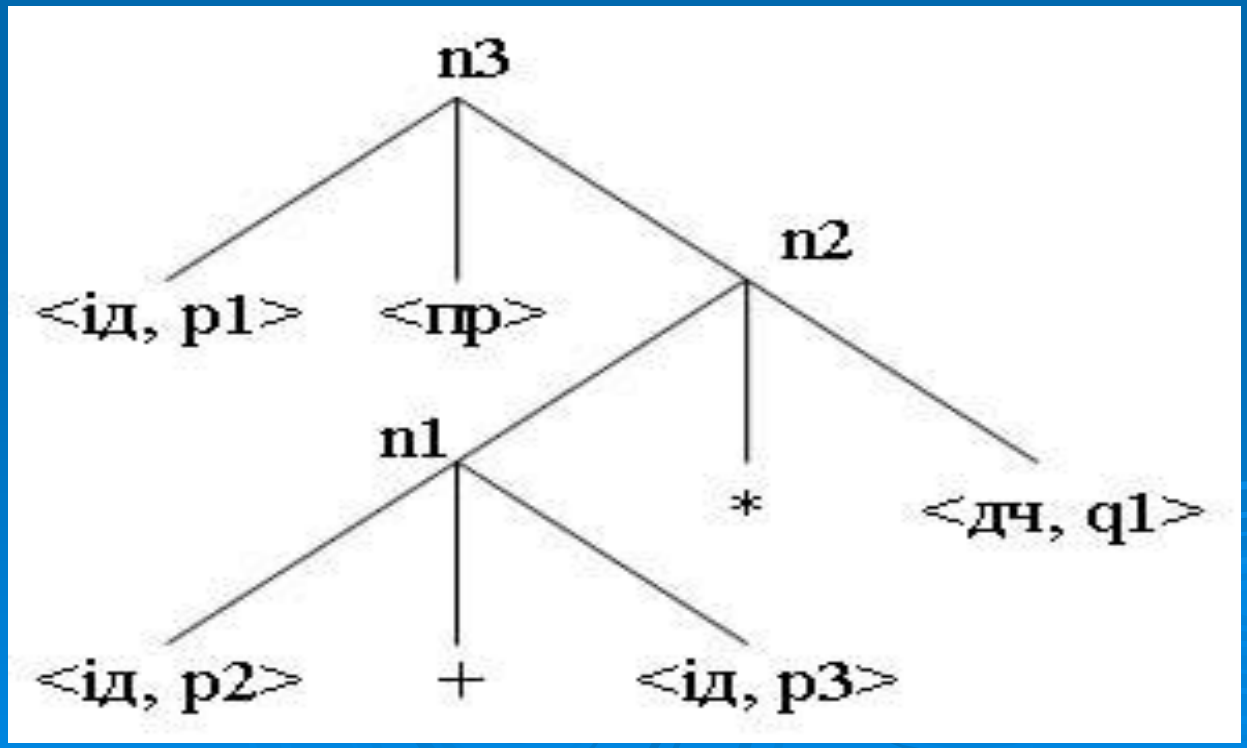
- $h_i(\alpha) = (h_0(\alpha) + i) \bmod n$, (1)
- $h_i(\alpha) = (h_0(\alpha) + r_i) \bmod n$, (2)
- $h_i(\alpha) = (i(h_0(\alpha) +1)) \bmod n$, (3)
- $h_i(\alpha) = (h_0(\alpha) + ai^2+bi) \bmod n$, (4)

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

2.3. Синтаксичний аналіз програми

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

<ід, p1> <пр> (<ід, p2> + <ід, p3>) * <дч, q1>



2.4. Генерація проміжного коду

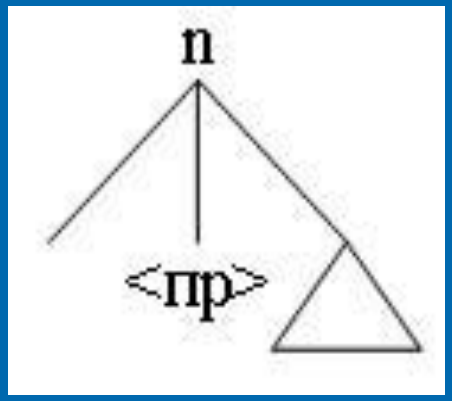
Введемо позначення:

R(m) – містиме комірки m.

