บทที่ 4

ผลการทดลอง

4.1 ตัวอย่างโปรแกรมที่ใช้ศึกษา

ตัวอย่างโปรแกรมที่ใช้ศึกษา ได้แก่ โปรแกรมโครงสร้างที่พัฒนาด้วยภาษาซีที่มีลักษณะต่างกัน 4 ลักษณะ คือ

1. โปรแกรมสัตว์เลี้ยง เป็นโปรแกรมที่มีลักษณะการเขียนแบบสืบทอดคุณสมบัติ ภายในโปรแกรมมีสตรักท์ 4 โครงสร้าง และฟังก์ชัน 10 ฟังก์ชัน ทั้งนี้เพื่อแสดงว่าเทคนิคฟัซซี่ เออาร์ทีสามารถใช้จัดกลุ่มข้อมูลที่มีลักษณะของการเขียนโปรแกรมเชิงโครงสร้างแบบสืบทอดคุณสมบัติได้ ดังแสดงในรูปที่ 4.1

```
#include "stdafx.h"
#include "stdio.h"
#include "string.h"

typedef struct animal
{
  char name[10];
  int weight;
}Animal;

typedef struct Cat
{
  Animal cat;
  char noise[20];
  char action[20];
  bool climb;
}Meaw;
```

```
char*  getAction ( )
{
  char* act;
  printf("What action is your animal can do
well");
  gets(act);
  return(act);
}

bool canClimbTree (Meaw cat)
{
   bool c;
   char act[20];
   strcpy(act, cat.action);
     if (strcmp(act, "climb") == 0)
          c = true;
          else
                    c = false;
          return (c);
}
```

ภาพที่ 4.1 โปรแกรมสัตว์เลี้ยง

```
typedef struct Dog
{
  Animal dog;
  char noise[20};
  char action[20];
}Woof;

typedef struct Cow
{
  Animal cow;
  char noise[20];
  char action[20];
}Moo;

char* getName ( )
{
  char nm[10];
  printf("Please enter your animal's Name");
  gets(nm);
  return(nm);
}

int getWeight ( )
{
  int wei;
  printf("Please enter your animal's Weight");
  scanf("%d",&wei);
  return(wei);
}

void setCatInfo(Meaw meaw)
{
  printf("Your animal name
is",meaw.cat.name);
  printf("Your animal weight
is",meaw.cat.weight);
  printf("His noise is",meaw.noise);
  printf("And his action is",meaw.action);
  if (meaw.climb == true)
     printf("Wow !! your cat can climb tree.");
  else
     printf("Sorry!! Your cat can't climb
tree.");
}
```

```
void getCatInfo (Meaw meaw )
{
  strcpy(meaw.cat.name ,getName( ));
  meaw.cat.weight = getWeight( );
  strcpy(meaw.noise , "Meaw Meaw");
  strcpy(meaw.action , getAction( ));
  meaw.climb = canClimbTree(meaw);
}

void getDogInfo (Woof woof )
{
  strcpy(woof.dog.name , getName( ));
  woof.dog.weight = getWeight( );
  strcpy(woof.noise , "Woof Woof");
  strcpy(woof.action , getAction( ));
}

void getCowInfo (Moo moo )
{
  strcpy(moo.cow.name , getName( ));
  moo.cow.weight = getWeight( );
  strcpy(moo.noise , "Moo Moo");
  strcpy(moo.action , getAction( ));
}

void setDogInfo(Woof woof)
{
  printf("Your animal name
is",woof.dog.name);
  printf("Your animal weight
is",woof.dog.weight);
  printf("His noise is",woof.noise);
  printf("And his action is",woof.action);
}

void setCowInfo(Moo moo)
{
  printf("Your animal name is",moo.cow.name);
  printf("Your animal weight
is",moo.cow.weight);
  printf("His noise is",moo.noise);
  printf("And his action is",moo.action);
}
```

ภาพที่ 4.1 (ต่อ)

2. โปรแกรมแถวลำดับ (Stack) และ แถวคอย (Queue) แบบส่วนจำเพาะ (Modular Case) (สมศักดิ์ ภัทรศุกล, 2545 :28) เพื่อทำการเปรียบเทียบผลที่ได้จากการทดลองในครั้งนี้กับงานวิจัยที่ได้ตีพิมพ์ ภายในโปรแกรมจะมีสตรักท์ 2 โครงสร้างและฟังก์ชัน 8 ฟังก์ชัน ซึ่งเรียกโครงสร้างแบบนี้ว่าเป็นแบบส่วนจำเพาะ (Modular Case) เพราะว่าการทำงานของแต่ละฟังก์ชันมีขอบเขตการทำงานที่ชัดเจนและไม่สามารถเรียกใช้ค่าจากส่วนข้อมูลอื่นที่ไม่เกี่ยวข้องได้โดยตรง ดังแสดงในภาพที่ 4.2

```
#define QUEUE_SIZE 10
struct stack
{    int *base, *sp, size;  };
struct queue
{
  struct stack *front, *back;
};
struct stack* initStack (int sz)
{
  struct stack* s = (struct stack*)
  malloc(sizeof(struct stack));
  s->base = s->sp =
    (int*)malloc(sz*(sizeof(int)));
  s->size = sz;
  return s;
}
struct queue initQ ()
{
  struct queue* q = (struct queue*)
  malloc(sizeof(struct queue));
  q->front = initStack(QUEUE_SIZE);
  q->back = initStack(QUEUE_SIZE);
  return q;
}
int isEmptyStack (struct stack* s)
{
  return (s->sp == s->base);
}
int isEmptyQueue (struct queue* q)
{
  return (isEmptyStack(q->front) &&
  isEmptyStack(q->back));
}
void push (struct stack* s, int i)
{
  /*no stack overflow check*/
  *(s->sp) = i;
  s->sp++;
}
void enQueue (struct queue* q, int i)
{  push(q->front, i); }
int pop (struct stack* s)
{
  if (isEmptyStack(s)) return -1;
  s->sp--;
  return (*(s->sp));
}
int  deQueue (struct queue* q)
{
  if  isEmptyQ (q))
    return -1;
  if  isEmptyStack (q-> back))
    while (! isEmptyStack(q -> front))
      push (q -> back , pop (q -> front));
  return pop (q -> back);
}
```

