Generalized linked list

Generalized Lists

- A generalized list, A, is a finite sequence of $n \geq 0$ elements, a_0, a_1, a_2, …, a_{n-1}, where a_i, is either an atom or a list. The elements a_i, $0 \leq i \leq n-1$, that are not atoms are said to be the sublists of A.
- A list A is written as $A = (a_0, \ldots, a_{n-1})$, and the length of the list is n.
- A list name is represented by a capital letter and an atom is represented by a lowercase letter.
- a_0 is the head of list A and the rest $(a_1, \ldots, a_{n-1})$ is the tail of list A.

Examples of Generalized Lists

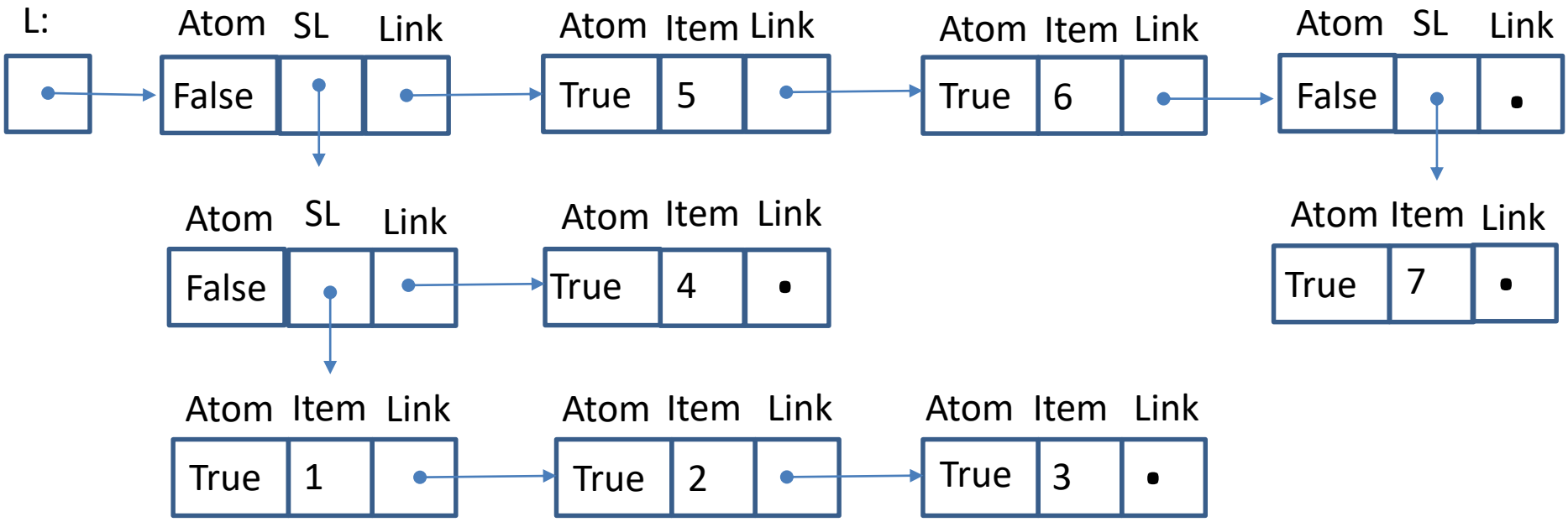
- A = ( ): the null, or empty, list; its length is zero.
- B = (a, (b, c)): a list of length two; its first element is the atom a, and its second element is the linear list (b, c).
- C = (B, B, ( )): A list of length three whose first two elements are the list B, and the third element is the null list.
- D = (a, D): is a recursive list of length two; D corresponds to the infinite list D = (a, (a, (a, …))).
- head(B) = 'a' and tail(B) = (b, c), head(tail(C))=B and tail(tail(C)) = ( ).
- Lists may be shared by other lists.
- Lists may be recursive.

Generalized Lists

- A **generalized list** is a list in which the individual list items are permitted to be sublists.
- **Example**: $(a_1, a_2, (b_1, (c_1, c_2), b_3), a_4, (d_1, d_2), a_6)$
- If a list item is not a sublist, it is said to be **atomic**.
- Generalized lists can be represented by sequential or linked representations.

