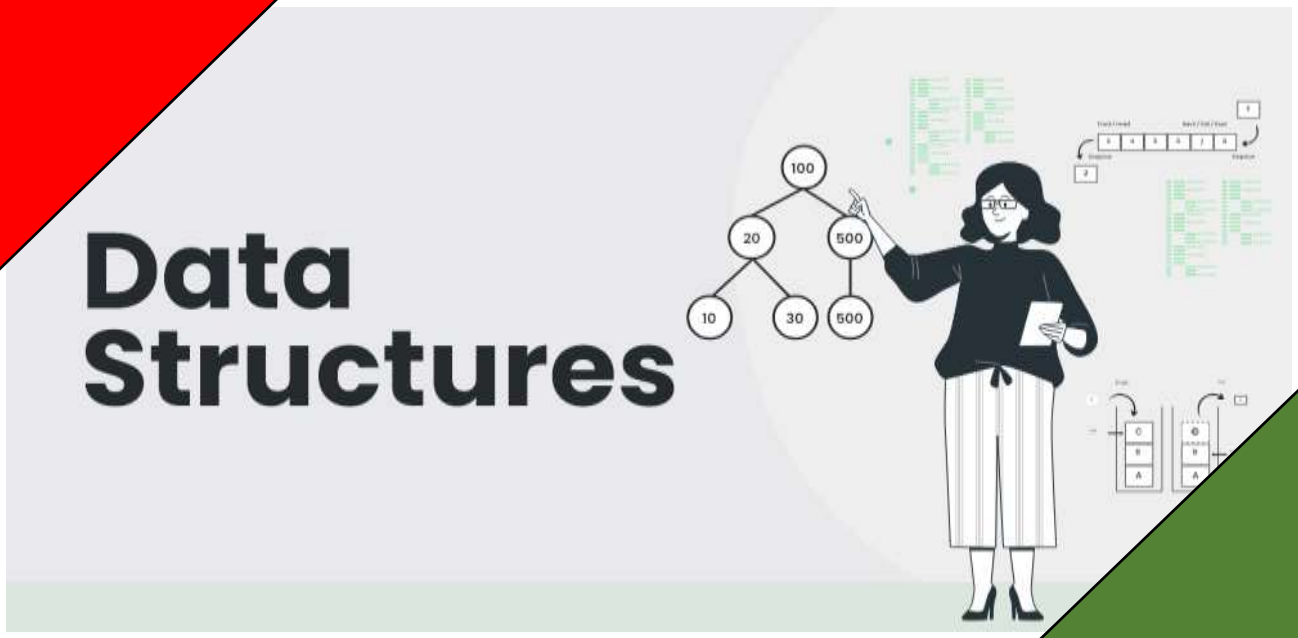
Royal University of Phnom Penh

Department of Science

Subject: Data Structure and Algorithms

Topic: Top Down, Stepwise and Linked List

Lecturer: Ung Rithy



Group 5

- CHHIM Seangleng
- CHAN Datheanarith
- CHHANG Mengchhay
- Chan Socheat
- AING Mengky
- CHAN Sarun
- CHEA Sokchan

អារម្ភកថា

សូមស្វាគមន៍មិត្តៗនិស្សិត ប្រិយមិត្តអ្នកអ្នកស្រាវជ្រាវ និងស្វែងយល់នូវឯកសារ ផ្សេងៗជាទីគោរព យើងខ្ញុំជានិស្សិតរៀននៅសាកលវិទ្យាល័យភូមិន្ទភ្នំពេញ ជំនាន់ទី ២៥ឆ្នាំទី ២ ឆមាស់ទី ២ ជំនាញ Computer Science យើងខ្ញុំមានសេក្តីសប្បាយរីករាយណាស់ ចំពោះការដាក់កិច្ចការស្រាវជ្រាវទៅលើមុខវិជ្ជា Data structure and Algorithms របស់លោកគ្រូសាស្ត្រាចារ្យ **អ៊ុង រឹទ្ធី** ដែលបានផ្តល់កិច្ចការស្រាវជ្រាវនេះដល់យើងខ្ញុំសម្រាប់ធ្វើការសិក្សាស្រាវជ្រាវ ដើម្បីពង្រីកចំណេះដឹង បទពិសោធន៍ និង ទុកជា ឯកសារទៅដល់សិស្សប្អូនជំនាន់ក្រោយៗទៀត។ ហេតុនេះទើបយើងខ្ញុំធ្វើការសិក្សាស្រាវជ្រាវ និងចងក្រងធ្វើជាសៀវភៅនេះឡើង។

