数组与广义表

本文档需要用到的编程相关知识点：

C语言：struct（结构体）、typedef、malloc（free）、va_list。

本文档的参考教材：

数据结构（C语言版） 严蔚敏

因为数组的维数是不定的，所以相关的操作函数的参数个数不定，因此数组的实现会用到 va_list 来解决变参问题 。

一. 数组的基本操作实现

1.va_list简介

va_list 是在C语言中解决变参问题的一组宏，变参问题是指函数参数的个数不定，可以是传入一个参数也可以是多个; 可变参数中的每个参数的类型可以不同,也可以相同; 可变参数的每个参数并没有实际的名称与之相对应，使用起来非常灵活。

首先通过一个简单的例子来了解一下 va_list 的具体用法：

```c
#include <stdarg.h>
#include <stdio.h>

int AveInt(int, ...);  // 求平均值函数原型

int main(void)
{
    printf("%d\n", AveInt(2, 2, 3));  // 第一个参数2指明了后续参数个数为两个
    printf("%d\n", AveInt(4, 2, 4, 6, 8));  // 第一个参数4指明了后续参数个数为四个

    return 0;
}

int AveInt(int v, ...)  // 求平均值函数，v为...所代表的参数的个数
{
    int ReturnValue = 0;
    int i = v;

    va_list
        ap;  // 首先在函数里定义一具VA_LIST型的变量，这个变量是指向参数的指针
    va_start(ap, v);  // 然后用VA_START宏初始化变量刚定义的VA_LIST变量

    // 然后用VA_ARG返回可变的参数，VA_ARG的第二个参数是你要返回的参数的类型
    // 如果函数有多个可变参数的，依次调用VA_ARG获取各个参数
    while (i > 0) {
        ReturnValue += va_arg(ap, int);
        i--;
```

```
    }
    va_end(ap);  // 最后用VA_END宏结束可变参数的获取

    return ReturnValue /= v;
}

```

从示例程序可以看出：va_list 可以看成是 C++ 函数重载的更一般形式。下面详细讲述C语言函数形参的相关知识及 va_list 的原理。

C语言的函数形参是从右向左压入堆栈的，以保证栈顶是第一个参数，而且x86平台内存分配顺序是从高地址到低地址。因此函数 AveInt(int var1, int var2, ..., int varN) 内存分配大致上是这样的：(可变参数在中间)

栈区:
|栈顶　　低地址
|第一个参数 var1　　<-- &v
|第二个参数 var2　　<-- va_start(ap, v)后 ap 指向地址
|...
|函数的最后一个参数 varN
|...
|函数的返回地址
|...
|栈底　　高地址

在理解了C语言的函数参数存储机制后，下面具体讲述 va_list 的原理：（以示例程序代码为例）

va_list ap：定义一个 va_list 变量 ap 。

va_start(ap, v)：执行ap = (va_list)&v + _INTSIZEOF(v)，ap 指向参数 v 之后的那个参数的地址，即 ap 指向第一个可变参数在堆栈的地址。

va_arg(ap, t)：执行(*(t *)((ap += _INTSIZEOF(t)) - _INTSIZEOF(t)))取出当前 ap 指针所指的值，并使 ap 指向下一个参数。 ap + = sizeof(t 类型)：让 ap 指向下一个参数的地址。然后返回 ap - sizeof(t 类型) 的 t 类型* 指针，这正是第一个可变参数在堆栈里的地址。然后 用 * 取得这个地址的内容。

va_end(ap)：清空 va_list ap 。

使用VA_LIST应该注意的问题：

（1）因为va_start, va_arg, va_end等被定义成宏，所以它显得很愚蠢，可变参数的类型和个数完全在该函数中由程序代码控制，它并不能智能地识别不同参数的个数和类型。也就是说，你想实现智能识别可变参数的话是要通过在自己的程序里作判断来实现的。

（2）另外，因为编译器对可变参数的函数原型检查不够严格，这不利于我们写出高质量的代码。
（3）由于参数的地址用于VA_START宏，所以参数不能声明为寄存器变量，或作为函数或数组类型。