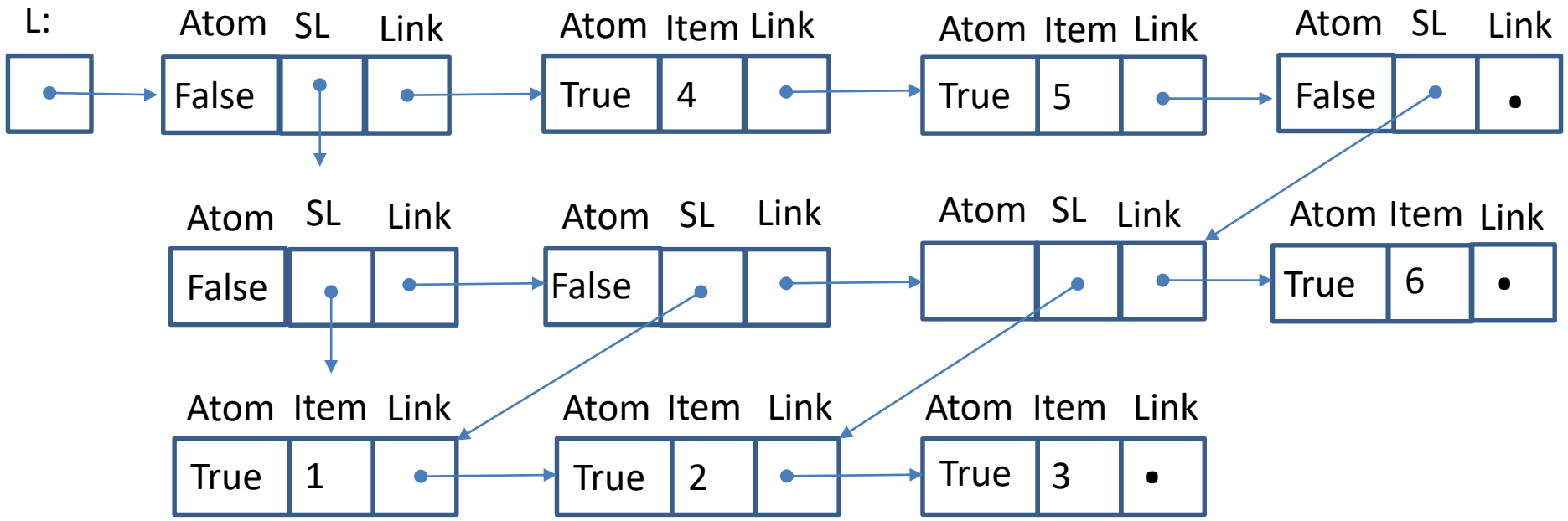
Generalized Lists (cont'd)

- The generalized list *L=(((1,2,3), 4), 5, 6, (7))* can be represented without shared sublists as follows:



Generalized Lists (cont'd)

- The generalized list *L=(((1, 2, 3), (1, 2, 3), (2, 3), 6), 4, 5, ((2, 3), 6))* can be represented with shared sublists as follows:



Generalized Lists

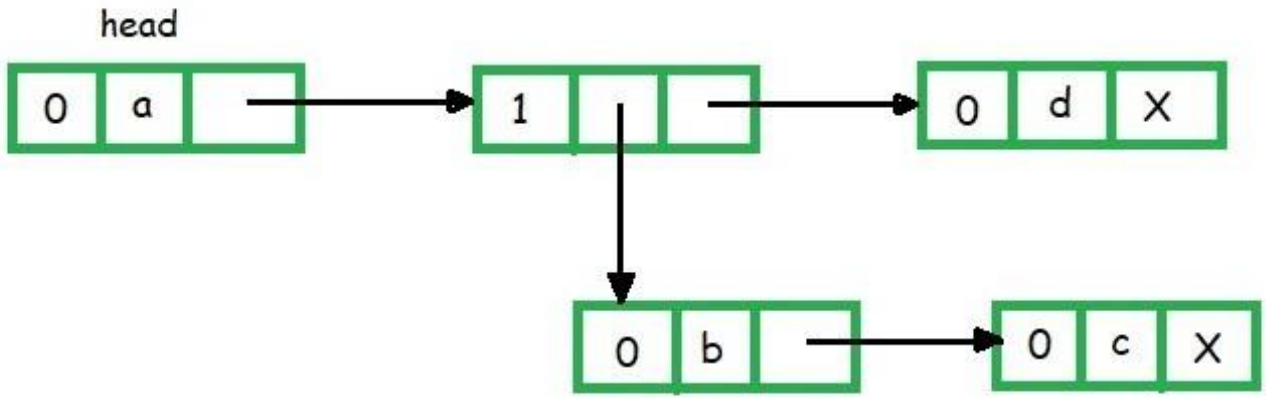
To represent a list of items there are certain assumptions about the node structure.