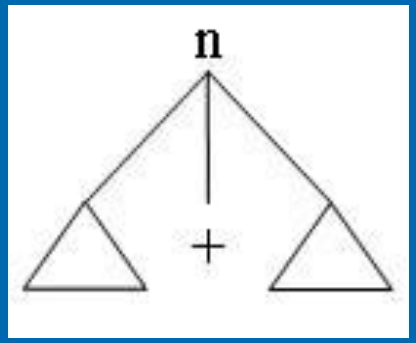
=m – числове значення m.

Команда	Значення
Load m	R(m)→суматор
Add m	суматор + R(m)→суматор
Mpy m	суматор * R(m)→суматор
Store m	суматор → R(m)
Load =m	m → суматор
Add =m	суматор + m → суматор
Mpy =m	суматор * m →суматор

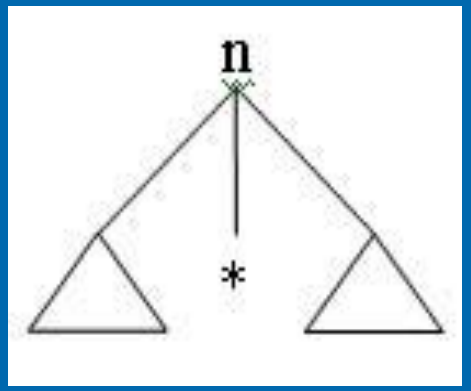
2.4. Генерація проміжного коду



а)



б)



в)

Введемо позначення:

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

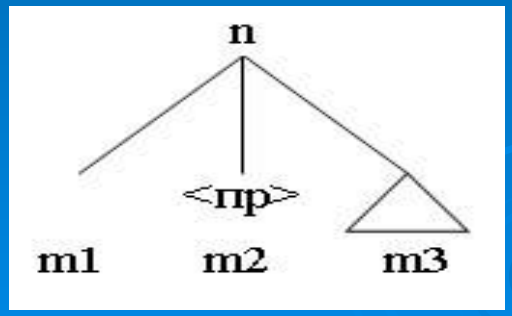
$l(n)$ – рівень вершини .

$$l(n)=\begin{cases}0,\ якщо\ n\ \ немає\ \ нащадків\\ \max\limits_{1\le i\le k} l(n_i)+1,\ \ n_i-нащадки\ \ вершини\ \ n\end{cases}$$

Визначимо код , для кожного типу вершини:

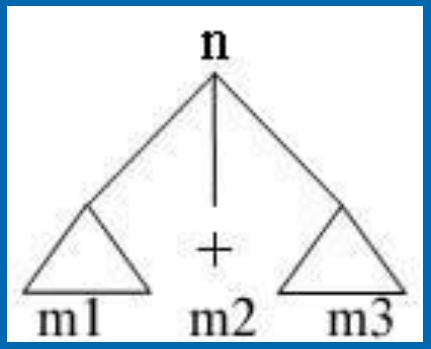
1)
- Якщо n – лист, який відповідає ідентифікатору, то C(n) – це ім’я змінної, яке відповідає ідентифікатору(cost).
- Якщо n – лист, який відповідає дійсному числу, то C(n) – дійсне число(=0.98).
- Якщо n – лист, який відповідає лексемам +, *, <пр>, то їм не відповідає ніякий код.