សៀភៅនេះ វាគ្រាន់តែបកស្រាយនូវបញ្ហាមួយចំនួនតូច ដែលខ្ញុំធ្លាប់បានសិក្សាស្រាវជ្រាវ កន្លងមក និងតាមឯកសារយោងខ្លះៗនៅក្នុងបណ្ណាល័យអ៊ីនធឺណែត និងកន្លែងផ្សេងៗ។ យើងខ្ញុំសង្ឃឹមថាសៀវភៅនេះជាមូលដ្ឋានមួយសម្រាប់ឈានទៅរកភាពលំអិតផងដែរ។ ហើយឯកសារនេះ គ្រាន់តែបង្ហាញពីវិធីសាស្ត្រមួយ ចំនួនតែប៉ុណ្ណោះ។ជាចុងក្រោយយើងខ្ញុំរង់ចាំទទួលការរិះគន់ពីសំណាក់មិត្តអ្នកអាន អ្នកស្រាវជ្រាវ លោកគ្រូ សាស្ត្រាចារ្យ មិត្តរួមជំនាន់ និងប្អូនៗជំនាន់ក្រោយ រាល់កំហុសឆ្គងដែរកើតឡើងដោយ អច្ចេតនា នូវពាក្យពេចន៍ វត្តឃ្លា វេយ្យាករណ៍ អក្ខរក្រម ដើម្បីជួយកែលំអរអោយសៀវភៅនេះកាន់តែល្អ និងមានភាពសុក្រិត។

ជាទីបញ្ចប់ យើងខ្ញុំសូមជូនពរដល់ មិត្តអ្នកនោះ អ្នកស្រាវជ្រាវ លោកគ្រូ សាស្ត្រាចារ្យ និងមិត្តនិស្សិតទាំងអស់ ទទួលបាននូវពរទាំងឡាយបួន សម្បត្តិបី អរិយទ្រព្យប្រាំពីរ និងមានសេក្តីសុខសំណាងល្អគ្រប់ពេលវេលា ជោគជ័យលើការសិក្សា និងជោគជ័យនូវគោលបំណងដែលប៉ងគ្រប់ៗគ្នាកុំបីខានឡើយ។

