C深度剖析

数据类型

本质：固定内存大小的别名

变量

本质：连续存储空间的别名

```
#include <stdio.h>

typedef int INT32;
typedef unsigned char BYTE ;
typedef struct _demo {
    short s;
    BYTE b1;
    BYTE b2;
    INT32 i;
}DEMO;

int main()
{
    INT32 i32;
    BYTE byte;
    DEMO d;

    printf("%d,%d\n", sizeof(INT32), sizeof(i32));
    printf("%d,%d\n", sizeof(BYTE), sizeof(byte));
    printf("%d,%d\n", sizeof(DEMO), sizeof(d));
    return 0;
}
```

关键字

1.auto

> 编译器默认所有局部变量都是auto属性，在栈上分配空间
>
> 如果修饰全局变量报错

2.static

> 静态属性+作用域限定符
>
> 变量在程序的静态区分配空间

> 作用域仅限于定义它的文件中
>
> 静态局部变量只会被初始化一次

3.register

> 将变量存到寄存器中
>
> 不能用&获取register变量的地址，因为寄存器不在内存中
>
> 修饰全局则报错，因为全局变量存在于整个程序运行期间，全局变量长期要放于寄存器中，最终都分配后无寄存器可用

```
#include <stdio.h>
auto int g = 9; // error

register int m=0;//error

int main()
{
    auto int i = 0;
    register int j = 0;
    static int k = 0;

    printf("%x\n", &i);
    printf("%x\n", &j);//error

    printf("%x\n", &k);

    return 0;
}
```

```
#include <stdio.h>

void f1() {
    int i = 0;
    i++;
    printf("%d\n", i);
}

void f2() {
    static int i = 0;
    i++;
    printf("%d\n", i);
}

int main()
{
    int i = 0;
    for (i = 0; i < 5; ++i) {
        f1();
    }
    for (i = 0; i < 5; ++i) {
        f2();
    }
}
```

4.if

> bool型变量应该直接出现于条件中，不要进行比较，不同的编译器对true定义不同，false始终为0
>
> 普通变量和0比较时，0应该出现在比较符号左边
>
> float变量不能直接和0比，需要定义精度

```
#include <stdio.h>

typedef enum _bool {
    TRUE = -1,
    FALSE = 0
}BOOL; //枚举定义bool



int main() {
    BOOL b = TRUE;
    int i = -1;

    if (b) {
        printf("ok\n");
    }
    else {
        pritnf("error\n");
    }

    float f = 5.0;
    if ((5 - E <= f) && (5 + E <= f)) {
    }

    return 0;
}
```

5.switch

case中的值只能是整型或字符型

> 对比
> if适合范围型
> switch适用离散值，对多分支简洁

6.do,while,for

> do先执行后判断，至少执行一次
> while先判断后执行，可能不执行
> for先判断后执行，比while简洁

```c
#include <stdio.h>

int f1(int n)
{
    int ret = 0;
    int i = 0;

    for(i=1; i<=n; i++)
    {
        ret += i;
    }

    return ret;
}

int f2(int n)
{
    int ret = 0;

    while( (n > 0) && (ret += n--) );

    return ret;
}

int f3(int n)
{
    int ret = 0;

  //  if( n > 0 )
  //  {
        do
        {
            ret += n--;
        }while( n>0 );
  //  }
  //  这里可能会出现负数
    return ret;
}

int main()
{
    printf("%d\n", f1(10));
    printf("%d\n", f2(10));
    printf("%d\n", f3(10));
}
```

do-while为何要存在

```c
int func(int n) {
    int i = 0;
    int ret = 0;

    //分配内存
    int* p = (int*)malloc(sizeof(int) * n);

    do
    {
```

```
        if (NULL == p) break;
        if (n < 0) break;
        for (i = 0; i < n; ++i) {
            p[i] = i;
            printf("%d\n", p[i]);
        }
        ret = 1;
    } while (0);

    //释放内存
    free(p);

    return ret;
}
```

对比

```
int func(int n) {
    int i = 0;
    int ret = 0;

    //分配内存
    int* p = (int*)malloc(sizeof(int) * n);

    //do
    //{
        if (NULL == p) return 0;
        if (n < 0) return 0;
        for (i = 0; i < n; ++i) {
            p[i] = i;
            printf("%d\n", p[i]);
        }
        ret = 1;
    //} while (0);

    free(p);
    return ret;
}

//如果 n<0条件就退出，那么p没有free有内存泄漏
//采用do-while内存调配只有一个入口一个出口，避免了内存泄漏
```

7.break,continue

> break 退出循环体
>
> continue推出当前循环

8.goto语句

避免使用!