ภาพที่ 4.2 โปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะ

3. โปรแกรมแถวลำดับ (Stack) และ แถวคอย (Queue) แบบผูกติด (Tangled Case) (สมศักดิ์ ภัทรศุกล, 2545 :29) เพื่อทำการเปรียบเทียบผลที่ได้จากการทดลองในครั้งนี้กับงานวิจัยที่ได้ตีพิมพ์ ภายในโปรแกรมจะมีสตรักท์ 2 โครงสร้างและฟังก์ชัน 8 ฟังก์ชัน รายละเอียดส่วนใหญ่เหมือนกับโปรแกรมแรก ยกเว้นฟังก์ชัน isEmptyQueue และ enQueue ที่ถูกเปลี่ยนให้ไปเรียกใช้ค่าจากแถวลำดับโดยตรง แทนการเรียกโดยอ้อมเหมือนในโปรแกรมแรก โดยฟังก์ชันทั้ง

สองเป็นฟังก์ชันของแถวคอยที่มีความเกี่ยวเนื่องกับส่วนข้อมูลของแถวลำดับ จึงเรียกโครงสร้างแบบนี้ว่าเป็นแบบผูกติด ดังแสดงในภาพที่ 4.3

```
int isEmptyQueue (struct queue* q)
{
  return(q->front->sp==q->front->base
  && q->back->sp == q->back->base);
}
```

```
void enQueue(struct queue* q, int i)
{
  *(q->front->sp) = i;
  q->front->sp++;
}
```

ภาพที่ 4.3 รายละเอียดของโปรแกรมแถวลำดับและแถวคอยแบบผูกติดที่ถูกแก้ไข

4. โปรแกรมการลงทะเบียนของพนักงาน (สมศักดิ์ ภัทรศุกล, 2545 : 30) เพื่อทำการเปรียบเทียบผลที่ได้จากการทดลองในครั้งนี้กับงานวิจัยที่ได้ตีพิมพ์ ภายในโปรแกรมมีสตรักท์ 2 โครงสร้าง ตัวแปรโกบอล 3 ตัว และฟังก์ชัน 10 ฟังก์ชัน ซึ่งเป็นโปรแกรมที่มีส่วนข้อมูลมากกว่าหนึ่งแบบ ดังแสดงในภาพที่ 4.4

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

int  max_pos, cur_pos;

struct  list
{
   struct employee *data;
   struct  list  *next, *back;
};
struct  list  *head_ptr;

struct employee
{
  char  *name;
  int age, salary;
};
void  add_pos (int i)
{
  max_pos = max_pos + i;
}
```

```
int clear_pos (int i)
{
  if (max_pos <= i)
    return -1;
  else
    return  max_pos – i;
}
int  check_pos()
{
  int  avai_pos = max_pos - cur_pos;
  if (avai_pos > 0)
     return  avai_pos;
  else
     return  -1 ;
}
void  init()
{
 head_ptr = (struct  list*) malloc(sizeof(struct
list));
 head_ptr ->data = NULL;
 head_ptr -> next = NULL;
 head_ptr -> back = NULL;
}
```

ภาพที่ 4.4 โปรแกรมการลงทะเบียนของพนักงาน

```
void add (struct  employee *e)
{
 struct  list *new_list;
 struct  list  *next_list;

 new_list = (struct  list*) malloc(sizeof(struct
list));
 new_list -> data = e;
 new_list -> next = head_ptr -> next;
 new_list -> back = head_ptr;

 head_ptr -> next = new_list;
 next_list = new_list -> next;
 if (next_list != NULL)
    next_list -> back = new_list;
}

void  del(struct  list *del_list)
{
 struct  list *back_list;
 struct  list *next_list;
 back_list = del_list -> back;
 next_list = del_list -> next;
 back_list -> next – next_list;
 if (next_list != NULL)
  next_list -> back = back_list;
}
int enroll (char *n, int a, int s)
{
 struct  employee  *new_emp;
 if (check_pos > 0)  {
    new_emp = (struct  employee*)
malloc(sizeof(struct  employee));
    new_emp -> name = n;
    new_emp -> age = a;
    new_emp -> salary = s;
    add(new_emp);
    printf("add: %s%d%d\n",new_emp -> name,
new_emp -> age, new_emp ->salary);
    cur_pos++;
    return 1;
  }  else  {
     printf ("Position  is  full");
     return -1;   }
}
```

```
struct  list* search (char* n)
{
 struct  list *cur_ptr = head_ptr -> nex;
 while (cur_ptr != NULL &&
strcmp(cur_ptr -> data -> name, n) != 0)
  cur_ptr = cur_ptr -> next;
  printf("found: %s\n",cur_ptr -> data ->
name);
  return (cur_ptr);
}

