Указатели на функции

Теги: Указатели на функции, Свёртки, фильтры, отображения, qsort, bsearch.

Указатели на функции

Как уже обсуждалось ранее функции – это набор команд, которые расположены в соответствующей области памяти. Вызов функции – это сохранение состояния, передача аргументов и переход по адресу, где располагается функция. В си есть возможность создавать указатели на функции. Указатели на функции позволяют упростить решение многих задач. Совместно с void указателями можно создавать функции общего назначения (например, сортировки и поиска). Указатели позволяют создавать функции высших порядков (функции, принимающие в качестве аргументов функции): отображение, свёртки, фильтры и пр. Указатели на функции позволяют уменьшать цикломатическую сложность программ, создавать легко масштабируемые конструкции. Рассмотрим пример. Создадим функцию и указатель на эту функцию

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

int sum(int a, int b) {
    return a + b;
}

void main () {
    //Объявляем указатель на функцию
    int (*fptr)(int, int) = NULL;
    int result;
    //Присваиваем указателю значение - адрес функции
    //Это похоже на работу с массивом: операцию взятия адреса использовать не нужно
    fptr = sum;
    //Вызов осуществляется также, как и обычной функции
    result = fptr(10, 40);
    printf("%d", result);
    getch();
}
```

Синтаксис объявления указателей на функцию

```
<возвращаемый тип> (* <имя>)(<тип аргументов>);
```

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

int dble(int a) {
    return 2*a;
}

int deleteEven(int a) {
    if (a % 2 == 0) {
        return 0;
    } else {
        return a;
    }
}

//Функция принимает массив, его размер и указатель на функцию,
//которая далее применяется ко всем элементам массива
void map(int *arr, unsigned size, int (*fun)(int)) {
    unsigned i;
    for (i = 0; i < size; i++) {
        arr[i] = fun(arr[i]);
    }
}

void main () {
    int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    unsigned i;
    map(a, 10, deleteEven);
    for (i = 0; i < 10; i++) {
```

```
        printf("%d ", a[i]);
    }
    map(a, 10, dble);
    printf("\n");
    for (i = 0; i < 10; i++) {
        printf("%d ", a[i]);
    }
    getch();
}
```

В этом примере мы создали функцию отображения, которая применяет ко всем элементам массива функцию, которая передаётся ей в качестве аргумента. Когда мы вызываем функцию map, достаточно просто передавать имена функций (они подменяются указателями). Запишем теперь функцию map, которая получает в качестве аргумента массив типа void:

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

void dbleInt(void *a) {
    *((int*) a) *= 2;
}

void deleteEvenInt(void* a) {
    int tmp = *((int*) a);
    if (tmp % 2 == 0) {
        *((int*) a) = 0;
    }
}

void dbleDouble(void *a) {
    *((double*) a) *= 2.0;
}

void deleteEvenDouble(void* a) {
    int tmp = *((double*) a);
    if (tmp % 2 == 0) {
        *((double*) a) = 0;
    }
}

//Функция принимает массив, его размер, размер одного элемента и указатель на функцию,
//которая далее применяется ко всем элементам массива
void map(void *arr, unsigned num, size_t size, void (*fun)(void *)) {
    unsigned i;
    char *ptr = (char*) arr;
    for (i = 0; i < num; i++) {
        fun((void*) (ptr + i*size));
    }
}

