TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM
MÔN: **KĨ THUẬT LẬP TRÌNH**

HƯỚNG DẪN THỰC HÀNH

TUẦN 07

KĨ THUẬT CON TRỎ VÔ KIỂU

TP.HCM, ngày 20 tháng 02 năm 2023

MỤC LỤC

1 Khái niệm con trỏ vô kiểu:

Con trỏ vô kiểu là một kiểu dữ liệu đặc biệt, được sử dụng để tổng quát hóa đoạn mã, giúp có thể tái sử dụng cho nhiều kiểu dữ liệu khác nhau (Tương tự như Template trong C++).

Cú pháp: void* p;

Trong tuần này ta sẽ sử dụng kĩ thuật con trỏ vô kiểu để giúp trường key của các node trong danh sách liên kết có khả năng lưu trữ nhiều loại dữ liệu khác nhau.

Xem khai báo sau:

```
typedef struct node* ref;
struct node{
  void* key;
  ref next;
};
```

Các dòng mã có nghĩa một cấu trúc tên node gồm 2 trường, trường dữ liệu kiểu con trỏ vô kiểu là key, trường dữ liệu thứ hai là địa chỉ trỏ tới một node khác.

2 Các thao tác trong danh sách liên kết đơn.

Trong phần này, ta cũng có 12 thao tác cơ bản liên quan bao gồm: xây dựng danh sách liên kết, duyệt danh sách, hủy danh sách, thêm phần tử vào cuối danh sách, thêm phần tử vào giữa danh sách, xóa phần tử đầu, xóa phần tử cuối, xóa phần tử giữa, đếm số phần tử, chèn phần tử vào vị trí xác định, xóa phần tử tại vị trí xác định. Nhưng ta sẽ thực hiện với kiểu dữ liệu tổng quát là (void*).

2.1 Thao tác xây dựng danh sách.

Ở thao tác này, giả sử ta có danh sách liên kết rỗng (head = tail = NULL), và ta cần thêm một node p nào đó vào danh sách rỗng này.

Dòng	
0	#include <stdio.h>
1	#include <stdlib.h>
2	typedef struct node* ref;
3	struct node{
4	void* key;
5	ref next;
6	};

```
7
8  ref getNode(void* k, int dSize){
9     ref p;
10    p = (ref)malloc(sizeof(struct node));
11    if(p == NULL){
12       printf("Khong du bon ho\n");
13       exit(0);
14    }
15    p->key = new char[dSize];
16    p->next = NULL;
17    if(p->key == NULL){delete p; return NULL;}
18    memcpy(p->key, k, dSize);
19    return p;
20 }
21
22 void addFirst(ref& head, ref& tail, void* k, int dSize){
23    if(k == NULL){return;}
24    ref p = getNode(k, dSize);
25    if(p != NULL){
26       p->next = head;
27       head = p;
28    }
29 }
30
31 void main(){
32    double x;
33    int dSize = sizeof(double);
34    ref head = NULL, tail = NULL;
35    x = 1.5; addFirst(head, tail, &x, dSize);
36    x = 2.6; addFirst(head, tail, &x, dSize);
37    x = 3.7; addFirst(head, tail, &x, dSize);
38 }
```

Ta lưu ý do dùng (void*) nên ta cần thêm biến dSize để biết được kích thước của kiểu dữ liệu cụ thể là gì khi sử dụng các hàm này. Ngoài ra, để thuận tiện ta dùng hàm memcpy thực hiện sao chép từng byte qua trường key (Sử dụng kiểu char).