រាជធានីភ្នំពេញ ថ្ងៃទី ១៨ ខែ សីហា ឆ្នាំ ២០២៣

មាតិកា

១. អារម្ភកថា ០២

២. មាតិកា ០៣

៣. សេចក្តីផ្តើម ០៤

៤. គោលបំណង់ ០៤

៥. Top Down Design ០៥

៦. Stepwise Refinement ០៧

៧. Source code ១៣

សេចក្តីផ្តើម

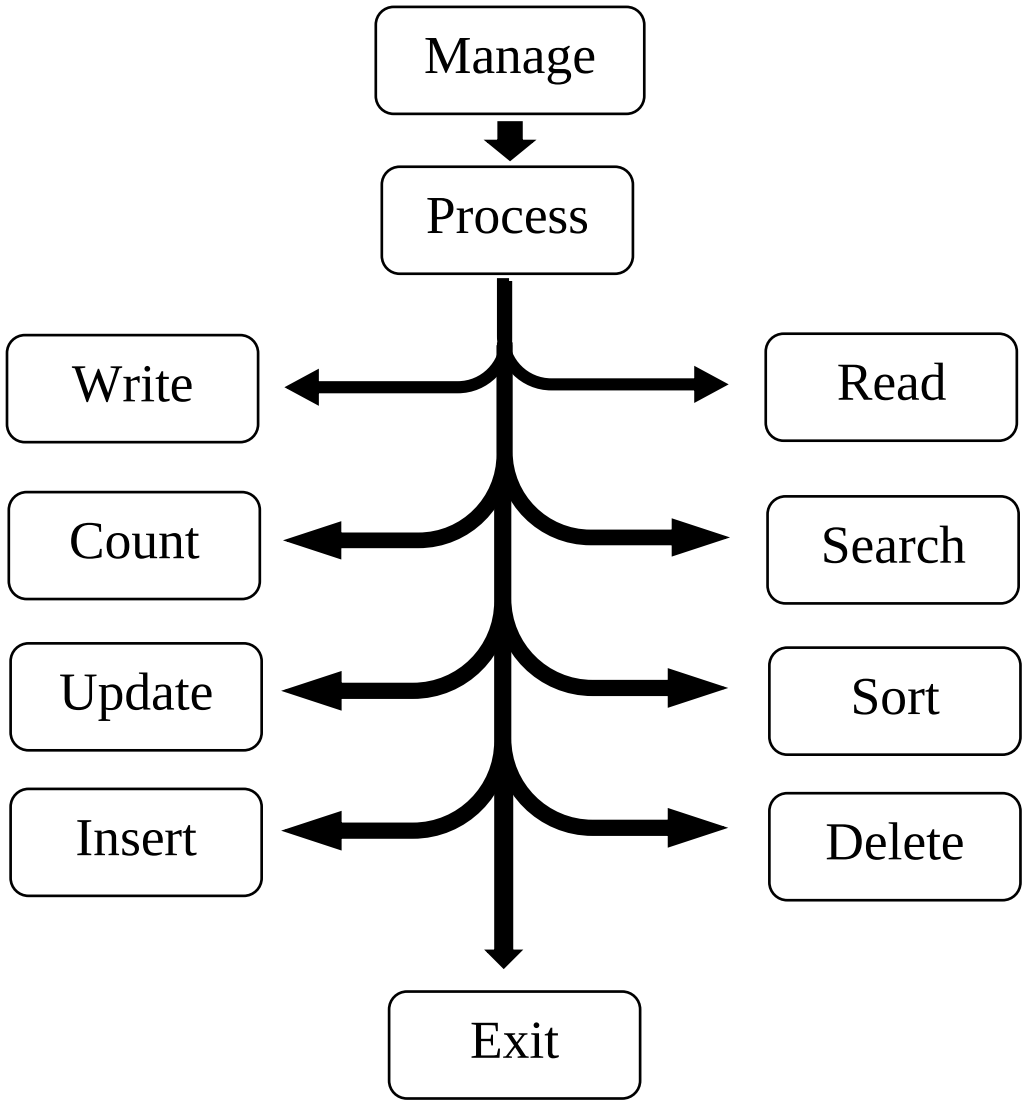
ការសិក្សានេះគឺកិច្ចការស្រាវជ្រាវនៃមុខវិជ្ជា Data Structure and Algorithms នៃសាស្ត្រាចារ្យ **អ៊ុង រិទ្ធី** នៃសកលវិទ្យាល័យភូមិន្ទភ្នំពេញ ដែលជាកិច្ចការស្រាវជ្រាវ សម្រាប់ឆ្នាំសិក្សា ឆ្នាំទី២ សមាសទី២ ដែលសិក្សាទៅលើការរចនាសម្ព័ន្ធនៃក្រុមហ៊ុនសម្រាប់ការគ្រប់គ្រង់ទិន្នន័យរបស់បុគ្គលិកទាំងអស់ដែលបានចូលបម្រើការងារនៅក្នុងស្ថាប័ន ដែលដំណើរការនៃការរក្សាទុកផ្តោតទៅលើ ការបង្កើតទិន្នន័យ ការរក្សារទុកទិន្នន័យ ការកែប្រែទិន្នន័យ ការរាប់ចំនួនទិន្នន័យ ការតម្រៀបទិន្នន័យ ការស្វែងរកទិន្នន័យ ការបន្ថែមទិន្នន័យ និង ការលុបទិន្នន័យរបស់ បុគ្គលិកស្ថាប័ន. ដែលមានប្រមាណវិធីដូចជា Create list , read list , count , Search , update , Sort , Insert , Delete.

គោលបំណង់

គោលបំណងនៃការសិក្សានេះគឺទាក់ទងទៅនឹងរចនាសម្ព័ននៃស្ថាប័នការងាមួយដែលអាចបង្កើតទិន្នន័យរបស់បុគ្គលិក រក្សារទុកទិន្នន័យរបស់បុគ្គលិក កែប្រែទិន្នន័យរបស់បុគ្គលិក រាប់ចំនួនបុគ្គលិកទាំងអស់ តម្រៀបទិន្នន័យតាមប្រាក់ខែ ស្វែងរកទិន្នន័យរបស់បុគ្គលិក ការបន្ថែមទិន្នន័យរបស់បុគ្គលិក និង ការលុបទិន្នន័យរបស់របស់បុគ្គលិក ដែលមានប្រមាណវិធី ដូចជា Create list , read list , count , Search , update , Sort , Insert , Delete ដែលផ្តល់ភាពងាយស្រួលដល់អ្នកប្រើប្រាស់សម្រាប់ ការគ្រប់គ្រងទិន្នន័យរបស់បុគ្គលិកនីមួយៗនៅក្នុងស្ថាប័ន។