int dismiss (char* n)
{
 struct list *del_list = (struct  list*)
search(n);
 if (cur_pos > 1) {
  if (del_list != NULL) {
    del(del_list);
    cur_pos--;
    return 1;
  } else  {
      printf("Warning: employee  not
found");
      return -1;}
  else {
    printf("Warning: only  one employee
left");
    return -1;  }
}
void  print()
{
 struct  list *cur_ptr = head_ptr -> next;
 while (cur_ptr != NULL) {
   printf("print:%s%d%d\n",cur_ptr -> data
-> name,cur_ptr -> data ->age ,cur_ptr ->
data -> salary);
  cur_ptr = cur_ptr -> next;
 }
}
```

ภาพที่ 4.4 (ต่อ)

4.2 การวัดผลการทดลอง

การวัดผลการทดลอง ทำได้โดยกำหนดกลุ่มส่วนคำสั่งที่ต้องการไว้ล่วงหน้า ก่อนที่จะนำวิธีการต่าง ๆ มาทดลองใช้ เมื่อตรวจผลการทดลองแล้วพบว่ามีกลุ่มส่วนคำสั่งดังกล่าวปรากฏอยู่ในผลลัพธ์ของการจัดกลุ่ม ให้ถือว่าวิธีการนั้น ๆ ทำงานได้ผล แต่ถ้าไม่พบก็ให้ถือว่าวิธีการทำงานนั้นๆ ทำงานผิดพลาด

โดยวิทยานิพนธ์ฉบับนี้ได้ใช้เกณฑ์วัดผลการทดลองเช่นเดียวกับ (สมศักดิ์ ภัทรศุกล, 2545 : 31-32) เพราะว่าต้องการเปรียบเทียบผลที่ได้จากการทดลองนี้กับผลการทดลองในงานวิจัยฉบับดังกล่าว

กลุ่มส่วนคำสั่งที่ถูกกำหนดไว้ล่วงหน้า หรือที่เรียกว่า กลุ่มสมบูรณ์ (Complete Module) หมายถึงกลุ่มส่วนคำสั่งที่สามารถสื่อถึงโอเปอเรชันของสิ่งใดสิ่งหนึ่งได้ เช่น กลุ่มส่วนคำสั่ง initStack isEmptyStack push และ pop เป็นกลุ่มสมบูรณ์ เพราะการรวมตัวกันของส่วนคำสั่งเหล่านี้สามารถสื่อถึงโอเปอเรชันของแถวลำดับได้ ในทางตรงกันข้ามกลุ่มส่วนคำสั่ง initStack isEmptyStack enQueue และ deQueue ไม่จัดว่าเป็นกลุ่มสมบูรณ์ เพราะการรวมตัวกันของส่วนคำสั่งเหล่านี้ไม่สามารถสื่อถึงโอเปอเรชันของสิ่งใดได้ ไม่ว่าจะเป็นโอเปอเรชันของแถวลำดับหรือแถวคอย การกำหนดกลุ่มสมบูรณ์ให้กับโปรแกรมที่ใช้ศึกษา มีรายละเอียดและเหตุผลดังนี้

4.2.1 กลุ่มสมบูรณ์ของโปรแกรมที่มีลักษณะการเขียนแบบสืบทอดคุณสมบัติ

เมื่อพิจารณาส่วนคำสั่งภายในโปรแกรม พบว่า โปรแกรมสัตว์เลี้ยง มีลักษณะการเขียนแบบสืบทอดคุณสมบัติ ส่วนคำสั่งมีการแบ่งออกเป็น 4 กลุ่มอย่างชัดเจน ซึ่งได้แก่ กลุ่มของส่วนคำสั่งที่ถูกเรียกใช้จากส่วนคำสั่งอื่น โดยมี 3 ส่วนคำสั่งที่ถูกเรียกใช้จากส่วนคำสั่งต่าง ๆ นอกจากนี้ส่วนคำสั่งต่าง ๆ ก็ยังสามารถแบ่งออกเป็นกลุ่ม ๆ ได้อย่างชัดเจน

ด้วยเหตุนี้จึงได้กำหนดให้ กลุ่มสมบูรณ์ของโปรแกรมลงสัตว์เลี้ยง มี 4 กลุ่ม โดยมีกลุ่มหนึ่งเป็นกลุ่มของฟังก์ชันที่ถูกเรียกใช้งานจากฟังก์ชันอื่นและอีก 3 กลุ่มที่เหลือนั้นจะแยกตามประเภทของสัตว์ คือ 1) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ getName getWeight และ getAction 2) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ getCatInfo setCatInfo และ CanClimbTree 3) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ getDogInfo และ setDogInfo 4) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ getCowInfo และ setCowInfo

4.2.2 กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะ

เมื่อพิจารณาส่วนคำสั่งภายในโปรแกรม พบว่า ส่วนคำสั่งมีการแบ่งออกเป็น 2 กลุ่มอย่างชัดเจน ซึ่งได้แก่ กลุ่มที่เกี่ยวข้องกับส่วนข้อมูล STACK และกลุ่มที่เกี่ยวข้องกับส่วนข้อมูล QUEUE เมื่อนำส่วนคำสั่งทั้งสองกลุ่มมาพิจารณาความหมาย จะพบว่าการรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล STACK สามารถสื่อถึงโอเปอเรชันของแถวลำดับ และการรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล QUEUE สามารถสื่อได้ถึงโอเปอเรชันของแถวคอย

ด้วยเหตุนี้จึงได้กำหนดให้ กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะ มี 2 กลุ่ม คือ 1) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ initStack isEmptyStack push และ pop 2) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ initQueue isEmptyQueue enQueue และ deQueue ดังแสดงในภาพที่ 4.5

4.2.3 กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด

เมื่อพิจารณาส่วนคำสั่งภายในโปรแกรม พบว่า ส่วนคำสั่งกระจายออกเป็น 2 กลุ่ม ได้แก่ กลุ่มที่เกี่ยวข้องกับส่วนข้อมูล STACK และกลุ่มที่เกี่ยวข้องกับส่วนข้อมูล QUEUE ซึ่งการกระจายตัวของส่วนคำสั่งในกรณีนี้ไม่ชัดเจนเท่ากับกรณีแรก เพราะมีการเรียกใช้ส่วนคำสั่งแถวคอยไปยังส่วนข้อมูลแถวลำดับ เมื่อนำทั้งสองกลุ่มมาพิจารณาความหมาย พบว่า การรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล STACK ยกเว้นส่วนคำสั่ง isEmptyQueue และ enQueue สามารถสื่อถึงโอเปอเรชันของแถวลำดับ และ การรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล QUEUE สามารถสื่อถึงโอเปอเรชันของแถวคอย

ด้วยเหตุนี้จึงได้กำหนดให้ กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด มี 2 กลุ่ม คือ 1) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ initStack isEmptyStack push และ pop 2) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ initQueue isEmptyQueue enQueue และ deQueue ดังแสดงในภาพที่ 4.5

Stack
int *base int *sp int size
struct stack *initStack (int sz) int isEmptyStack (struct stack *s) void push (struct stack *s , int i) int pop (struct stack *s)

Queue
int *base int *sp int size stack *front stack *back
struct queue *initQ ( ) int isEmptyQueue (struct queue *q) void enQueue (struct stack *q , int i) int deQueue (struct queue *q)

ภาพที่ 4.5 กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับและแถวคอย แบบส่วนจำเพาะและแบบผูกติด

4.2.4 กลุ่มสมบูรณ์ของโปรแกรมการลงทะเบียนของพนักงาน

เมื่อพิจารณาส่วนคำสั่งภายในโปรแกรม พบว่า โปรแกรมการลงทะเบียนของพนักงานมีความซับซ้อนมากกว่า 2 โปรแกรมแรก ทำให้การกระจายตัวของส่วนคำสั่งไม่ชัดเจนเหมือนกับสองโปรแกรมแรก แต่ก็สามารถมองเห็นว่าส่วนคำสั่งดังกล่าวกระจายตัวออกเป็นสองกลุ่มใหญ่ ซึ่งก็คือ กลุ่มที่เกี่ยวข้องกับส่วนข้อมูล max_pos กับ cur_pos โดยสามารถสื่อถึง โอเปอเรชันการจัดการจำนวนพนักงาน และกลุ่มที่เกี่ยวข้องกับส่วนข้อมูล head_ptr employee กับ list ที่สามารถสื่อถึงโอเปอเรชันการจัดการข้อมูลพนักงานในลิสต์ได้

ด้วยเหตุนี้จึงได้กำหนดให้ กลุ่มสมบูรณ์ของโปรแกรมลงทะเบียนพนักงาน มี 2 กลุ่ม คือ 1) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ add_pos clear_pos check_pos enroll และ dismiss 2) กลุ่มที่ประกอบด้วยส่วนคำสั่งของ init add del search และ print ดังแสดงในภาพที่ 4.6

Manage_position
int max_pos int cur_pos
void add_pos (int i) int clear_pos (int i) int check_pos ( ) int enroll (char *n , int a , int s) int dismiss (char *n)