- Flag = 1 implies that *down pointer* exists (sublist)
- Flag = 0 implies that *next pointer* exists(atom)
- Data means the atom
- Down pointer is the address of node which is down of the current node
- Next pointer is the address of node which is attached as the next node

```
typedef struct node {
    char c;                     //Data
    int index;                  //Flag
    struct node *next, *down;    //Next & Down pointer
}GLL;
```

Example of GLL {List representation}

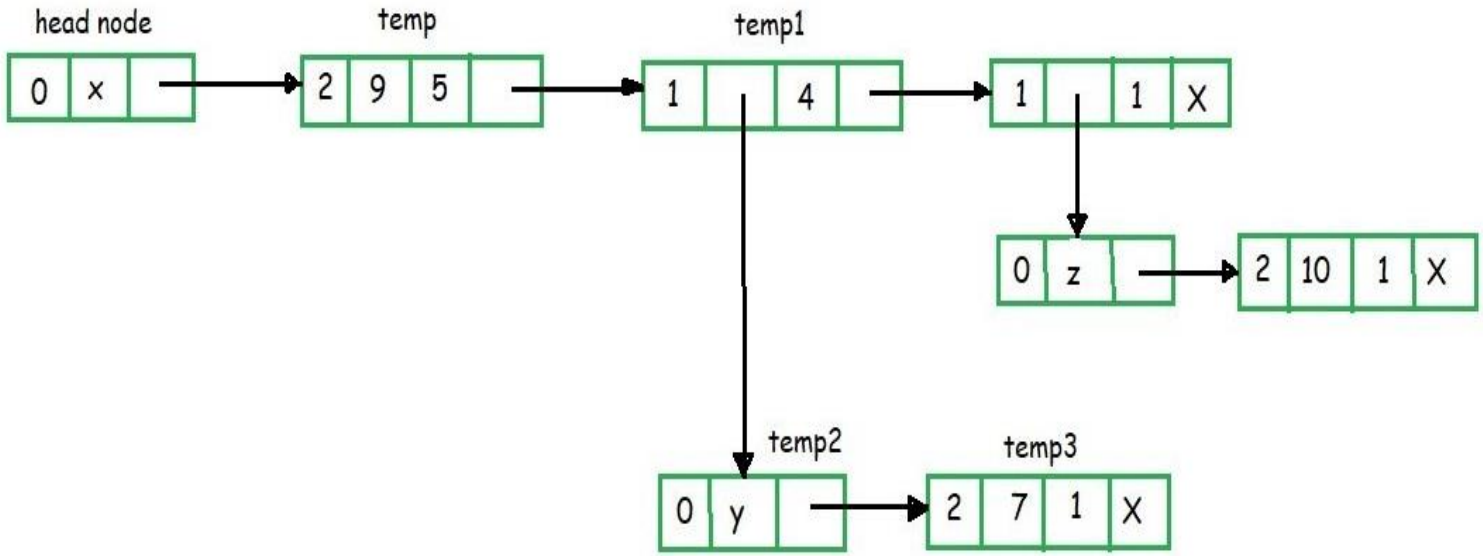
(a, (b, c), d)



Polynomial Representation using Generalized Linked List

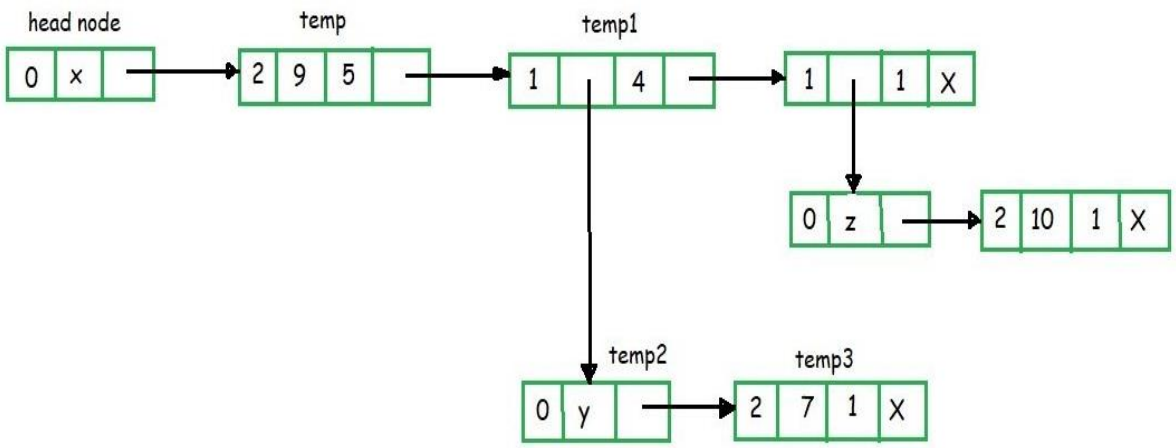
- Flag = 0 means *variable* is present
- Flag = 1 means *down pointer* is present
- Flag = 2 means *coefficient* and *exponent* is present