2) Якщо n – вершина типу a), m1, m2, m3 – нащадки, тоді код цієї вершини :



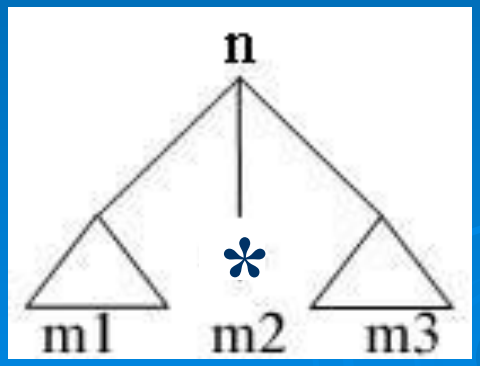
$$C(n) \rightarrow Load \quad C(m3)$$
$$Store \quad C(m1)$$

3) Якщо n – вершина типу б), m1, m2, m3 – нащадки, то вершині відповідає такий



$$C(n) \rightarrow C(m3)$$
$$Store\ \$l(n)$$
$$Load\ C(m1)$$
$$Add\ \$l(n)$$

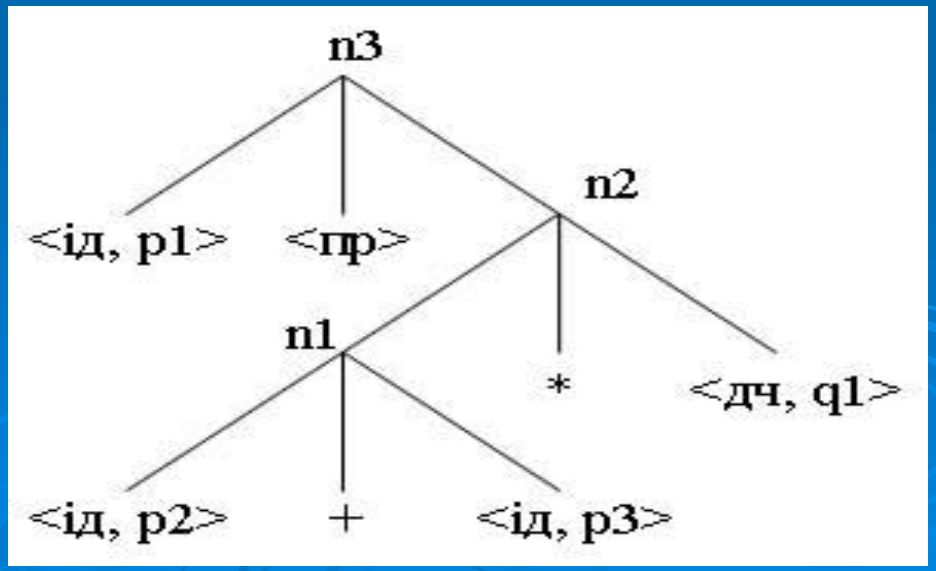
4) Якщо n – вершина типу в), **m1, m2, m3 –** нащадки, тоді код цієї вершини :