Manage_general
char *name int age int salary list *head_ptr list *next list *back
void add (struct employee *e) void init ( ) struct list *search (char *n) void print ( ) void del (struct list *del_list)

ภาพที่ 4.6 กลุ่มสมบูรณ์ของโปรแกรมการลงทะเบียนของพนักงาน

4.3 ผลการทดลอง

4.3.1 ผลการคัดแยกส่วนประกอบเดิม

ผลที่ได้จากการทำงานในขั้นตอนที่ 1 แยกฟังก์ชันของโปรแกรมเดิมที่เราต้องการศึกษาจากตัวอย่างโปรแกรม คือ รายการส่วนประกอบเดิม (Artifact List) ที่คัดแยกได้ ซึ่งก็คือ รายการส่วนประกอบเดิมของโปรแกรมสัตว์เลี้ยง รายการส่วนประกอบเดิมของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะ รายการส่วนประกอบเดิมของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด และ รายการส่วนประกอบเดิมของโปรแกรมการลงทะเบียนพนักงาน ตามตารางที่ 4.1 4.2 และ 4.3 ตามลำดับ

ตารางที่ 4.1 รายการส่วนประกอบเดิมของโปรแกรมสัตว์เลี้ยง

Software Component	Abbreviate
struct Animal	Animal
struct Meaw	Meaw
Struct Woof	Woof
struct Moo	Moo
void char getName ( )	getN
void int getWeight ( )	getW
void getCatInfo ( )	getCI
void getDogInfo ( )	getDoI
void getCowInfo	getCoI
char getAction ( )	getA
boolean canClimbTree (Meaw cat)	canCli
void setCatInfo (Meaw meaw)	setCI
void setDogInfo (Woof woof)	setDoI
void setCowInfo (Moo moo)	setCoI

ตารางที่ 4.2 รายการส่วนประกอบเดิมของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะและแบบผูกติด

Software Component	Abbreviate
struct stack	Stack
struct queue	Queue
struct stack* initStack(int sz)	initS
struct queue* initQ()	initQ
int isEmptyStack(struct stack* s)	isEmpS
int isEmptyQueue (struct queue* q)	isEmpQ
void push (struct stack* s, int i)	push

ตารางที่ 4.2 (ต่อ)

Software Component	Abbreviate
void enQueue (struct stack* q, int i)	enQ
int pop (struct stack* s)	pop
int deQueue (struct queue* q)	deQ

ตารางที่ 4.3 รายการส่วนประกอบเดิมของโปรแกรมการลงทะเบียนพนักงาน

Software Component	Abbreviate
variable max_pos	max_pos
variable cur_pos	cur_pos
variable head_ptr	head_ptr
struct list	list
struct employee	employee
void add_pos (int i)	add_pos
int clear_pos (int i)	clr_pos
int check_pos	chk_pos
int enroll (char *n, int a, int s)	enroll
int dismiss (char *n)	dismiss
void init	init
void add	add
void del	del
struct list* search	search
void print	print

4.3.2 ผลการหาความสัมพันธ์ระหว่างส่วนประกอบเดิมที่ดัดแยกได้

เนื่องจากในขั้นตอนนี้ ได้แบ่งขั้นตอนการทำงานออกเป็น 2 ขั้นตอนย่อย คือ 1) กำหนดนิยามความสัมพันธ์ 2) ตรวจสอบความสัมพันธ์ ดังนั้นจึงได้แบ่งการแสดงผลออกเป็น 2 ส่วน ซึ่งมีรายละเอียด ดังนี้

4.3.2.1 ผลการกำหนดนิยามความสัมพันธ์

ผลที่ได้จากการทำงานขั้นตอนนี้ คือ นิยามความสัมพันธ์เฉพาะเจาะจงที่เกิดจากการนำนิยามความสัมพันธ์ทั่วไป 3 แบบ ซึ่งได้แก่ 1) ความสัมพันธ์แบบใช้เป็นค่าส่งออก (Return-Type Relationship) 2) ความสัมพันธ์แบบใช้เป็นค่าส่งเข้า (Argument Relationship) 3) ความสัมพันธ์แบบเรียกใช้ (Use Relationship) มาประยุกต์ใช้กับส่วนข้อมูล ผลลัพธ์ที่ได้ คือ นิยามความความสัมพันธ์เฉพาะเจาะจงของโปรแกรมสัตว์เลี้ยง นิยามความความสัมพันธ์เฉพาะเจาะจงของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะและแบบผูกติด และนิยามความความสัมพันธ์เฉพาะเจาะจงของโปรแกรมการลงทะเบียนของพนักงาน ตามตารางที่ 4.4 4.5 และ 4.6 ตามลำดับ