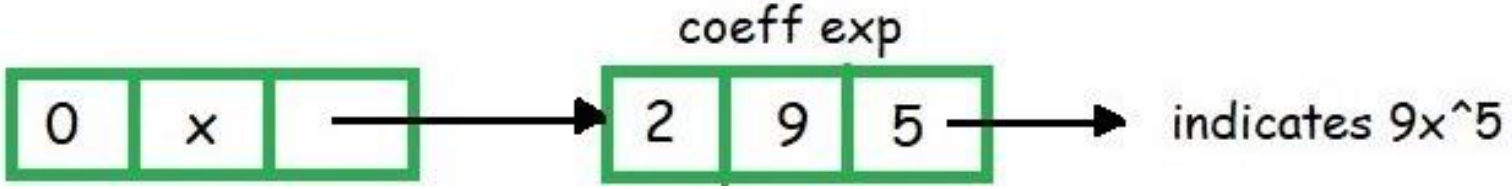
Example: $9x^5 + 7x^4y + 10xz$



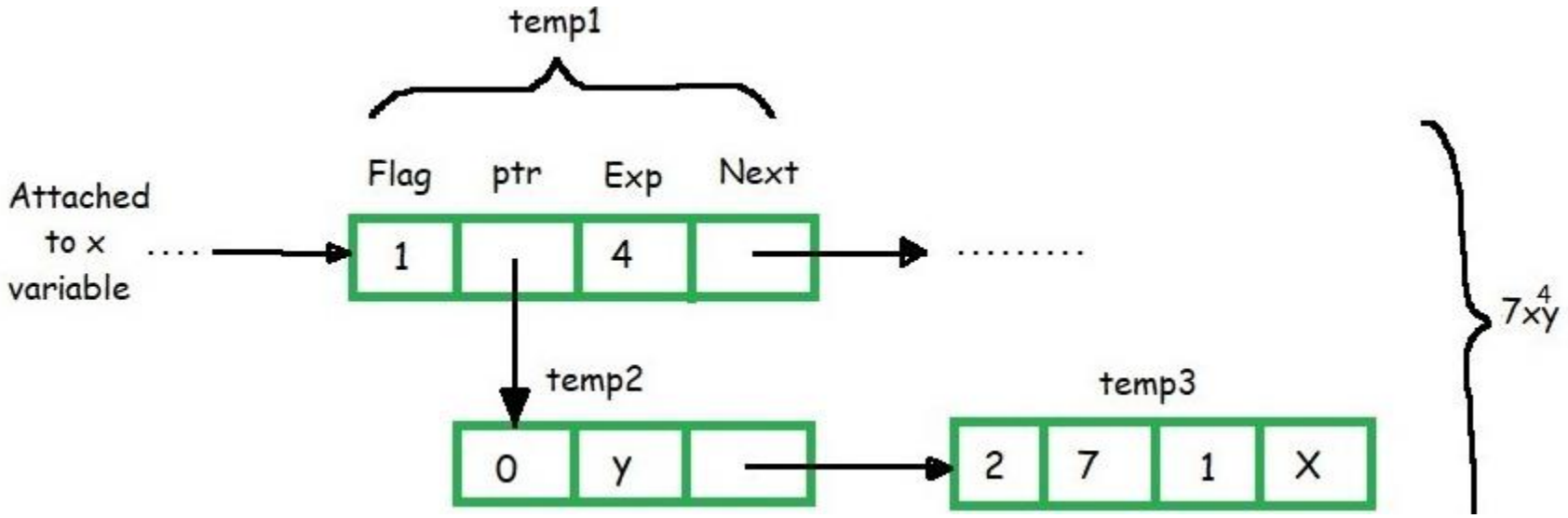
Example: $9x^5 + 7xy^4 + 10xz$

In this example, the head node is of variable x. The temp node shows the first field as 2 means coefficient and exponent are present.





As the temp node is attached to head node and head node is having variable x, temp node having coefficient=9 and exponent=5.
The above two nodes can be read as $9x^5$.



The node temp1 can be read as x^4.