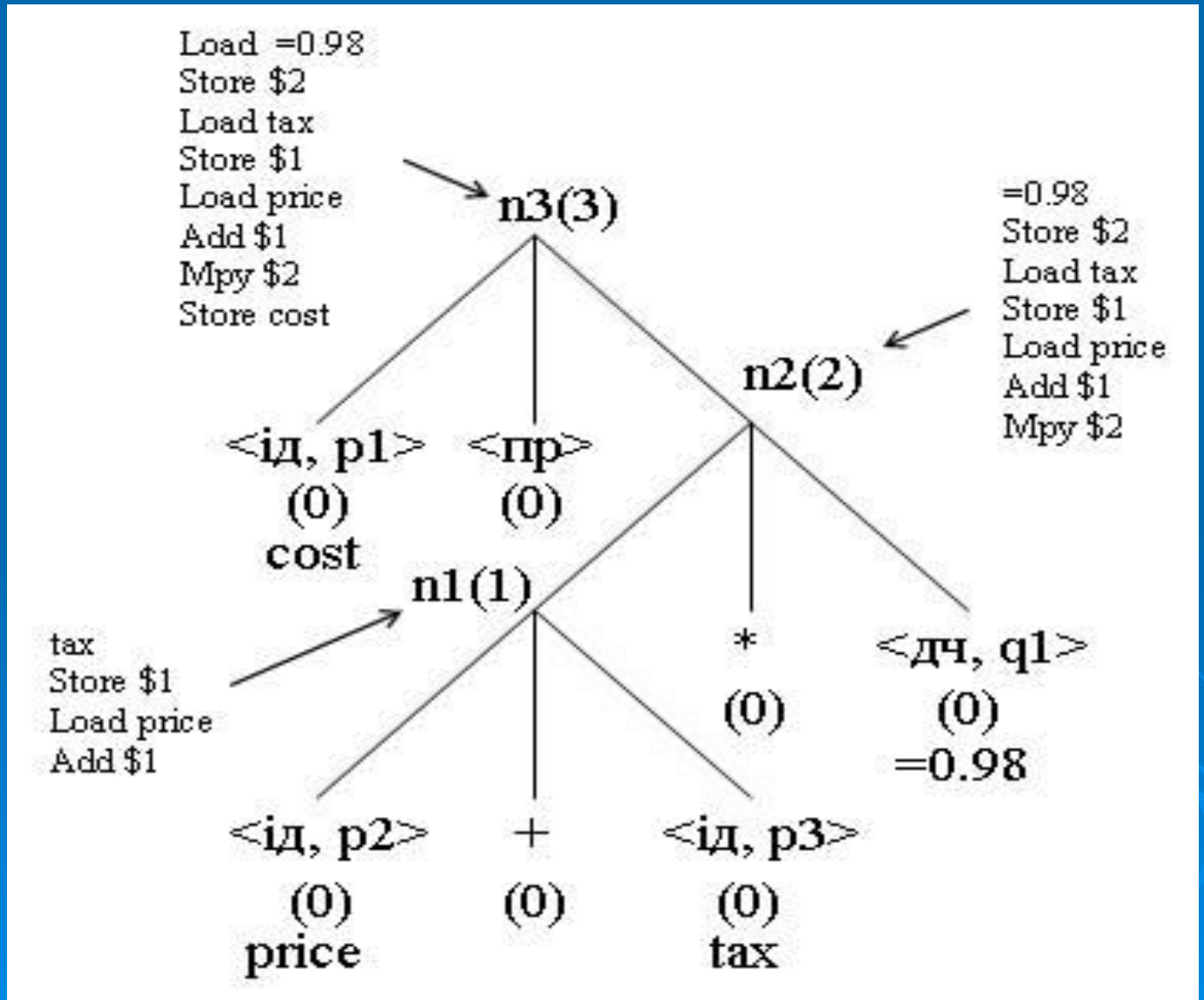
$$C(n) \rightarrow C(m3)$$
$$Store\ \$l(n)$$
$$Load\ C(m1)$$
$$Mpy\ \$l(n)$$