ตารางที่ 4.4 นิยามความสัมพันธ์เฉพาะเจาะจงของโปรแกรมสัตว์เลี้ยง

Name	Definition
R0	Return type is struct Animal
R1	Return type is struct Meaw
R2	Return type is struct Woof
R3	Return type is struct Moo
R4	Has argument of type struct Animal
R5	Has argument of type struct Meaw
R6	Has argument of type struct Woof
R7	Has argument of type struct Moo
R8	Use field of type struct Animal
R9	Use field of type struct Meaw
R10	Use field of type struct Woof
R11	Use field of type struct Moo

ตารางที่ 4.5 นิยามความสัมพันธ์เฉพาะเจาะจงของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะและแบบผูกติด

Name	Definition
R0	Return type is struct stack
R1	Return type is struct queue
R2	Has argument of type struct stack
R3	Has argument of type struct queue
R4	Use field of struct stack
R5	Use field of struct queue

ตารางที่ 4.6 นิยามความสัมพันธ์เฉพาะเจาะจงของโปรแกรมการลงทะเบียนของพนักงาน

Name	Definition
R0	Return variable max_pos
R1	Has variable max_pos argument
R2	Use variable max_pos
R3	Return variable cur_pos
R4	Has variable cur_pos argument
R5	Use variable cur_pos
R6	Return variable head_ptr
R7	Has variable head_ptr argument
R8	Use variable head_ptr
R9	Return struct list
R10	Has struct list argument

4.3.2.2 ผลการตรวจสอบความสัมพันธ์

ผลที่ได้จากการทำงานในขั้นตอนนี้ คือ ตารางความสัมพันธ์ที่แสดงถึงความสัมพันธ์ระหว่างส่วนคำสั่งกับส่วนข้อมูล ได้แก่ ตารางความสัมพันธ์ของโปรแกรมสัตว์เลี้ยง ตารางความสัมพันธ์ของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะ ตารางความสัมพันธ์ของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด และตารางความสัมพันธ์ของโปรแกรมการลงทะเบียนของพนักงาน ตามตารางที่ 4.7 4.8 4.9 และ 4.10 ตามลำดับ

ตารางที่ 4.7 ตารางความสัมพันธ์ระหว่างส่วนคำสั่งกับส่วนข้อมูลของโปรแกรมสัตว์เลี้ยง

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
getName												
getWeight												
getCatInfo						T			T	T		
getDogInfo							T		T		T	
getCowInfo								T	T			T
canClimbTree						T				T		
setCatInfo						T			T	T		
setDogInfo							T		T		T	
setCowInfo								T	T			T

ตารางที่ 4.8 ตารางความสัมพันธ์ระหว่างส่วนคำสั่งกับส่วนข้อมูลของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะ

	R0	R1	R2	R3	R4	R5
initStack	T				T	
initQ		T				T
isEmptyStack			T		T	
isEmptyQ				T		T
push			T		T	
enQ				T		T
pop			T		T	
deQ				T		T

ตารางที่ 4.9 ตารางความสัมพันธ์ระหว่างส่วนคำสั่งกับส่วนข้อมูลของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด

	R0	R1	R2	R3	R4	R5
initStack	T				T	
initQ		T				T
isEmptyStack		T			T	
isEmptyQ			T	T	T	T
push		T			T	
enQ			T	T	T	T
pop		T			T	
deQ			T		T	