Top Down Design

- ដំបូងចូលទៅកាន់ រួចចាប់ផ្តើមដំណើរការ
- Write សម្រាប់បង្កើនទិន្នន័យនៃចំនួនបុគ្គលិកដែលមាន
- Read សម្រាប់បង្ហាញចេញនូវទិន្នន័យដែលបានបង្កើត
- Count សម្រាប់បង្ហាញចេញនូវចំនួនបុគ្គលិកទាំងអស់ដែលបានបង្កើត
- Search សម្រាប់ស្វែងរកទិន្នន័យជាក់លាក់របស់បុគ្គលិកណាម្នាក់
- Update សម្រាប់កែប្រែទិន្នន័យរបស់បុគ្គលិកណាម្នាក់
- Sort សម្រាប់តម្រៀបរបស់បុគ្គលិកទៅតាមប្រាក់ខែរបស់ពួកគេ
- Insert សម្រាប់បង្កើតទិន្នន័យបន្ថែមទៅលើទិន្នន័យដែលមាន
  - បន្ថែមទិន្នន័យនៅទីតាំងដើម
  - បន្ថែមទិន្នន័យនៅទីតាំងចុងក្រោយ
  - បន្ថែមទិន្នន័យនៅទីតាំងជាក់លាក់ណាមួយ
- Delete សម្រាប់លុបទិន្នន័យជាក់លាក់ណាមួយចេញ
  - លុបទិន្នន័យនៅទីតាំងដើម
  - លុបទិន្នន័យនៅទីតាំងចុងក្រោយ
  - លុបទិន្នន័យនៅទីតាំងជាក់លាក់ណាមួយ


Manage
Process
Write
Read
Count
Search
Update
Sort
Insert
Delete
Exit

Stepwise Refinement

1. Write

- ក្នុងការសិក្សានេះយើងមានទិន្នន័យដូចខាងក្រោមដែលយើងត្រូវធ្វើការសិក្សា
- ទិន្នន័យមានដូចជា id;, name[MAX_SIZE], address[MAX_SIZE],

  gender[MAX_SIZE], phoneNumber[MAX_SIZE],

  birthOfDate[MAX_SIZE], salary
- ភាសាកម្មវិធី

```
struct Employee{
    int id;
    char name[MAX_SIZE];
    char address[MAX_SIZE];
    char gender[MAX_SIZE];
    char phoneNumber[MAX_SIZE];
    char birthOfDate[MAX_SIZE];
    float salary;
};
```

- បង្កើត memory location អោយទៅ newnode ដើម្បីផ្ទុកព័ត៌មានរបស់បុគ្គលិក
- បន្តាប់មកធ្វើការបញ្ចូលព័ត៌មានរបស់បុគ្គលិក
- អោយ newnode ផ្ទុកទិន្នន័យដែលបានបញ្ចូល
- ភ្ជាប់ newnode ទៅក្នុង Linked List

```
struct NodeType *createList()
{
    struct NodeType *newnode;
    newnode = getNode();
    // gets(name);
    newnode->emp=setInfo();
    newnode->next = NULL;
    return newnode;
}
struct Employee setInfo(){
    struct Employee stu;
    printf("\tInput Employee ID: ");
    scanf("%d",&stu.id);
    printf("\tInput Employee Name: ");
```

```
    fflush(stdin);
    gets(stu.name);
    printf("\tInput Gender: ");
    gets(stu.gender);
    printf("\tInput BOD: ");
    gets(stu.birthOfDate);
    printf("\tInput Phone number: ");
    gets(stu.phoneNumber);
    printf("\tInput Address: ");
    gets(stu.address);
    printf("\tInput Salary: ");
    scanf("%f",&stu.salary);

    return stu;
}
```

2. Read

- ដំបូងយើងត្រូវបង្កើត pointer getString
- ទាញយកឈ្មោះចេញមក្រៅដោយប្រើ gets(name);
- ផ្តល់ទៅវិញនូវតម្លៃ name ដោយ return name
- បង្កើត Function getInfo សម្រាប់បញ្ចេញទិន្នន័យ

```
char *getString()
{
    char *name;
    name = (char *)malloc(sizeof(name));
    gets(name);
    return name;
}


void getInfo(struct NodeType* temp){
    printf("\t%d\t%s\t%s\t%s\t%s\t%s\t%.2f\n",temp->stu.id,temp-
>stu.name,temp->stu.gender,temp->stu.birthOfDate,
    temp->stu.phoneNumber,temp->stu.address,temp->stu.salary);
}
```

3. Count

- ដំបូងយើងត្រូវបង្កើត temp ដើម្បីជា អាញាត់ ដើម្បីចាប់ផ្តើម
- អោយតម្លៃ temp= *plist* (តម្លៃដំបូង)
- ហើយ ប្រើ funcation while (temp != NULL) ដើម្បីអោយ temp រត់ពីដើមដល់ចប់ នៃ Linked list
- នៅក្នុង while បើលក្ខខណ្ឌ ពិត អោយតម្លៃ count កើនមួយតម្លៃ
- ហើយ អោយ temp = temp->next (temp ស្មើតម្លៃបន្ទាប់)

```
int countNode(struct NodeType *plist)
 {
    struct NodeType *temp;
    temp = plist;
    int count = 0;
    while (temp != NULL)
    {
       count++;
       temp = temp->next;
    }
    return count;
 }
```

```
int count;
        // Count numbers of node in linked list
        count = countNode(plist);
        printf("\n\tNumbers of employee in the list : %d employee\n", count);
```