2.2 Thao tác hủy danh sách liên kết đơn

Việc hủy danh sách liên kết đơn trong trường hợp này cần quan tâm thêm tới trường dữ liệu key do ta sử dụng con trỏ trong trường này.

```
Dòng
```

```
1   void destroyList(ref& head){
2     ref p;
3     while(head){
4       p = head;
5       if(p->key != NULL) delete[] (char*)p->key;
6       head = head->next;
7       free(p);
8     }
9   }
10
10  void main(){
11    double x;
12    int dSize = sizeof(double);
13    ref head = NULL, tail = NULL;
14    x = 1.5; addFirst(head, tail, &x, dSize);
15    x = 2.6; addFirst(head, tail, &x, dSize);
16    x = 3.7; addFirst(head, tail, &x, dSize);
17    destroyList(head);
18  }
```

2.3 Thao tác tìm một phần tử dựa vào trường key trong danh sách

Trong thao tác này, ta có nhu cầu tìm kiếm xem có phần tử node nào có trường key **bằng với** giá trị ta cần tìm. Từ "bằng với" ở đây phải hiểu theo nghĩa tổng quát, có nghĩa là ta cần có một hàm định nghĩa thế nào là "**bằng với**". Vì kiểu dữ liệu của ta là (void*) nên khi định nghĩa các kiểu dữ liệu mới như PHANSO hay SOPHUC ta cũng cần cài đặt hàm so sánh. Bên dưới là đoạn mã tìm kiếm findList (Giả sử đã có struct Fraction).

```
Dòng
1   ref findList(ref head, void* dat, bool (*cmp)(void*, void*)){
2     ref p = head;
3     while(p != NULL){
4       if(cmp(p->key, dat)) break;
5       p = p->next;
8     }
      return p;
9   }
10
11  bool fracComp(void* p, void* q){
12    Fraction* r = (Fraction*)p, *s = (Fraction*)q;
13    int num1 = r->Numerator*s->Denominator;
14    int num2 = r->Denominator*s->Numerator;
```

```
15    return (num1 == num2);
16  }
17
18  void main(){
19    Fraction ps[] = {{1, 2}, {-2, 3}, {3, 4}, {-4, 5}, {5, 6}};
20    int n = sizeof(ps)/sizeof(ps[0]);
21    ref head = NULL, tail = NULL, p;
22    int dSize = sizeof(struct Fraction);
23    for(int i = 0; i < n; i++) addFirst(head, tail, &ps[i], dSize);
24    Fraction r1 = {5, 3}; p = findList(head, &r1, fracComp);
25    Fraction r2 = {-8, 10}; p = findList(head, &r2, fracComp);
26    destroyList(head);
27  }
```

2.4 Thêm phần tử vào sau một phần tử nào đó trong danh sách

Ở thao tác này, ta cũng có nhu cầu thêm phần tử node p vào phía sau phần tử node q nào đó trong danh sách. Tuy nhiên, trong trường hợp này, tham số đầu vào của hàm chỉ là các giá trị của trường key, vì vậy công việc đầu tiên ta cần là tìm phần tử q có trường key cần tìm, sau đó chèn node p vào.

```
Dòng
1   ref insertAfter(ref& head, void* X, void* Val, int dSize,
    bool(*cmp)(void*, void*)){
2     ref q = findList(head, X, cmp);
3     if(q == NULL) return NULL;
4     ref p = getNode(Val, dSize);
5     if(p != NULL)
6     {
7       p->next = q->next;
8       q->next = p;
9     }
10    return p;
11  }
12
13  void main(){
14    Fraction ps[] = {{1, 2}, {-2, 3}, {3, 4}, {-4, 5}, {5, 6}};
15    int n = sizeof(ps)/sizeof(ps[0]), dSize = sizeof(struct Fraction);
16    ref head = NULL, p = NULL, tail = NULL;
17    for(int i = 0; i < n; i++)
18      addFirst(head, tail, &ps[i], dSize);
19    Fraction r1 = {1, 2}, r3 = {16, 20}, r4 = {113, 17};
```

20	p = insertAfter(head, &r1, &r4, dSize, fracComp);
21	p = insertAfter(head, &r3, &r4, dSize, fracComp);
22	destroyList(head);
23	}

3 Bài tập.

Xem kĩ các ví dụ trên, sinh viên hãy tự luyện tập các thao tác còn lại (như trong tuần 05) cho danh sách liên kết đơn vô kiểu.

Dùng dslk đơn vô kiểu này, sinh viên **làm lại các thao tác như bài tập tuần 05** nhưng với hai kiểu "string" và PHANSO.