ตารางที่ 4.10 ตารางความสัมพันธ์ระหว่างส่วนคำสั่งกับส่วนข้อมูลของโปรแกรมการลงทะเบียนของพนักงาน

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
add_pos			T												
clear_pos			T												
check_pos			T			T									
enroll		T													T
dismiss		T													
init			T					T							
add			T					T				T			
del							T	T							
search			T	T				T							T
print			T					T							T

4.3.3 ผลการจัดเรียงส่วนประกอบเดิม

เนื่องจากการทำงานในขั้นตอนนี้ แบ่งการทำงานออกเป็นหลายส่วน ดังนั้นจึงได้แบ่งการแสดงผลออกเป็นกลุ่ม ๆ ดังนี้

4.3.3.1 ผลการสร้างเมตริกซ์ข้อมูล

ผลที่ได้จากการทำงานในขั้นตอนนี้ คือ เมตริกซ์ที่ได้จากการแปลงตารางความสัมพันธ์ ได้แก่ เมตริกซ์ข้อมูลของโปรแกรมสัตว์เลี้ยง เมตริกซ์ข้อมูลของโปรแกรมลำดับแถวคอยแบบส่วนจำเพาะ เมตริกซ์ข้อมูลของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด และเมตริกซ์ข้อมูลของโปรแกรมการลงทะเบียนของพนักงาน ตามตารางที่ 4.11 4.12 4.13 และ 4.14 ตามลำดับ

ตารางที่ 4.11 เมตริกซ์ข้อมูลของโปรแกรมสัตว์เลี้ยง

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
getName	0	0	0	0	0	0	0	0	0	0	0	0
getWeight	0	0	0	0	0	0	0	0	0	0	0	0
getCatInfo	0	0	0	0	0	1	0	0	1	1	0	0
getDogInfo	0	0	0	0	0	0	1	0	1	0	1	0
getCowInfo	0	0	0	0	0	0	0	1	1	0	0	1
getAction	0	0	0	0	0	0	0	0	0	0	0	0
canClimbTree	0	0	0	0	0	1	0	0	0	1	0	0
setCatInfo	0	0	0	0	0	1	0	0	1	1	0	0
setDogInfo	0	0	0	0	0	0	1	0	1	0	1	0
setCowInfo	0	0	0	0	0	0	0	1	1	0	0	1

ตารางที่ 4.12 เมตริกซ์ข้อมูลของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะ

	R0	R1	R2	R3	R4	R5
initStack	1	0	0	0	1	0
initQ	0	1	0	0	0	1
isEmptyStack	0	0	1	0	1	0
isEmptyQ	0	0	0	1	0	1
push	0	0	1	0	1	0
enQ	0	0	0	1	0	1
pop	0	0	1	0	1	0
deQ	0	0	0	1	0	1

ตารางที่ 4.13 เมตริกซ์ข้อมูลของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด

	R0	R1	R2	R3	R4	R5
initStack	1	0	0	0	1	0
initQ	0	1	0	0	0	1
isEmptyStack	0	0	1	0	1	0
isEmptyQ	0	0	0	1	1	1
push	0	0	1	0	1	0
enQ	0	0	0	1	1	1
pop	0	0	1	0	1	0
deQ	0	0	0	1	0	1

ตารางที่ 4.14 เมตริกซ์ข้อมูลของโปรแกรมการลงทะเบียนของพนักงาน

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
add_pos	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
clear_pos	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
check_pos	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
enroll	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
dismiss	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
init	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
add	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0
del	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
search	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1
print	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1

4.3.3.2 ผลการจัดเรียงส่วนประกอบเดิม

ผลที่ได้จากการทำงานในขั้นตอนนี้ คือ เมตริกซ์ค่าความคล้ายคลึงของโปรแกรมสัตว์เลี้ยง เมตริกซ์ค่าความคล้ายคลึงของของโปรแกรมลำดับแถวคอยแบบส่วนจำเพาะ เมตริกซ์ค่าความคล้ายคลึงของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด และเมตริกซ์ค่าความคล้ายคลึงของโปรแกรมการลงทะเบียนของพนักงาน ตามตารางที่ 4.15 4.16 4.17 และ 4.18 ตามลำดับ