void main () {
    int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    double b[] = {1., 2., 3., 4., 5., 6., 7., 8., 9., 10.};
    unsigned i;
    //Работаем с массивом типа int
    map(a, 10, sizeof(int), deleteEvenInt);
    for (i = 0; i < 10; i++) {
        printf("%d ", a[i]);
    }
    map(a, 10, sizeof(int), dbleInt);
    printf("\n");
    for (i = 0; i < 10; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
    //Работаем с массивом типа double
    map(b, 10, sizeof(double), deleteEvenDouble);
    for (i = 0; i < 10; i++) {
        printf("%.3f ", b[i]);
    }
    map(b, 10, sizeof(double), dbleDouble);
    printf("\n");
    for (i = 0; i < 10; i++) {
        printf("%.3f ", b[i]);
    }
    getch();
}
```

Вот где нам понадобились указатели типа void. Так как map получает указатель на функцию, то все функции должны иметь одинаковые аргументы и возвращать один и тот же тип. Но аргументы функций должны быть разного типа, поэтому мы делаем их типа void. Функция map получает указатель типа void (*)(void*), поэтому ей теперь можно передавать любую из четырёх функций.
Пример другой функции: функция filter получает указатель на массив и возвращает размер нового массива, оставляя в нём только те элементы, для которых переданный предикат возвращает логическую истину (предикат – функция, которая возвращает истину или ложь). Сначала напишем для массива типа int:

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

int isOdd(int a) {
    return (a % 2 != 0);
}
```

```
8
9   int isGtThree(int a) {
10      return a > 3;
11  }
12
13  unsigned int filter(int *arr, unsigned size, int (*pred)(int), int** out) {
14      unsigned i;
15      unsigned j; //размер возвращаемого масива
16      *out = (int*) malloc(sizeof(int)*size);
17      for (i = 0, j = 0; i < size; i++) {
18          if (pred(arr[i])) {
19              (*out)[j] = arr[i];
20              j++;
21          }
22      }
23      *out = (int*) realloc(*out, j*sizeof(int));
24      return j;
25  }
26
27  void main () {
28      int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
29      int *aOdd = NULL;
30      int *aGtThree = NULL;
31      unsigned i;
32      unsigned size;
33
34      size = filter(a, 10, isOdd, &aOdd);
35      for (i = 0; i < size; i++) {
36          printf("%d ", aOdd[i]);
37      }
38      printf("\n");
39      size = filter(a, 10, isGtThree, &aGtThree);
40      for (i = 0; i < size; i++) {
41          printf("%d ", aGtThree[i]);
42      }
43      free(aOdd);
44      free(aGtThree);
45      getch();
46  }
```

Теперь для массива типа void

```
1   #include <conio.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <string.h>
5   #include <math.h>
6
7   int isOddInt(void *a) {
8       return (*((int*) a) % 2 != 0);
9   }
10
11  int isGtThreeInt(void* a) {
12      return *((int*) a) > 3;
13  }
14
15  int isOddDouble(void* a) {
16      return (int)*((double*) a) / 2 != 0;
17  }
18
19  int isGtThreeDouble(void* a) {
20      return *((double*) a) > 3.0;
21  }
22
23  unsigned int filter(void *arr, unsigned num, size_t size, int (*pred)(void*), void** out) {
24      unsigned i;
25      unsigned j; //размер возвращаемого масива
26      char* ptrIn = (char*) arr;
27      char* ptrOut = NULL;
28      *out = (void*) malloc(num*size);
29      ptrOut = (char*) (*out);
30      for (i = 0, j = 0; i < num; i++) {
31          if (pred(ptrIn + i*size)) {
32              memcpy(ptrOut + j*size, ptrIn + i*size, size);
33              j++;
34          }
35      }
36      *out = (void*) realloc(*out, j*size);
37      return j;
38  }
39
40  void main () {
41      int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
42      double b[] = {1., 2., 3., 4., 5., 6., 7., .8, .9, 10.,};
43      int *aOdd = NULL;
44      int *aGtThree = NULL;
45      double *bOdd = NULL;
46      double *bGtThree = NULL;
47      unsigned i;
48      unsigned size;
49
50      size = filter(a, 10, sizeof(int), isOddInt, (void**)&aOdd);
```

```
    for (i = 0; i 7lt; size; i++) {
        printf("%d ", aOdd[i]);
    }
    printf("\n");
    size = filter(a, 10, sizeof(int), isGtThreeInt, (void**)&aGtThree);
    for (i = 0; i < size; i++) {
        printf("%d ", aGtThree[i]);
    }
    printf("\n");
    size = filter(b, 10, sizeof(double), isOddDouble, (void**)&bOdd);
    for (i = 0; i < size; i++) {
        printf("%.3f ", bOdd[i]);
    }
    printf("\n");
    size = filter(b, 10, sizeof(double), isGtThreeDouble, (void**)&bGtThree);
    for (i = 0; i < size; i++) {
        printf("%.3f ", bGtThree[i]);
    }

    free(aOdd);
    free(bOdd);
    free(aGtThree);
    free(bGtThree);
    getch();
}
```

Ещё одна функция – свёртка. Она получает в качестве аргументов массив и функцию от двух аргументов. Эта функция действует следующим образом: сначала она применяется к первым двум аргументам, затем она применяется к третьему аргументу и результату предыдущего вызова, затем к четвёртому аргументу и результату предыдущего вызова и т.д. С помощью свёртки можно, например, найти сумму всех элементов массива, максимальный элемент массива, факториал числа и т.п.