4. Search

- យើងត្រូវបង្កើត SearchName; ដើម្បីអោយគេ បញ្ចូលដើម្បីស្វែងរកឈ្មោះដែលចង់រក
- ប្រើ SearchName = getString() ដើម្បីអោយ គេអាចបញ្ចូលឈ្មោះនោះបាន
- ហៅ function searchNode ដែលបានបង្កើតរួច យកមកប្រើដើម្បីsearch
- ពេលរកឃើញ វានឹងបង្ហាញទិន្នន័យនោះ
- បើរកមិនឃើញវានឹងផ្ញើរសារមកវិញថា Search not found

```
char *searchName;
        int found = 0;
        // Input name
        printf("\tPlease input name to search: ");
        fflush(stdin);
       searchName = getString();
        // Search node by name
        temp = searchNode(plist, searchName, &found);
        if (found == 1)
        {
           printf("\nSearch is found!!!\n");
           printf("ID\tName\tGender\tDoB\tTEL\tAddress\tSalary\n");
           getInfo(temp);
         }
        else
           printf("\nSearch not found!!!\n");
```

5. Update

- បង្កើត Function updateNode ដើម្បី update ទិន្នន័យ
- បង្កើត Object របស់ struct Employee ដើម្បីទទួលទិន្នន័យដែលត្រូវ update
- អោយ emp=setInfo() ដើម្បី update ទិន្នន័យ
- អោយ temp->emp=emp ដែលមានន័យថា ទិន្នន័យថ្មីស្មើទិន្នន័យចាស់

```
void updateNode(struct NodeType *temp)
 {
    struct Employee emp;
    emp=setInfo();
    temp->emp=emp;
 }
```

6. Sort

- បង្កើត Function sortNode ដើម្បី sort ទិន្នន័យ
- បង្កើត struct NodeType ដូចជា*p, *temp, *ptr ដើម្បីប្រតិបត្តិការ
- អោយ p = plist; (ទិន្នន័យដំបូង)

- បង្កើត លក្ខខណ្ឌ while (p != NULL)
- កំណត់លក្ខខណ្ឌ if (strcmp(p->emp.name, ptr-> emp.name) > 0) ដែលមានន័យថាបើសិន ទិន្នន័យ p->emp.name ប្រៀបធៀបនឹង ptr-> emp.name ធំជាង 0 នោះអោយទិន្នន័យទាំងពីរផ្លាស់ប្តូរទិន្នន័យគ្នា
- អោយ ptr = ptr->next; ទិន្នន័យបន្ទាប់
- អោយ p = p->next; ទិន្នន័យបន្ទាប់

```
void sortNode(struct NodeType *plist)
 {
    struct NodeType *p;
    struct NodeType *temp;
    struct NodeType *ptr;
    temp = getNode();
    p = plist;
    while (p != NULL)
    {
       ptr = p->next;
       while (ptr != NULL)
       {
          if (strcmp(p->emp.name, ptr-> emp.name) > 0)
          {
             strcpy(temp-> emp.name, p-> emp.name);
             strcpy(p-> emp.name, ptr-> emp.name);
             strcpy(ptr-> emp.name, temp-> emp.name);
          }
          ptr = ptr->next;
       }
       p = p->next;
    }
 }
```

7. Insert

-
- បង្កើត *newnode ដើម្បីផ្ទុកទិន្នន័យរបស់បុគ្គលិកថ្មី
- ធ្វើការបញ្ចូលពត័មានរបស់បុគ្គលិកថ្មី រួចហើយអោយ next របស់ newnode ចង្អុលទៅកាន់ address ទី 0

- ប្រសិនបើ i តូចជាង position អោយតម្លៃ i កើនមួយ ហើយអោយ temp = temp->next។ ធ្វើបែបនេះរហូតដល់ i ធំជាង position
- បន្ទាប់មក អោយ next នៃ newnode ស្មើនឹង next នៃ nodetemp ហើយ next នៃ nodetemp ស្មើនឹង newnode ចុងក្រោយ return plist

```
struct NodeType *insertAtPosition(struct NodeType *plist, char *name, int position)
 {
    struct NodeType *newnode;
    struct NodeType *temp;
    int i = 1;
    temp = plist;
    newnode = getNode();
    newnode->stu=setInfo();
    newnode->next = NULL;
    while (i < position)
    {
       temp = temp->next;
       i++;
    }
    newnode->next = temp->next;
    temp->next = newnode;
    return plist;
 }
```