ตารางที่ 4.15 เมตริกซ์ค่าความคล้ายคลึงของโปรแกรมสัตว์เลี้ยง

	getN	getW	getCI	getDoI	getCoI	getA	canCli	setCI	setDoI	setCoI
getN	0	0	1.732	1.732	1.732	0	1.414	1.732	1.732	1.732
getW		0	1.732	1.732	1.732	0	1.414	1.732	1.732	1.732
getCI			0	2	2	1.732	1	0	2	2
getDoI				0	2	1.732	2.236	2	0	2
getCoI					0	1.732	2.236	2	2	0
getA						0	1.414	1.732	1.732	1.732
caCli							0	1	2.236	2.236
setCI								0	2	2
setDoI									0	2
setCoI										0

ตารางที่ 4.16 เมตริกซ์ค่าความคล้ายคลึงของโปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะ

	initS	initQ	isEmpS	isEmpQ	push	enQ	pop	deQ
initS	0	2	1.414	2	1.414	2	1.414	2
initQ		0	2	1.414	2	1.414	2	1.414
isEmpS			0	2	0	2	0	2
isEmpQ				0	2	0	2	0
push					0	2	0	2
enQ						0	2	0
pop							0	2
deQ								0

ตารางที่ 4.17 เมตริกซ์ค่าความคล้ายคลึงของโปรแกรมแถวลำดับและแถวคอยแบบผูกติด

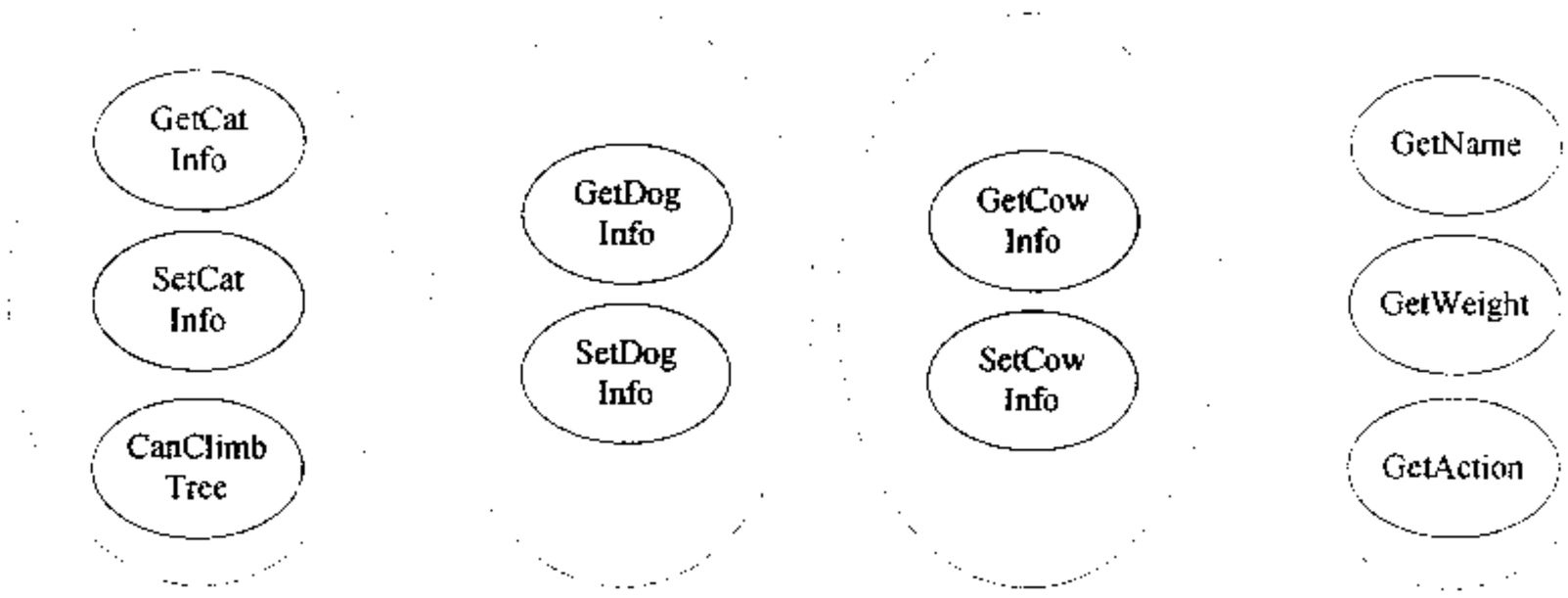
	initS	initQ	isEmpS	isEmpQ	push	enQ	pop	deQ
initS	0	2	1.414	1.732	1.414	1.732	1.414	2
initQ		0	2	1.732	2	1.732	2	1.414
isEmpS			0	1.732	0	1.732	0	2
isEmpQ				0	1.732	0	1.732	1
push					0	1.732	0	2
enQ						0	1.732	1
pop							0	2
deQ								0

ตารางที่ 4.18 เมตริกซ์ค่าความคล้ายคลึงของโปรแกรมการลงทะเบียนของพนักงาน

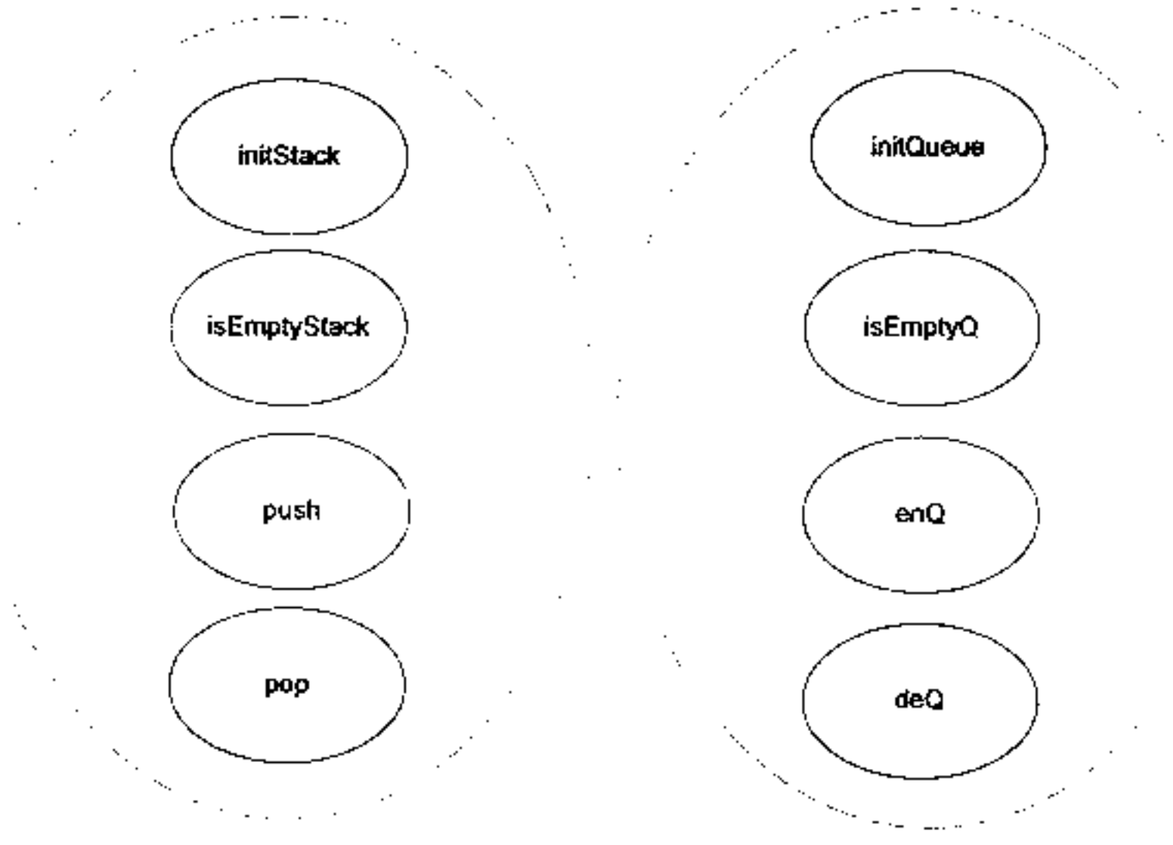
	add_p	clr_p	chk_p	enroll	dismiss	init	add	del	search	print
add_p	0	0	1	1.732	1.414	1.732	2	1.732	2.236	2
clr_p		0	1	1.732	1.414	1.732	2	1.732	2.236	2
chk_p			0	1.414	1	2	2.236	2	2.449	2.236
enroll				0	1	2	2.236	2	2	1.732
dismiss					0	1.732	2	1.732	2.236	2
init						0	1	1.414	1.414	1
add							0	1.732	1.732	1.414
del								0	2	1.732
search									0	1
print										0

4.3.3.3 ผลการจัดกลุ่มโดยใช้ดัชนีชี้วัด Dunn Like Indices

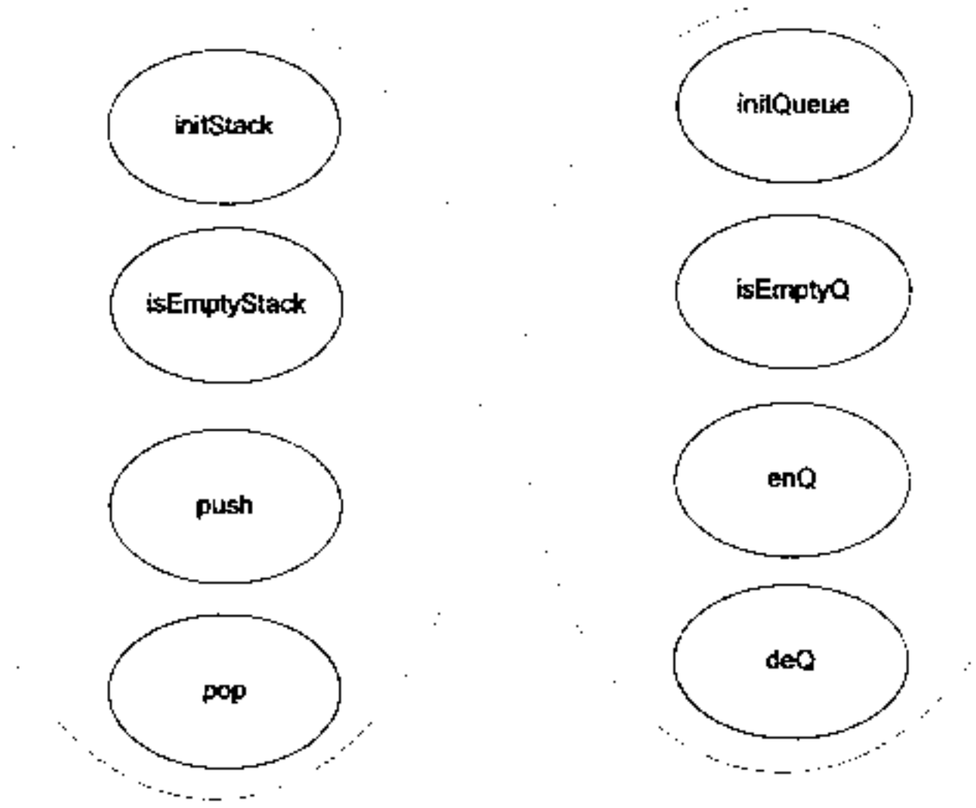
ผลที่ได้จากการทำงานในขั้นตอนนี้ คือ ลักษณะของการจัดกลุ่มของส่วนคำสั่งต่างๆ ของโปรแกรมสัตว์เลี้ยง โปรแกรมแถวลำดับและแถวคอยแบบส่วนจำเพาะและแบบผูกติด โปรแกรมการลงทะเบียนของพนักงาน ดังแสดงในรูปที่ 4.8 4.9 4.10 และ 4.11 ตามลำดับ สำหรับรายละเอียดของการจัดกลุ่มของส่วนคำสั่ง จะแสดงไว้ที่ภาคผนวก



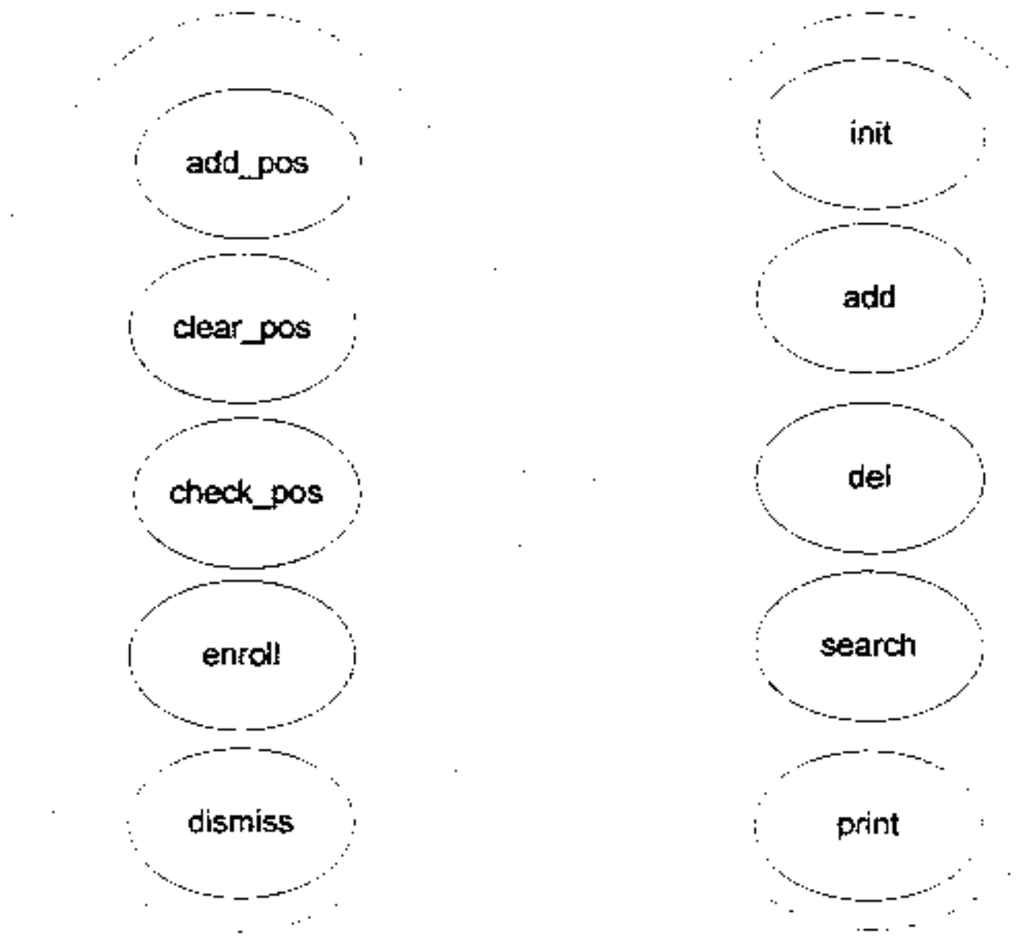
ภาพที่ 4.7 ผลการจัดกลุ่มข้อมูลของโปรแกรมสัตว์เลี้ยง



ภาพที่ 4.8 ผลการจัดกลุ่มข้อมูลของโปรแกรมแถวลำดับและแถวคอย แบบส่วนจำเพาะ



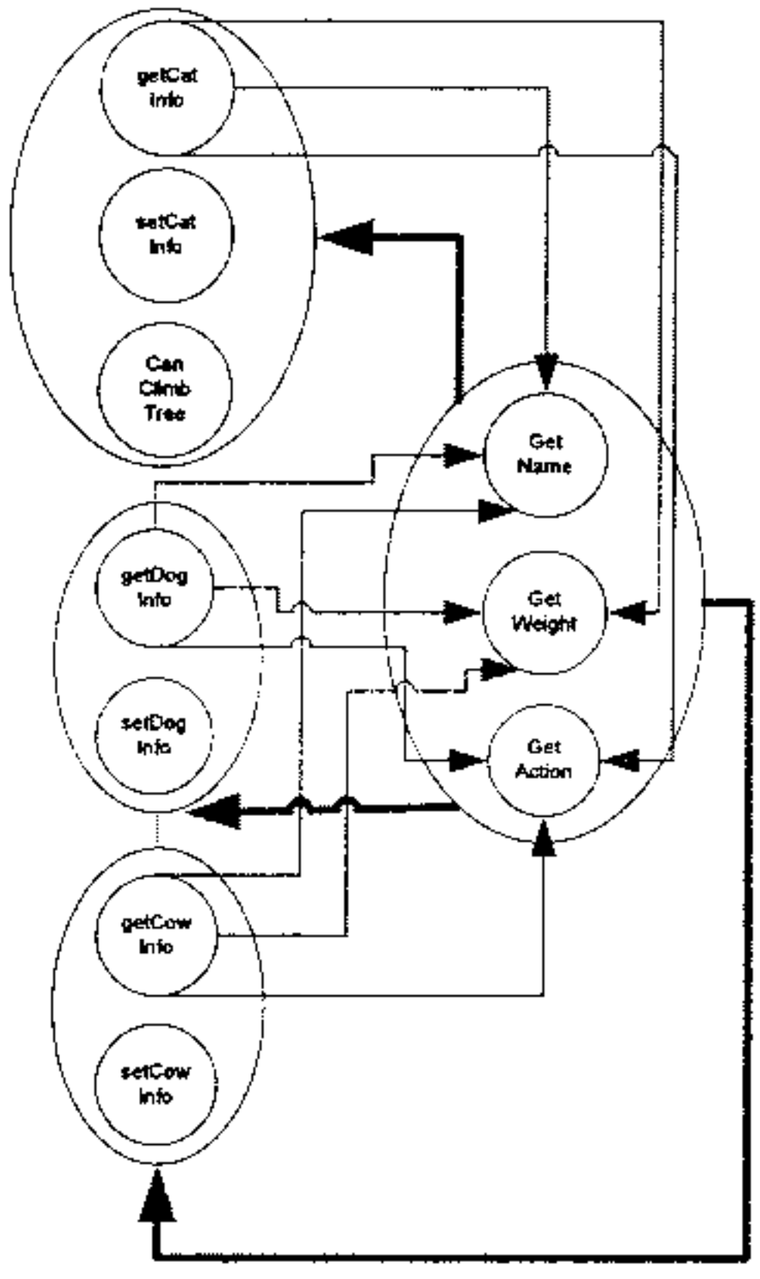
ภาพที่ 4.9 ผลการจัดกลุ่มข้อมูลของโปรแกรมแถวลำดับและแถวคอย แบบผูกติด



ภาพที่ 4.10 ผลการจัดกลุ่มข้อมูลของโปรแกรมการลงทะเบียนของพนักงาน

4.3.3.4 ผลจากการหาความสัมพันธ์แบบสืบทอดโดยอัตโนมัติ

ผลที่ได้จากการใช้เทคนิคของกราฟเพื่อหาความสัมพันธ์แบบสืบทอดโดยอัตโนมัติกับโปรแกรมสัตว์เลี้ยง พบว่า โปรแกรมสัตว์เลี้ยงมีความสัมพันธ์แบบสืบทอดโดยอัตโนมัติ ดังแสดงในภาพที่ 4.12 โดยลูกศรเส้นบาง จะแสดงลักษณะของการถูกเรียกใช้งานวัตถุต่าง ๆ และลูกศรเส้นเข้ม จะแสดงความสัมพันธ์แบบพ่อกับลูก



ภาพที่ 4.11 ผลการหาความสัมพันธ์แบบพ่อกับลูกของโปรแกรมสัตว์เลี้ยง