Lập trình tổng quát

Giới thiệu

- Lập trình tổng quát: tạo ra các thành phần phần mềm có thể dễ dàng sử dụng trong nhiều bài toán khác nhau
- Ví dụ:hàm memcpy() cho phép sao chép dữ liệu
  - void* memcpy(void* region1, const void* region2, size_t n);
  - Hàm memcpy có tham số là con trỏ void* nên có thể sao chép dữ liệu từ nhiều kiểu dữ liệu khác nhau
- Tổng quát: để sao chép dữ liệu, chúng ta chỉ cần biết địa chỉ, và kích thước vùng cần sao chép

memcpy

- Hàm mempcy() có thể được cài đặt là:

```
void* memcpy(void* region1,
             const void* region2,
             size_t n) {
    const char* first = (const char*) region2;
    const char* last = ((const char*) region2) + n;
    char* result = (char*) region1;
    while (first != last) *result++ = *first++;
    return result;
}
```

Lưu ý: hằng con trỏ (Pointer constant) và con trỏ hằng (constant pointer)

PC: không cho phép thay đổi địa chỉ. int* constant p=&x;

CP: không cho phép thay đổi giá trị. constant int*p;

Hàm tổng quát

- Trong một hàm tổng quát, dữ liệu nên được truyền theo cách tổng quát (bởi địa chỉ và kích thước)
- Nếu giải thuật đòi hỏi một hàm cụ thể để thao tác dữ liệu (ví dụ so sánh hai giá trị), hàm đó nên được truyền sử dụng con trỏ hàm
- Ví dụ: Hàm sắp xếp tổng quát qsort
  - Truyền dữ liệu cho hàm?
  - So sánh hai phần tử trong mảng?

Cài đặt (1)

- Mảng dữ liệu tổng quát nên được truyền thông qua các tham số
  - void *buf: địa chỉ của vùng nhớ chứa dữ liệu
  - int size: kích thước của mỗi phần tử
  - int total: số lượng các phần tử của mảng
- Giải thuật tìm kiếm cần một hàm so sánh hai phần tử. Các phần tử được truyền thông qua địa chỉ của nó. Sử dụng một con trỏ hàm để biểu diễn một hàm so sánh tổng quát
  - int (*compare) (void * item1, void * item2)

Cài đặt (2)

```
// return -1 if not found
int search(  void* buf,
             int size,
             int l, int r,
             void * item,
             int (*compare)(void*, void*)) {
   int i, res;
   if (r < l) return -1;
   i = (l + r)/2;
   res = compare(item, (char*)buf+(size*i) );
   if (res==0)
       return i;
   else if (res < 0)
       return search(buf, size, l, i-1, item, compare);
   else
       return search(buf, size, i+1, r, item, compare);
}
```

Sử dụng như thế nào?

```
int int_compare(void const* x, void const *y) {
   int m, n;
   m = *((int*)x);
   n = *((int*)y);
   if (m == n) return 0;
   return m>n?1:-1;
}
int main() {
   int a[100], i, res;
   int n = 100, item = 5;
   for (i=0; i<n; i++) a[i] = rand();
   qsort(a, n, sizeof(int), int_compare);
   res = search (a,sizeof(int),0,n-1,&item,int_compare);
}
```

Bài 1

- Phát triển hàm sắp xếp tổng quát của riêng bạn dựa trên giải thuật được đưa ra trong bài giảng 1
- Viết lại các chương trình trong bài giảng 1 sử dụng hàm sắp xếp tổng quát

Hướng dẫn

- Để đổi chỗ hai phần tử trong mảng, chúng ta cần xây dựng hàm đổi chỗ tổng quát như sau
  - void exch (void * buf, int size, int i, int j);

Giải pháp

```
void sort(void* a, int size, int l, int r,
                    int (*compare)(void*, void*)) {
   if (r <= l) return;
   int i = l-1, j = r;
   int p = l-1, q = r;
   while(1)      {
        while (compare((char*)a+(++i)*size, (char*)a+r*size) < 0 );
        while (compare((char*)a+r*size, (char*)a+(--j)*size) < 0 )
            if (j == l) break;
        if (i >= j) break;
        exch(a, size, i, j);
        if (compare((char*)a+i*size, (char*)a+r*size)==0)
           exch(a, size, ++p, i);
        if (compare((char*)a+j*size, (char*)a+r*size)==0)
           exch(a, size, --q, j);
   }
   exch(a, size, i, r);
   j = i - 1;
   i = i + 1;
   for (int k = l ; k <= p; k++) exch(a, size, k, j--);
   for (int k = r-1; k >= q; k--) exch(a, size, k, i++);
   sort(a, size, l, j, compare);
   sort(a, size, i, r, compare);
}
```