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

int sum(int a, int b) {
    return a + b;
}

int maxx(int a, int b) {
    return (a > b)? a: b;
}

int mul(int a, int b) {
    return a*b;
}

void fold(int *arr, unsigned size, int (*fun)(int, int), int *acc) {
    unsigned i;
    *acc = fun(arr[0], arr[1]);
    for (i = 2; i < size; i++) {
        *acc = fun(*acc, arr[i]);
    }
}

void main () {
    int a[] = {1, 2, 3, 4, 5, 10, 9, 8, 7, 6};
    int result;

    fold(a, 10, sum, &result);
    printf("%d\n", result);
    fold(a, 10, maxx, &result);
    printf("%d\n", result);
    fold(a, 10, mul, &result);
    printf("%d\n", result);

    getch();
}
```

Последний пример: функция сортировки вставками для массива типа void. Так как тип массива не известен, то необходимо передавать функцию сравнения.

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

int cmpIntDesc(void *a, void* b) {
    return *((int*) a) < *((int*) b);
}

int cmpIntAsc(void *a, void* b) {
    return *((int*) a) > *((int*) b);
}

int cmpDoubleAsc(void *a, void* b) {
    return *((double*) a) < *((double*) b);
}

int cmpDoubleDesc(void *a, void* b) {
    return *((double*) a) > *((double*) b);
}
```

```
21
22  void insertionSort(void* arr, unsigned num, size_t size, int (*cmp)(void *a, void *b)) {
23      unsigned i, j;
24      char *ptr = (char*) arr;
25      char *tmp = (char*) malloc(size);
26      for (i = 1; i < num; i++) {
27          if (cmp(ptr + i*size, ptr + (i-1)*size)) {
28              j = i;
29              while (cmp(ptr + j*size, ptr + (j-1)*size) && j > 0) {
30                  memcpy(tmp, ptr + j*size, size);
31                  memcpy(ptr + j*size, ptr + (j-1)*size, size);
32                  memcpy(ptr + (j-1)*size, tmp, size);
33                  j--;
34              }
35          }
36      }
37  }
38
39  void main () {
40      int a[] = {1, 2, 3, 4, 5, 10, 9, 8, 7, 6};
41      double b[] = {1, 2, 3, 4, 5, 10, 9, 8, 7, 6};
42      int i;
43
44      insertionSort(a, 10, sizeof(int), cmpIntAsc);
45      for (i = 0; i < 10; i++) {
46          printf("%d ", a[i]);
47      }
48      printf("\n");
49      insertionSort(a, 10, sizeof(int), cmpIntDesc);
50      for (i = 0; i < 10; i++) {
51          printf("%d ", a[i]);
52      }
53      printf("\n");
54      insertionSort(b, 10, sizeof(double), cmpDoubleAsc);
55      for (i = 0; i < 10; i++) {
56          printf("%.3f ", b[i]);
57      }
58      printf("\n");
59      insertionSort(b, 10, sizeof(double), cmpDoubleDesc);
60      for (i = 0; i < 10; i++) {
61          printf("%.3f ", b[i]);
62      }
63
64      getch();
65  }
```

Массив указателей на функции

Массив указателей на функции определяется точно также, как и обычный массив – с помощью квадратных скобок после имени:

```
1   #include <conio.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <math.h>
5
6   #define ERROR_DIV_BY_ZERO -2
7   #define EPSILON 0.000001f
8
9   float doSum(float a, float b) {
10      return a + b;
11  }
12
13  float doSub(float a, float b) {
14      return a - b;
15  }
16
17  float doMul(float a, float b) {
18      return a * b;
19  }
20
21  float doDiv(float a, float b) {
22      if (fabs(b) <= EPSILON) {
23          exit(ERROR_DIV_BY_ZERO);
24      }
25      return a / b;
26  }
27
28  void main() {
29      float (*menu[4])(float, float);
30      int op;
31      float a, b;
32      menu[0] = doSum;
33      menu[1] = doSub;
34      menu[2] = doMul;
35      menu[3] = doDiv;
36      printf("enter a: ");
37      scanf("%f", &a);
38      printf("enter b: ");
39      scanf("%f", &b);
40      printf("enter operation [0 - add, 1 - sub, 2 - mul, 3 - div]");
41      scanf("%d", &op);
```

```
    if (op >= 0 && op < 4) {
        printf("%.6f", menu[op](a, b));
    }
    getch();
}
```

Точно также можно было создать массив динамически

```
void main() {
    float (**menu)(float, float) = NULL;
    int op;
    float a, b;
    menu = (float(**)(float, float)) malloc(4*sizeof(float(*)(float, float)));
    menu[0] = doSum;
    menu[1] = doSub;
    menu[2] = doMul;
    menu[3] = doDiv;
    printf("enter a: ");
    scanf("%f", &a);
    printf("enter b: ");
    scanf("%f", &b);
    printf("enter operation [0 - add, 1 - sub, 2 - mul, 3 - div]");
    scanf("%d", &op);
    if (op >= 0 && op < 4) {
        printf("%.6f", menu[op](a, b));
    }
    free(menu);
    getch();
}
```