8. Delete

- អោយ nodetemp = nodeplist
- បើ i តូចជាង position - 1អោយតម្លៃ i កើនមួយ មួយ ហើយអោយ temp = temp->next។ ធ្វើបែបនេះរហូតដល់ i ធំជាង position
- អោយ ptr = temp->next ហើយ temp->next = ptr->next;
- លុប nodeptr រួចធ្វើការ ហើយចុងក្រោយ return plist;

```
struct NodeType *deleteAtPosition(struct NodeType *plist, int position)
 {
    struct NodeType *ptr;
    struct NodeType *temp;
    int i = 1;
    temp = plist;
    while (i < position - 1)
    {
       temp = temp->next;
       i++;
    }
    ptr = temp->next;
    temp->next = ptr->next;
    free(ptr);
    return plist;
 }
```

Source Code

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
#define MAX_SIZE 20
struct Employee{
   int id;
   char name[MAX_SIZE];
   char address[MAX_SIZE];
   char gender[MAX_SIZE];
   char phoneNumber[MAX_SIZE];
   char birthOfDate[MAX_SIZE];
   float salary;
};
struct NodeType
{
   struct Employee emp;
   struct NodeType *next;
};
void initialization(struct NodeType *);
```

```
struct Employee setInfo();
void getInfo(struct NodeType*);
void freeNode(struct NodeType *);
struct NodeType *getNode();
void traverseNode(struct NodeType *);
struct NodeType *createList();
char *getString();
int countNode(struct NodeType *);
int getOption();
char getChar();
void run();
void menu();
void insertMenu();
void deleteMenu();
struct NodeType *searchNode(struct NodeType *, char *, int *);
void updateNode(struct NodeType *);
struct NodeType *insertNode(struct NodeType *);
struct NodeType *insertAtBegin(struct NodeType *, char *);
struct NodeType *insertAtEnd(struct NodeType *, char *);
struct NodeType *insertAtPosition(struct NodeType *, char *, int);
void sortNode(struct NodeType *);
struct NodeType *deleteNode(struct NodeType *);
struct NodeType *deleteAtBegin(struct NodeType *);
struct NodeType *deleteAtEnd(struct NodeType *);
struct NodeType *deleteAtPosition(struct NodeType *, int);
int main()
{
   run();
   return 0;
}
void run()
{
   int option;
   char ch;
   struct NodeType *plist, *temp, *newnode;
//    initialization(plist);
    plist=NULL;
   do
   {
      system("cls");
      menu();
      option = getOption();
      switch (option)
      {
      case 1:
      {
         do
         {
            // Input string
```

```
        printf("\tPlease input data: \n");
        // Create node
        newnode = createList();
        if (plist == NULL)
        {
           plist = newnode;
           temp = newnode;
        }
        else
        {
           temp->next = newnode;
           temp = newnode;
        }
        printf("\tDo you want to add again(y/n)?: ");
        fflush(stdin);
        ch = getChar();
    } while (ch == 'y' || ch == 'Y');
    printf("\n");
}
break;
case 2:
{
    temp = plist;
   //Traverse linked list
    printf("\n\tList of Employee: \n");
    printf("\tID\tNAME\tGENDER\tDOB\tTEL\tADDRESS\tSalary\n");
    traverseNode(temp);
}
break;
case 3:
{
    int count;
    // Count numbers of node in linked list
    count = countNode(plist);
    printf("\n\tNumbers of employee in the list : %d employee\n", count);
}
break;
case 4:
{
    char *searchName;
    int found = 0;
    // Input name
    printf("\tPlease input name to search: ");
    fflush(stdin);

    searchName = getString();
    // Search node by name
    temp = searchNode(plist, searchName, &found);
    if (found == 1)
```

```
    {
        printf("\n\tSearch is found!!!\n");
        printf("\tID\tName\tGender\tDOB\tTEL\tAddress\tSalary\n");
        getInfo(temp);
    }
    else
        printf("\n\tSearch not found!!!\n");
}
break;
case 5:
{
    char *searchName;
    int found = 0;
    // Display all node in linkes
    temp = plist;
    printf("\n\tEmployee LIST: \n");
    printf("\tID\tName\tGender\tDOB\tTEL\tAddress\tSalary\n");
    traverseNode(temp);
    // Input name
    printf("\tPlease input name to update: ");
    fflush(stdin);
    // Update nod by name
    searchName = getString();
    temp = searchNode(plist, searchName, &found);
    if (found == 1)
    {
        printf("\n\tSearch is found!!!\n");
        printf("\tPlease input new data: ");
        updateNode(temp);
        printf("\tUpdate successfully!!!\n");
    }
    else
        printf("\n\tSearch not found!!!\n");
}
break;
case 6:
{
    sortNode(plist);
    printf("\tSorted Successfully!!!\n");
}
break;
case 7:
{
    plist = insertNode(plist);
    printf("\tInserted successfully!!!\n");
}
break;
case 8:
{
```

