Generic programming

ducnh-fit@mail.hut.edu.vn

Introduction

- Generic programming is about generalizing software components so that they can be easily reused in a wide variety of situations.
- As a simple example of generic programming, the memcpy() function of the C standard library is a generic function to copy data from a container to another.
    - void* memcpy(void* region1, const void* region2, size_t n);
- The memcpy() function is already generalized to some extent by the use of void* so that the function can be used to copy arrays of different kinds of data.
- Generally, to copy data we need to know only the address and the size of the container to copy.

memcpy

- An implementation of memcpy() might look like the following:

```
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;
}
```

Generic functions

- In a generic function, data should be passed in a generic way (by address and size).
- If the algorithm demands a specific function to manipulate data (e.g.., compare two values), such a function should be passed using a function pointer.
- Example: A generic search function on an array.
  - How to pass data to this function ?
  - How the algorithm can detect if two data items in the array is equal or not ?

Implementation (1)

- A generic data array should be passed as the following parameters
  - void * buf: the address of the buffer containing the array's data
  - int size: the size of a data item in the array
  - int total: the total number of data items in the array
- The search algorithm need also a function to compare the data items in the array for searching. A data item passed to such a function via its address. Use a function pointer to represent a generic comparison algorithm.
  - int (*compare) (void * item1, void * item2)

Implementation (2)

```
// return -1 if not found
int search(  void* buf,
             int size,
             int l, int r,
             void * item,
             int (*compare)(void*, void*)) {
   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);
}
```

How to use ?

```
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];
  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, int_compare);
}
```

Quiz 1

- Develop yourself a generic sort function based on the algorithm given in lesson 1.
- Rewrite your programs in lesson 1 using the generic sort function.

Instruction

- In order to exchange two items in the array, we need to develop a generic exchange function as the following
  - void exch (void * buf, int size, int i, int j);

Solution

```
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);
}
```

Generic data type

- How we can create a generic data container where the data item can be either integer, float, char and event a records.
- Generic data type should be useful to develop a generic ADT in C such as linked list, binary tree, etc.
- 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 can be used to store different kinds of data as the following:

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

Constructor functions

- To simply the usage of Jval, some data constructor functions are created
  - Jval new_jval_i(int);
  - Jval new_jval_f(float);
  - Jval new_jval_d(double);
  - Jval new_jval_s(char *);
- Example:

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

Access functions

- To read value from a generic, access functions can be used for specific types
  - int jval_i(Jval);
  - float jval_f(Jval);
  - double jval_d(Jval);
  - char* jval_s(Jval);
- Example:

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

Implementation

```
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; }
...
```

Quiz 2

- Rewrite the generic sorting and searching functions using Jval to represent the generic data container as the following
  - 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*));

Instruction

- After creating the generic sorting and searching functions, you can create functions to manipulate a specific data as the following.

```
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;
}
```