函数

引入

函数是一组一起执行一个任务的语句。每个 C 程序都至少有一个函数，即主函数 main()，所有简单的程序都可以定义其他额外的函数。

函数结构

函数声明告诉编译器函数的名称、返回类型和参数。函数定义提供了函数的实际主体。

```
return_type function_name( parameter list )
{
   body of the function
}
```

例如

```
#include<stdio.h>
int add(int a,int b)
{
    return a+b;
}
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n",add(a,b));
    return 0;
}
```

调用函数

创建 C 函数时，会定义函数做什么，然后通过调用函数来完成已定义的任务。

当程序调用函数时，程序控制权会转移给被调用的函数。被调用的函数执行已定义的任务，当函数的返回语句被执行时，或到达函数的结束括号时，会把程序控制权交还给主程序。

调用函数时，传递所需参数，如果函数返回一个值，则可以存储返回值。

例如：

```c
#include<stdio.h>

int add(int a,int b);

int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n",add(a,b));
    return 0;
}

int add(int a,int b)
{
    return a+b;
}
```

函数传参

如果函数要使用参数，则必须声明接受参数值的变量。这些变量称为函数的形式参数。

形式参数就像函数内的其他局部变量，在进入函数时被创建，退出函数时被销毁。

当调用函数时，有两种向函数传递参数的方式：

调用类型	描述
传值调用	该方法把参数的实际值复制给函数的形式参数。在这种情况下，修改函数内的形式参数不会影响实际参数。
引用调用	通过指针传递方式，形参为指向实参地址的指针，当对形参的指向操作时，就相当于对实参本身进行的操作。

引用调用

```
#include<stdio.h>

void add(int a,int b,int &c);

int main()
{
    int a,b,c;
    scanf("%d%d",&a,&b);
    add(a,b,c);
    printf("%d\n",c);
    return 0;
}

void add(int a,int b,int &c)
{
    c=a+b;
}
```

局部变量与全局变量

局部变量（Local Variable）：定义在函数体内部的变量，作用域仅限于函数体内部。离开函数体就会无效。再调用就是出错。

全局变量（Global Variable）:所有的函数外部定义的变量，它的作用域是整个程序，也就是所有的源文件，包括.c 和.h 文件。

```c
#include<stdio.h>
int a=10;
void test()
{
    printf("%d\n",a);
}
int main()
{
    int a=20;
    test();
    printf("%d\n",a);
    return 0;
}
```

联合和枚举类型

结构体

引入

在实际问题中有时候我们需要其中的几种一起来修饰某个变量，例如一个学生的信息就需要学号（字符串），姓名（字符串），年龄（整形）等等，这些数据类型都不同但是他们又是表示一个整体，要存在联系，那么我们就需要一个新的数据类型——结构体，他就将不同类型的数据存放在一起，作为一个整体进行处理。

结构体的声明及调用

```
struct tag {
    member-list
    member-list
    member-list
    ...
} variable-list ;
```

例如：

```
#include<stdio.h>

struct student
{
    int age;
    char name[20];
};

int main()
{
    student stu;
    scanf("%d%s",&stu.age,&stu.name);
    printf("%d\n%s\n",stu.age,stu.name);
    return 0;
}
```

结构作为函数参数

把结构作为函数参数，传参方式与其他类型的变量类似。

```
student elder(student a,student b)
{
    student c;
    if(a.age>b.age)return a;
    else return b;
}
```

联合和枚举类型

联合

union，又称联合体或是共用体，是一个能在同一块存储空间中（但非同时）存储不同类型数据的数据类型。也就是说几种成员数据“共用”了同一块存储空间。联合体特别适用于那些彼此互斥，但却又有一定联系的内容。联合体的作用不光是节省存储空间那么简单，更重要的是为数据提供了一个统一的封装和外部访问的接口。C 语言编译器保证了 union 的共用体的长度等于最长的成员的长度

```c
#include<stdio.h>

union group {
    int    digit;
    double myfloat;
    char   letter;
};

int main()
{
    group valA;
    valA.digit   = 7;     // 将7存储到valA中，使用2 bytes
    valA.myfloat = 2.23; // 将7抹去，存储2.23，使用8 bytes
    valA.letter  = 'T';  // 将2.23抹去，存储字母T，使用1 bytes
    printf("%d\n",a);
    return 0;
}
```

枚举

枚举类型（enumerated type）是一种代表整数常量的数据类型。通过关键字 enum，可以创建一个新“类型”并指定它的值。枚举类型的语法与结构体的语法相类似。设计枚举类型的目的在于提高程序的可读性。

例如，可以使用枚举类型指代是或非，使用“1”代表是，使用“0”代表非。又如，可以使用枚举类型代表一个星期的七天。使用整形 1~7 代表从 Monday 到 Sunday 的七天。

值得注意的是，虽然枚举常量是 int 型的，但枚举常量可以宽松地限定为任何一种整数类型，只要该类型能保存这些枚举常量。例如，上述中星期的取值从 1 到 7，我们就可以使用 unsigned char 来表示。

```
// Definition
enum WEEKDAY {
      MON=1, TUE, WED, THU, FRI, SAT, SUN
};
// Declaration
enum DAY tomorrow;
enum DAY good_day, bad_day;
```

指针

什么是指针?

指针是一个变量，其值为另一个变量的地址，即，内存位置的直接地址，就像其他变量或常量一样。

指针变量声明

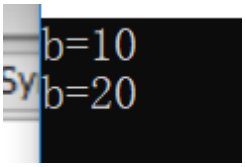
*：取地址符

&：取值符

```
type *var-name;
```

例如：

```c
#include<stdio.h>
int main()
{
    int b=10;
    int*a=&b;
    printf("b=%d\n",b);
    *a=20;
    printf("b=%d\n",b);
    return 0;
}
```



结构体指针的使用

使用指针调用结构体内部成员变量时，需要使用 -> 而非.

```c
struct student
{
    int age;
    char name[20];
};
int main()
{
    student a,*b=&a;
    a.age=18;
    printf("%d\n",b->age);
    return 0;
}
```

动态内存分配

首先，我们应该知道，所有的程序都必须留出足够的内存空间来存储所使用的数据，所以我们经常会预先给程序开辟好

内存空间，然后进行操作，但其实还有一种选择，能够让内存分配自动进行下去。

对于传统数组，会遇到这样的问题：

```
int a[10];
```

对这个数组我们在定义的时候必须给提前开辟好空间，并且在程序执行的过程中，这个开辟的内存空间是一直存在的，除非等到这个函数执行完毕，才会将空间释放。还有一个问题就是这个数组在程序中无法被修改。

malloc 和 free 函数

```
指针首地址=(类型*)malloc(空间大小)
```

```
int *a;
a=(int*)malloc(10*sizeof(int));

for(int i=0;i<10;i++)
*(a+i)=i;
for(int i=0;i<10;i++)
printf("a+%d=%d\n",i,*(a+i));
free(a);
```

```
a+0=0
a+1=1
a+2=2
a+3=3
a+4=4
a+5=5
a+6=6
a+7=7
a+8=8
a+9=9
```

new 和 delete 函数

数据类型*=new 数据类型;

```
1 int *x = new int;         //开辟一个存放整数的存储空间，返回一个指向该存储空间的地址(即指针)
2 int *a = new int(100);    //开辟一个存放整数的空间，并指定该整数的初值为100，返回一个指向该存储空间的地址
3 char *b = new char[10]; //开辟一个存放字符数组(包括10个元素)的空间，返回首元素的地址
```

1. int *a = new int;

   delete a; //释放单个int的空间

2.int *a = new int[5];

   delete [] a; //释放int数组空间