Часто указатели на функцию становятся громоздкими. Работу с ними можно упростить, если ввести новый тип. Предыдущий пример можно переписать так

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define ERROR_DIV_BY_ZERO -2
#define EPSILON 0.000001f

typedef float (*operation)(float, float);

float doSum(float a, float b) {
    return a + b;
}
float doSub(float a, float b) {
    return a - b;
}
float doMul(float a, float b) {
    return a * b;
}
float doDiv(float a, float b) {
    if (fabs(b) <= EPSILON) {
        exit(ERROR_DIV_BY_ZERO);
    }
    return a / b;
}

void main() {
    operation *menu = NULL;
    int op;
    float a, b;
    menu = (operation*) malloc(4*sizeof(operation));
    menu[0] = doSum;
    menu[1] = doSub;
    menu[2] = doMul;
    menu[3] = doDiv;
    printf("enter a: ");
    scanf("%f", &a);
    printf("enter b: ");
    scanf("%f", &b);
    printf("enter operation [0 - add, 1 - sub, 2 - mul, 3 - div]");
    scanf("%d", &op);
    if (op >= 0 && op < 4) {
        printf("%.6f", menu[op](a, b));
    }
    free(menu);
    getch();
}
```

Ещё один пример: функция any возвращает 1, если в переданном массиве содержится хотя бы один элемент, удовлетворяющий условию pred и 0 в противном случае.

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

typedef int (*Predicat)(void*);

int isBetweenInt(void* a) {
    return *((int*) a) > 10 && *((int*) a) < 12;
```

```
}

int isBetweenDouble(void* a) {
    return *((double*) a) > 10.0 && *((double*) a) < 12.0;
}

int any(void* arr, unsigned num, size_t size, Predicat pred) {
    unsigned i;
    char* ptr = (char*) arr;
    for (i = 0; i < num; i++) {
        if (pred(ptr + i*size)) {
            return 1;
        }
    }
    return 0;
}

void main() {
    int a[10] = {1, 1, 2, 2, 3, 0, 1, 2, 1, 3};
    double b[10] = {1, 2, 11, 2, 3, 4, 5, 6, 7, 10};

    printf("has 'a' any value > 10 and < 12? %d\n", any(a, 10, sizeof(int), isBetweenInt));
    printf("has 'b' any value > 10 and < 12? %d", any(b, 10, sizeof(double), isBetweenDouble));

    getch();
}
```

qsort и bsearch

В библиотеке stdlib си имеется несколько функций, которые получают в качестве аргументов указатели на функции. Функция qsort получает такие же аргументы, как и написанная нами функция insertionSort: массив типа void, размер массива, размер элемента и указатель на функцию сравнения. Давайте посмотрим простой пример – сортировка массива строк:

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

#define SIZE 10

//qsort передаёт функции сравнения указатели на элементы.
//но наш массив - это массив указателей, так что qsort будет
//передавать указатели на указатели
int cmpString(const void* a, const void* b) {
    return strcmp(*(const char**) a, *(const char**) b);
}

void main() {
    char *words[SIZE];
    char buffer[128];
    int i, length;

    printf("Enter %d words:", SIZE);
    for (i = 0; i < SIZE; i++) {
        scanf("%127s", buffer);
        length = strlen(buffer);
        words[i] = (char*) malloc(length+1);
        strcpy(words[i], buffer);
    }

    printf("\n");
    qsort(words, SIZE, sizeof(char*), cmpString);
    for (i = 0; i < SIZE; i++) {
        printf("%s\n", words[i]);
        free(words[i]);
    }
    getch();
}
```

Функция bsearch проводит бинарный поиск в отсортированном массиве и получает указатель на функцию сравнения, такую же, как и функция qsort. В случае, если элемент найден, то она возвращает указатель на этот элемент, если элемент не найден, то NULL.

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

int cmpInt(const void* a, const void* b) {
    return *(int*) a - *(int*) b;
}

void main() {
    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int elm;
    int *index;
    printf("Enter number to search: ");
    scanf("%d", &elm);
    index = (int*) bsearch(&elm, a, 10, sizeof(int), cmpInt);
    if (index == NULL) {
        printf("element not found");
    } else {
        printf("index = %d", (index - a));
```

```
    }
    getch();
}
```

Указатели типа void

Об аргументах функции