```
            plist = deleteNode(plist);
            printf("\tDeleted successfully!!!\n");
        }
        break;
        case 0:
        {
            printf("\tAre you sure to exit this program(y/n)?: ");
            ch = getChar();
            if ((ch == 'y') || (ch == 'Y'))
            {
                exit(0);
            }
            else
                continue;
        }
        default:
        {
            printf("\tInvalid!!!\n");
        }
        break;
        }
        printf("\n\tPlease any key to countinue...");
        getch();
        printf("\n");
    } while (option != 0);
}
void initialization(struct NodeType *plist)
{
    plist = NULL;
}
struct NodeType *getNode()
{
    struct NodeType *p;
    p = (struct NodeType *)malloc(sizeof(struct NodeType));
    return p;
}
void traverseNode(struct NodeType *temp)
{
    while (temp != NULL)
    {
       getInfo(temp);
        temp = temp->next;
    }
}
void getInfo(struct NodeType* temp){
    printf("\t%d\t%s\t%s\t%s\t%s\t%.2f\n",temp->emp.id,temp->emp.name,temp-
>emp.gender,temp->emp.birthOfDate,
    temp->emp.phoneNumber,temp->stu.address,temp->stu.salary);
}
```

```
void freeNode(struct NodeType *temp)
{
   free(temp);
}
struct NodeType *createList()
{
   struct NodeType *newnode;
   newnode = getNode();
   // gets(name);
   newnode->emp=setInfo();
   newnode->next = NULL;
   return newnode;
}
struct Employee setInfo(){
   struct Employee emp;
   printf("\tInput Employee ID: ");
   scanf("%d",&emp.id);
   printf("\tInput Employee Name: ");
   fflush(stdin);
   gets(emp.name);
   printf("\tInput Gender: ");
   gets(emp.gender);
   printf("\tInput BOD: ");
   gets(emp.birthOfDate);
   printf("\tInput Phone number: ");
   gets(emp.phoneNumber);
   printf("\tInput Address: ");
   gets(emp.address);
   printf("\tInput Salary: ");
   scanf("%f",&emp.salary);

   return emp;
}
char *getString()
{
   char *name;
   name = (char *)malloc(sizeof(name));
   gets(name);
   return name;
}
void menu()
{
   printf("\tGroup 5\n");
   printf("1.Create list\n");
   printf("2.Read list\n");
   printf("3.Count \n");
   printf("4.Search\n");
   printf("5.Update \n");
   printf("6.Sort\n");
```

```
    printf("7.Insert\n");
    printf("8.Delete\n");
    printf("0.Exit\n");
    printf("Please choose one option(1-8): ");
}
int getOption()
{
    int option;
    scanf("%d", &option);
    return option;
}
char getChar()
{
    char ch;
    scanf("%c", &ch);
    return ch;
}
int countNode(struct NodeType *plist)
{
    struct NodeType *temp;
    temp = plist;
    int count = 0;
    while (temp != NULL)
    {
        count++;
        temp = temp->next;
    }
    return count;
}
struct NodeType *searchNode(struct NodeType *plist, char *name, int *found)
{
    struct NodeType *temp;
    temp = plist;
    while (temp != NULL)
    {
        if (strcmp(temp->stu.name, name) == 0)
        {
            *found = 1;
            return temp;
        }
        temp = temp->next;
    }
    *found = 0;
    return NULL;
}
void updateNode(struct NodeType *temp)
{
    struct Employee emp;
    emp=setInfo();
```

```
    temp->emp=emp;
}
void insertMenu()
{
    printf("\t1.Insert Begin\n");
    printf("\t2.Insert End\n");
    printf("\t3.Insert Position\n");
    printf("\t0.Exit\n");
    printf("\tPlease choose one option(1-3): ");
}
void deleteMenu()
{
    printf("\t1.Delete Begin\n");
    printf("\t2.Delete End\n");
    printf("\t3.Delete Position\n");
    printf("\t0.Exit\n");
    printf("\tPlease choose one option(1-3): ");
}
struct NodeType *insertNode(struct NodeType *plist)
{
    int option;
    char *name;
    int position;
    int count;
    do
    {
        system("cls");
        insertMenu();
        option = getOption();
        switch (option)
        {
        case 1:
        {
            printf("\tInsert at begin: ");
            printf("\n\tPlease input name to add: ");
            fflush(stdin);
            name = getString();
            plist = insertAtBegin(plist, name);
            return plist;
        }
        break;
        case 2:
        {
            printf("\tInsert at end: ");
            printf("\n\tPlease input name to add: ");
            fflush(stdin);
            name = getString();
            plist = insertAtEnd(plist, name);
            return plist;
```