- The flag field is 1 means down pointer is there
- temp2 = y
- temp3 = coefficient = 7
- exponent = 1
- flag = 2 means the node contains coefficient and exponent values.
- temp2 is attached to temp3 this means $7y_1$ and temp2 is also attached to temp1 means
- temp1 x temp2
- x^4 x $7y^1$ = $7x^4y^1$ value is represented by above figure

Printing Generalized Lists

```
void PrintList(GenListNode *L)
{
      GenListNode *G;

      printf("(");
      G=L;
      while (G != NULL){
            if (G->Atom){
               printf("%d", G->SubNode.Item);
            } else {
               printList(G->SubNode.SubList);
            }
            if (G->Link != NULL) printf(",");
            G=G->Link;
      }
      printf(")");
}
```

Generalized List Application Example

$$p(x, y, z) = x^{10}y^3z^2 + 2x^8y^3z^2 + 3x^8y^2z^2 + x^4y^4z + 6x^3y^4z + 2yz$$

- Consider the polynomial $P(x, y, z)$ with various variables. It is obvious the sequential representation is not suitable to this.
- What if a linear list is used?
  - The size of the node will vary in size, causing problems in storage management.
- Let's try the generalized list.

Generalized List Application Example

- $P(x, y, z)$ can be rewritten as follows:

$$((x^{10} + 2x^8)y^3 + 3x^8y^2)z^2 + ((x^4 + 6x^3)y^4 + 2y)z$$

- The above can be written as $Cz^2 + Dz$. Both C and D are polynomials themselves but with variables x and y only.
- If we look at polynomial C only, it is actually of the form $Ey^3 + Fy^2$, where E and F are polynomial of x only.
- Continuing this way, every polynomial consists of a variable plus coefficient-exponent pairs. Each coefficient is itself a polynomial.

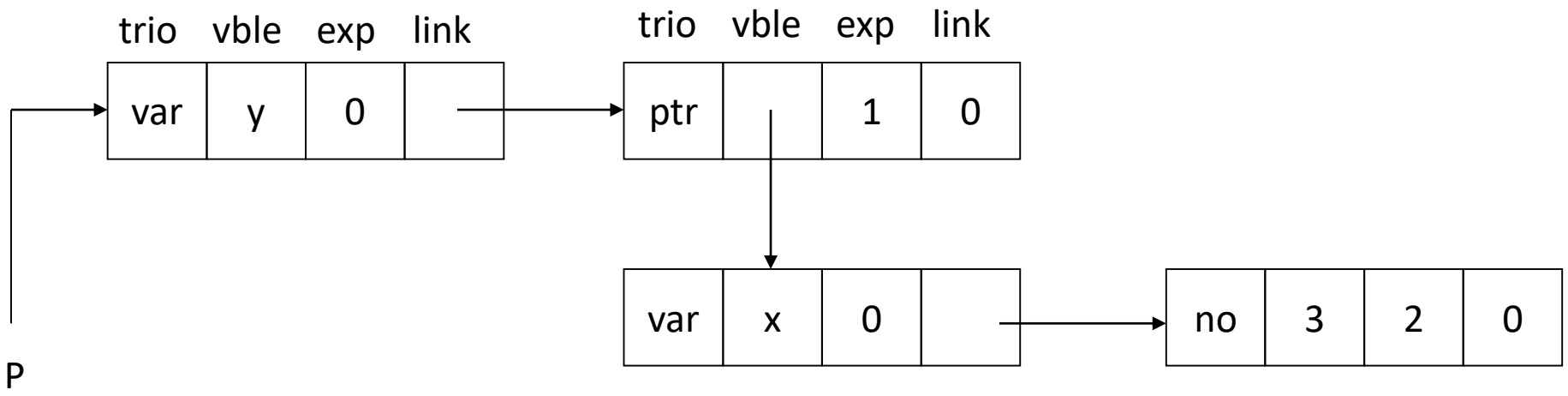
PolyNode structure

```
enum Triple{ var, ptr, no };
Struct  PolyNode
{
    PolyNode *link;
    int exp;
    Triple trio;
    union {
            char vble;
            PolyNode *dlink;
            int coef;
    };
};
```

PolyNode in C

- trio == var: the node is a head node.
  - vble indicates the name of the variable. Or it is an integer point to the variable in a variable table.
  - exp is set to 0.
- trio == ptr: coefficient itself is a list and is pointed by the field dlink. exp is the exponent of the variable on which the list is based on.
- trio == no, coefficient is an integer and is stored in coef. exp is the exponent of the variable on which the list is based on.

Representing $3x^2y$



Representation of $P(x, y, z)$