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



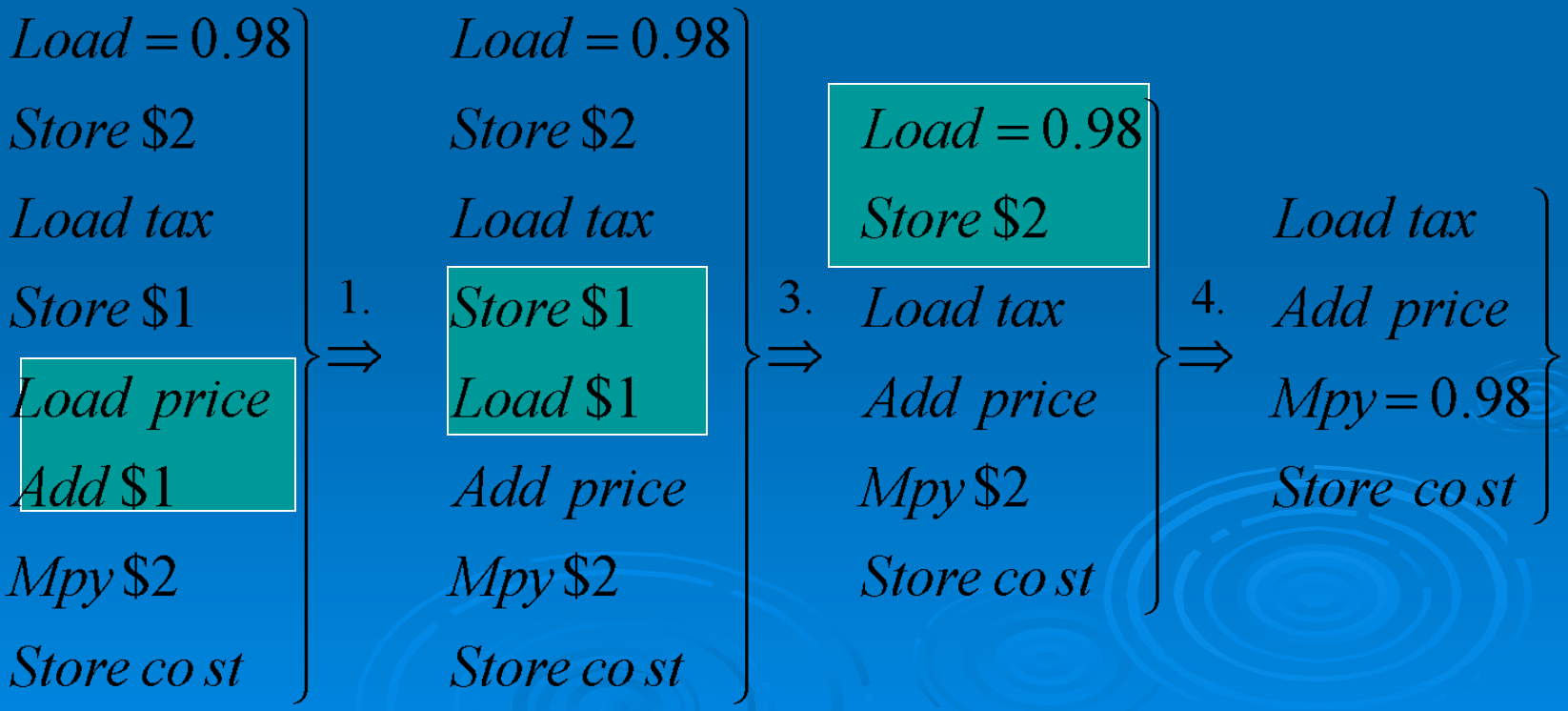
2.5. Оптимізація проміжного коду

1) $\left.\begin{matrix} Load\ a \\ Add\ b \end{matrix}\right\} \Leftrightarrow \left\{\begin{matrix} Load\ b \\ Add\ a \end{matrix}\right.$ операція «+» є комутативною в тому випадку, коли на *Add b* не передавалось управління.