```
        }
        break;
        case 3:
        {
            printf("\tInsert at postion:\n");
            printf("\tPlease input postion: ");
            scanf("%d", &position);
            count = countNode(plist);
            if (position <= 0 || position > count)
            {
                printf("\tInvalid position\n");
            }
            else
            {
                printf("\n\tPlease input name to add: ");
                fflush(stdin);
                name = getString();
                plist = insertAtPosition(plist, name, position);
                return plist;
            }
        }
        break;
        case 0:
        {
            option = 0;
        }
        break;
        default:
        {
            printf("\tInvalid!!!\n");
        }
        break;
        }
    } while (option != 0);
}
struct NodeType *insertAtBegin(struct NodeType *plist, char *name)
{
    struct NodeType *newnode;
    struct NodeType *temp;
    temp = plist;
    newnode = getNode();
    newnode->stu=setInfo();
    if (temp == NULL)
    {
        temp = newnode;
    }
    else
    {
        newnode->next = temp;
```

```
      temp = newnode;
   }
   return temp;
}
struct NodeType *insertAtEnd(struct NodeType *plist, char *name)
{
   struct NodeType *newnode;
   struct NodeType *temp;
   temp = plist;
   newnode = getNode();
   newnode->emp=setInfo();
   newnode->next = NULL;
   while (temp->next != NULL)
   {
      temp = temp->next;
   }
   temp->next = newnode;
   return plist;
}
struct NodeType *insertAtPosition(struct NodeType *plist, char *name, int position)
{
   struct NodeType *newnode;
   struct NodeType *temp;
   int i = 1;
   temp = plist;
   newnode = getNode();
   newnode->emp=setInfo();
   newnode->next = NULL;
   while (i < position)
   {
      temp = temp->next;
      i++;
   }
   newnode->next = temp->next;
   temp->next = newnode;
   return plist;
}
struct NodeType *deleteNode(struct NodeType *plist)
{
   struct NodeType *temp;
   temp = plist;
   int option;
   char *name;
   int position;
   int count;
   do
   {
      system("cls");
      insertMenu();
```

```
option = getOption();
switch (option)
{
case 1:
{
   printf("\tBefore delete:\n");
   traverseNode(temp);
   printf("\n\tDelete at begin:\n");
   plist = deleteAtBegin(plist);
   return plist;
}
break;
case 2:
{
   printf("\tBefore delete:\n");
   traverseNode(temp);
   printf("\n\tDelete at end: \n");
   plist = deleteAtEnd(plist);
   return plist;
}
break;
case 3:
{
   printf("\tBefore delete:\n");
   traverseNode(temp);
   printf("\tDelete at postion:\n");
   printf("\tPlease input postion: ");
   scanf("%d", &position);
   count = countNode(plist);
   if (position <= 0 || position > count)
   {
      printf("\tInvalid position\n");
   }
   else
   {
      plist = deleteAtPosition(plist, position);
      return plist;
   }
}
break;
case 0:
{
   option = 0;
}
break;
default:
{
   printf("\tInvalid!!!\n");
}
```

```
        break;
        }
    } while (option != 0);
}
void sortNode(struct NodeType *plist)
{
    struct NodeType *p;
    struct NodeType *temp;
    struct NodeType *ptr;
    temp = getNode();
    p = plist;
    while (p != NULL)
    {
        ptr = p->next;
        while (ptr != NULL)
        {
            if (strcmp(p->emp.name, ptr->emp.name) > 0)
            {
                strcpy(temp->emp.name, p->emp.name);
                strcpy(p->emp.name, ptr->emp.name);
                strcpy(ptr->emp.name, temp->emp.name);
            }
            ptr = ptr->next;
        }
        p = p->next;
    }
}
struct NodeType *deleteAtBegin(struct NodeType *plist)
{
    struct NodeType *temp;
    struct NodeType *ptr;
    temp = plist;
    if (temp->next == plist)
    {
        plist = NULL;
        ptr = plist;
    }
    else
    {
        ptr = temp;
        temp = temp->next;
        plist = temp;
    }
    freeNode(ptr);
    return plist;
}
struct NodeType *deleteAtEnd(struct NodeType *plist)
{
    struct NodeType *temp;
```

```
    struct NodeType *ptr;
    temp = plist;
    if (temp->next == plist)
    {
        plist = NULL;
        ptr = plist;
    }
    else
    {
        while (temp->next != NULL)
        {
            ptr = temp;
            temp = temp->next;
        }
        ptr->next = NULL;
    }
    return plist;
}
struct NodeType *deleteAtPosition(struct NodeType *plist, int position)
{
    struct NodeType *ptr;
    struct NodeType *temp;
    int i = 1;
    temp = plist;
    while (i < position - 1)
    {
        temp = temp->next;
        i++;
    }
    ptr = temp->next;
    temp->next = ptr->next;
    free(ptr);
    return plist;
}
```