Kiểu dữ liệu tổng quát

- Làm thế nào chúng ta có thể tạo ra một vùng lưu trữ dữ liệu tổng quát, nơi mà dữ liệu có thể ở dạng số nguyên, số thực, kí tự hoặc thậm chí là một bản ghi
- Kiểu dữ liệu tổng quát nên hiệu quả để phát triển một cấu trúc dữ liệu trừu tượng (ADT) trong C như danh sách liên kết, cây nhị phân…
- Kiểu hợp (Union) có thể là một cách hữu hiệu để thực thi một cấu trúc dữ liệu tổng quát
- Union can be an interesting way to implement a generic data type.

Jval (libfdr lib)

```
typedef union {
        int i;
        long l;
        float f;
        double d;
        void *v;
        char *s;
        char c;
} Jval;
```

- Jval có thể được sử dụng để lưu trữ các loại dữ liệu khác nhau, ví dụ:

```
Jval a, b;
a.i = 5;
b.f = 3.14;
```

Các hàm khởi tạo

- Để đơn giản việc sử dụng Jval, một vài hàm khởi tạo dữ liệu được xây dựng
  - Jval new_jval_i(int);
  - Jval new_jval_f(float);
  - Jval new_jval_d(double);
  - Jval new_jval_s(char *);
- Ví dụ:

```
Jval a, b;
a = new_jval_i(5);
b = new_jval_f(3.14);
```

Các hàm truy cập

- Để đọc giá trị từ một vùng nhớ tổng quát, các hàm truy cập tới kiểu dữ liệu cụ thể, sẽ được sử dụng
  - int jval_i(Jval);
  - float jval_f(Jval);
  - double jval_d(Jval);
  - char* jval_s(Jval);
- Ví dụ:

```
Jval a, b;
a = new_jval_i(5);
b = new_jval_float(3.14);
printf(“%d”, jval_i(a));
printf(“%f”, jval_f(a));
```

Thực thi

```
Jval new_jval_i(int i) { Jval j; j.i = i; return j; }
Jval new_jval_l(long l) { Jval j; j.l = l; return j; }
Jval new_jval_f(float f) { Jval j; j.f = f; return j; }
Jval new_jval_d(double d) { Jval j; j.d = d; return j; }
Jval new_jval_v(void *v) { Jval j; j.v = v; return j; }
...

int jval_i(Jval j) { return j.i; }
long jval_l(Jval j) { return j.l; }
float jval_f(Jval j) { return j.f; }
double jval_d(Jval j) { return j.d; }
void *jval_v(Jval j) { return j.v; }
...
```

Bài 2

- Viết lại hàm tìm kiếm và sắp xếp tổng quát sử dụng Jval, như sau:
  - void sort_gen (Jval a[], int l, int r, int (*compare) (Jval*, Jval*));
  - int search_gen (Jval a[], int l, int r, Jval item, int (*compare)(Jval*, Jval*));

Hướng dẫn

- Sau khi tạo ra hàm sắp xếp và tìm kiếm tổng quát, bạn có thể tạo ra các hàm xử lý đối với một kiểu dữ liệu cụ thể, như:

```
int compare_i(Jval* a, Jval* b);
void sort_i (Jval a[], int l, int r);
int search_i (Jval a[], int l, int r, int x);
Jval* create_array_i (int n);
```

Solution

```
int compare_i(Jval* a, Jval* b) {
   if ( jval_i(*a)==jval_i(*b) ) return 0;
   if ( jval_i(*a) < jval_i(*b) ) return -1;
   else return 1;
}
void sort_i (Jval a[], int l, int r) {
   sort_gen(a, l, r, compare_i);
}
int search_i (Jval a[], int l, int r, int x) {
   return search_gen(a, l, r, new_jval_i(x), compare_i);
}
Jval* create_array_i (int n) {
   Jval * array = (Jval *) malloc(sizeof(Jval)*n);
   for (i=0; i<n; i++) array[i] = new_jval_i( rand() );
   return array;
}
```