9.void

没有返回值/参数用void修饰
不存在void变量
有void*指针，作为左值可接受任意类型的指针，作为右值赋值需要强制类型转换

```
#include <stdio.h>

void* m_memset(void* p, char v, int size)
{
    void* ret = p;          //转换成void*返回类型
    char* dest = (char*)p; //按字节赋值
    int i = 0;

    for (i = 0; i < size; i++)
    {
        dest[i] = v;
    }
    return ret;
}

int main() {

    int a[5] = { 1,2,3,4,5 };
    int i = 0;
    for (i = 0; i < 5; i++)
    {
        printf("%d\n", a[i]);
    }

    m_memset(a, 0, sizeof(a));

    for (i = 0; i < 5; i++)
    {
        printf("%d\n", a[i]);
    }
    return 0;
}
```

10.extern

> 声明外部定义的变量和函数
>
> 告诉编译器使用c方式编译（用在c++中），gcc编译不过，g++过
>
> ```
> extern "C"
> {
>     int f(int a,int b)
>     {
>         return a+b;
>     }
> }
> ```

11.sizeof关键字

> sizeof是编译器内置指示符，不是函数
>
> 用于计算所占内存大小
>
> sizeof的值在编译期已经确定了，不需要运行

```
int main() {
    int a;
    printf("%d\n", sizeof(a));
    printf("%d\n", sizeof a);
    printf("%d\n", sizeof(int));
}
```

12.const

> 修饰只读变量，本质还是变量
>
> const修饰的变量会在内存占用空间
>
> 本质const只对编译器有用，，运行时无用，仍然可用指针改变值

```
int main() {

    const int cc = 1;


    printf("%d\n", cc);

    //error
    //cc = 3;

    int* p = (int*)&cc;
    *p = 3;
    printf("%d\n", cc);
}
```

> const修饰的数组是只读的
>
> const修饰的数组空间不可改变

> const修饰指针
>
> 左数右指：const在*左边指针指向的数据为常量，const在 *右边指针本身为常量
>
> const修饰函数
>
> 多用于返回指针，表示返回值不可改变

13.volatile

> 编译器警告指示字

> 用于告诉编译器必须每次都去从内存中取值，避免优化c
>
> 主要用于多线程访问的变量
>
> 也可修饰可能被未知因数更改的变量

const volatile int i=0是否合理，参考https://www.cnblogs.com/melons/p/5791839.html

14.struct

> 空struct避免使用，可能为1或者报错
>
> 柔性数组：大小待定，struct最后的一个元素可以是大小未知的数组,大小为除了最后一个数组之外的大小

```
#include <stdio.h>

typedef struct _soft_array {
    int len;
    int array[]; //视作占位符
}SoftArray;

int main() {

    int i = 0;
    SoftArray s;
    SoftArray* sa = (SoftArray*)malloc(sizeof(SoftArray) + sizeof(int) *
10);
    sa->len = 10;
    for (i = 0; i < sa->len; ++i) {
        sa->array[i] = i + 1;
        printf("%d\n", sa->array[i]);
    }
    printf("%d\n", sizeof(s));
}
```

```
#include <stdio.h>
#include <malloc.h>
//柔性数组实现斐波那契数列
typedef struct _soft_array
{
    int len;
    int array[];
}SoftArray;

SoftArray* create_soft_array(int size)
{
    SoftArray* ret = NULL;

    if( size > 0 )
    {
        ret = (SoftArray*)malloc(sizeof(*ret) + sizeof(*(ret->array)) *
size);

        ret->len = size;
    }
```

```
    return ret;
}

void fac(SoftArray* sa)
{
    int i = 0;

    if( NULL != sa )
    {
        if( 1 == sa->len )
        {
           sa->array[0] = 1;
        }
        else
        {
            sa->array[0] = 1;
            sa->array[1] = 1;

            for(i=2; i<sa->len; i++)
            {
                sa->array[i] = sa->array[i-1] + sa->array[i-2];
            }
        }
    }
}

void delete_soft_array(SoftArray* sa)
{
    free(sa);
}

int main()
{
    int i = 0;
    SoftArray* sa = create_soft_array(10);

    fac(sa);

    for(i=0; i<sa->len; i++)
    {
        printf("%d\n", sa->array[i]);
    }

    delete_soft_array(sa);

    return 0;
}
```

14.union

> 共享空间，选最大的

15.enum

> 定义自定义类型
>
> 变量只能取定义时的离散值，默认常量在前一个值的基础上+1

enum　#define对比

> #define只是值替换，枚举常量是真正的常量
>
> #define宏常量无法被调试，枚举常量可
>
> #define宏常量没有类型信息，枚举常量是一种特定类型的常量

16.typedef

> typedef 用于给一个已经存在的数据类型重命名
>
> ```
> typedef char* PCHAR;
> #define PCHAR2 char*
>
> PCHAR p1, p2;
> PCHAR2 p3, p4; //p4是char型
> ```