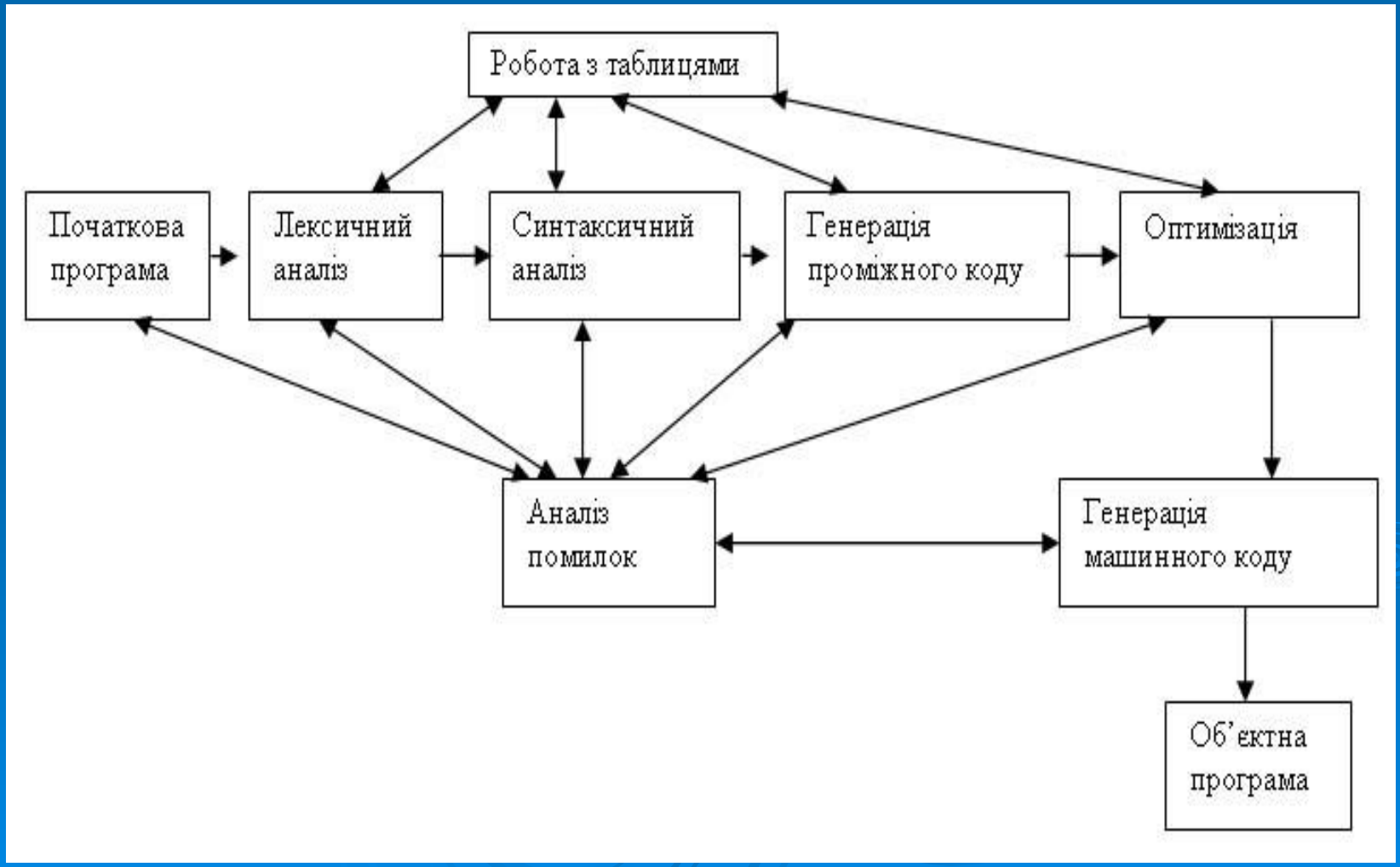
- 2) $\left.\begin{matrix} Load\ a \\ Mpy\ b \end{matrix}\right\} \Leftrightarrow \left\{\begin{matrix} Load\ b \\ Mpy\ a \end{matrix}\right.$ операція «*» є комутативною .
- 3) $\begin{matrix} Store\ a \\ Load\ a \end{matrix}$ можна вилучити при умові, що надалі комірка *a* не буде використовуватись або буде заповнена безпосередньо перед використанням.
- 4) $\begin{matrix} Load\ a \\ Store\ b \end{matrix}$ 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

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.

Тема 2