---

2. 数组的顺序存储表示和实现

数组的顺序存储表示和实现，参照教材93页。

```c
/**
 * 数组的顺序存储表示和实现
 * 参照教材93页
 * */

// <stdarg> 为标准头文件，提供宏 va_start、 va_arg、 va_end，用于存取变长参数表
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

/**
 * 宏定义及类型定义
 * */
#define OK 1
#define ERROR -1
#define OVERFLOW -1
#define MAX_ARRAY_DIM 8  // 假设数组维数的最大值为8

typedef int ElemType;
typedef int Status;

/**
 * 定义顺序数组
 * */
typedef struct {
    ElemType *base;  // 数组元素基址，由InitArray分配
    int dim;         // 数组维数
    int *bounds;  // 数组维界基址，由InitArray分配（存放每一维的个数）
    int *constants;  // 数组映像函数常量基址，由InitArray分配
} Array;

/**
 * 初始化数组，若维数dim和随后的各维长度合法，则构造相应的数组A，并返回OK
 */
Status InitArray(Array &A, int dim, ...)
{
    va_list ap;
    int i;
    int elemtotal = 1;  // 保存数组的元素个数
    if (dim < 1 || dim > MAX_ARRAY_DIM)
        return ERROR;
    A.dim = dim;
    A.bounds = (int *)malloc(dim * sizeof(int));
    if (!A.bounds)
        exit(OVERFLOW);

    va_start(ap, dim);  // ap为va_list类型，是存放变长参数信息的数组
    for (i = 0; i < dim; i++) {
        A.bounds[i] = va_arg(ap, int);  // 读入一个变长参数
        if (A.bounds[i] < 0)
            return ERROR;
        elemtotal *= A.bounds[i];
    }

    va_end(ap);
```

```
    A.base = (ElemType *)malloc(elemtotal * sizeof(ElemType));
    if (!A.base)
        exit(OVERFLOW);

    // 下面求映像函数Ci,并存入A.constants[i-1], i=1,...,dim
    A.constants = (int *)malloc(dim * sizeof(int));
    if (!A.constants)
        exit(OVERFLOW);
    A.constants[dim - 1] = 1;  // L=1，指针的增减以元素的大小为单位
    for (i = dim - 2; i >= 0; i--)  // 对应于93页顶上那个东西
        A.constants[i] = A.bounds[i + 1] * A.constants[i + 1];

    return OK;
}

/**
 * 销毁数组A
 */
Status DestroyArray(Array &A)
{
    if (!A.base)
        return ERROR;
    free(A.base);
    A.base = NULL;

    if (!A.bounds)
        return ERROR;
    free(A.bounds);
    A.bounds = NULL;

    if (!A.constants)
        return ERROR;
    free(A.constants);
    A.constants = NULL;

    return OK;
}

/**
 * 若ap指示的各下标值合法，则求出该元素在A中的相对地址off
 */
Status Locate(Array A, va_list ap, int &off)
{
    int i, ind;
    off = 0;
    for (i = 0; i < A.dim; i++) {
        ind = va_arg(ap, int);
        if (ind < 0 || ind >= A.bounds[i])
            return OVERFLOW;
        off += A.constants[i] * ind;
    }

    return OK;
}

/**
 * A是n维数组，e为元素变量，随后是n个下标值，若下标不越界，则e赋值为所指定的A的元素值，
并返回OK
```

```
 */
Status Value(Array A, ElemType &e, ...)
{
    int result;
    int off;
    va_list ap;
    va_start(ap, e);
    result = Locate(A, ap, off);
    if (OVERFLOW == result)
        return ERROR;
    e = *(A.base + off);

    return OK;
}

/**
 * 若下标不越界，将e的值赋给所指定的A的元素，并返回OK
 */
Status Assign(Array &A, ElemType e, ...)
{
    int result, off;
    va_list ap;
    va_start(ap, e);
    result = Locate(A, ap, off);
    if (OVERFLOW == result)
        return ERROR;
    *(A.base + off) = e;

    return OK;
}
```

数组测试程序：（测试程序代码放在实现代码下方即可）

```
int main(void)
{
    Array arr1;  // 定义一个数组arr1
    InitArray(arr1, 3, 3, 3, 3);

    // 为3*3*3维的数组赋值，依次赋为1~27
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            for (int k = 0; k < 3; k++) {
                ElemType ele = i * 9 + j * 3 + k + 1;
                Assign(arr1, ele, i, j, k);
            }

    // 将数组中的每个元素的值依次输出
    for (int i = 0; i < 3; i++) {
        printf("第%d层元素为: \n", i + 1);
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                ElemType ele = 0;
                Value(arr1, ele, i, j, k);
                printf("%d  ", ele);
            }
```

```
            printf("\n");
        }
        printf("\n");
    }

    return 0;
}
```

测试程序运行结果为：

第1层元素为:
1 2 3
4 5 6
7 8 9

第2层元素为:
10 11 12
13 14 15
16 17 18

第3层元素为:
19 20 21
22 23 24
25 26 27

二. 广义表的基本操作实现

Author: XF

Finished time: 2019/11/17