1 数据类型

1.1 基本数据类型

它们是算术类型

> - int 整型
> - char 字符型
> - float 浮点型
> - double 双精度浮点型

1.2 枚举类型

也是算术类型

> 只能赋予其一定的离散整数值的变量

1.3 void 类型

表明没有值的数据类型

1.4 派生类型

> - 数组类型
> - 指针类型
> - 结构体类型

各种类型的存储大小与系统位数有关，但目前通用的以64位系统为主。 以下列出了32位系统与64位系统的存储大小的差别（windows 相同）：

Windows vc12		Linux gcc-5.3.1		Compiler
win32	x64	i686	x86_64	Target
1		1	1	char
1		1	1	unsigned char
2		2	2	short
2		2	2	unsigned short
4		4	4	int
4		4	4	unsigned int
4		4	8	long
4		4	8	unsigned long
4		4	4	float
8		8	8	double
4		4	8	long int
8		8	8	long long
8		12	16	long double

1.5 强制类型转换

> - **隐式类型转换**：隐式类型转换是在表达式中自动发生的，无需进行任何明确的指令或函数调用。**它通常是将一种较小的类型自动转换为较大的类型，例如，将int类型转换为long类型或float类型转换为double类型。隐式类型转换也可能会导致数据精度丢失或数据截断。**
> - **显式类型转换：显式类型转换需要使用强制类型转换运算符（type casting operator），它可以将一个数据类型的值强制转换为另一种数据类型的值**。强制类型转换可以使程序员在必要时对数据类型进行更精确的控制，但也可能会导致数据丢失或截断。

```
/*隐式类型转换实例*/
int i = 10;
float f = 3.14;
double d = i + f; // 隐式将int类型转换为double类型

/*显式类型转换实例*/
double d = 3.14159;
int i = (int)d; // 显式将double类型转换为int类型
```

2 C常量

2.1 整数常量

> 整数常量可以是十进制、八进制或十六进制的常量。**前缀指定基数：0x 或 0X 表示十六进制，0 表示八进制，不带前缀则默认表示十进制。** 整数常量也可以带一个后缀，后缀是 U 和 L 的组合，U 表示无符号整数（unsigned），L 表示长整数（long）。后缀可以是大写，也可以是小写，U 和 L 的顺序任意。但是不能重复。

```
212         /* 合法的 */
215u        /* 合法的 */
0xFeeL      /* 合法的 */
078         /* 非法的：8 不是八进制的数字 */
032UU       /* 非法的：不能重复后缀 */

/*下面是各种类型的整数常量的实例*/
85          /* 十进制 */
0213        /* 八进制 */
0x4b        /* 十六进制 */
30          /* 整数 */
30u         /* 无符号整数 */
30l         /* 长整数 */
30ul        /* 无符号长整数 */
```

2.2 浮点常量

> - 浮点常量由整数部分、小数点、小数部分和指数部分组成。您可以使用小数形式或者指数形式来表示浮点常量。
> - 当使用小数形式表示时，必须包含整数部分、小数部分，或同时包含两者。当使用指数形式表示时， 必须包含小数点、指数，或同时包含两者。带符号的指数是用 e 或 E 引入的。

```
/*浮点常量的例子*/
3.14159       /* 合法的 */
314159E-5L    /* 合法的 */
510E          /* 非法的：不完整的指数 */
210f          /* 非法的：没有小数或指数 */
.e55          /* 非法的：缺少整数或分数 */
```

2.3 字符常量和字符串常量【不作详细展开】

2.4 定义常量

> 1. **使用 #define 预处理器： #define 可以在程序中定义一个常量，它在编译时会被替换为其对应的值。**
> 2. **使用 const 关键字：const 关键字用于声明一个只读变量，即该变量的值不能在程序运行时修改。**

```
/*把常量定义为大写字母形式，是一个很好的编程习惯*/

/*define 定义常量*/

#define 常量名 常量值
#define PI 3.14159

/*const 定义常量*/
const 数据类型 常量名 = 常量值;
/*下面的代码定义了一个名为MAX_VALUE的常量：*/
const int MAX_VALUE = 100;
```

2.4.1 #define 与 const 区别

#define 与 const 这两种方式都可以用来定义常量，选择哪种方式取决于具体的需求和编程习惯。通常情况下，建议使用 const 关键字来定义常量，因为它具有类型检查和作用域的优势，而 #define 仅进行简单的文本替换，可能会导致一些意外的问题。#define 预处理指令和 const 关键字在定义常量时有一些区别：

- 替换机制：**#define 是进行简单的文本替换，而 const 是声明一个具有类型的常量。**#define 定义的常量在编译时会被直接替换为其对应的值，而 const 定义的常量在程序运行时会分配内存，并且具有类型信息。
- 类型检查：**#define 不进行类型检查，因为它只是进行简单的文本替换。而 const 定义的常量具有类型信息，编译器可以对其进行类型检查。这可以帮助捕获一些潜在的类型错误。**
- 作用域：**#define 定义的常量没有作用域限制，它在定义之后的整个代码中都有效。而 const 定义的常量具有块级作用域，只在其定义所在的作用域内有效。**
- 调试和符号表：**使用 #define 定义的常量在符号表中不会有相应的条目，因为它只是进行文本替换。而使用 const 定义的常量会在符号表中有相应的条目，有助于调试和可读性。**

下面的表述可以更清晰地了解二者的区别

284

const 定义的是变量不是常量，只是这个变量的值不允许改变是常变量！带有类型。编译运行的时候起作用存在类型检查。

define 定义的是不带类型的常数，只进行简单的字符替换。在预编译的时候起作用，不存在类型检查。

1、两者的区别

(1) 编译器处理方式不同

- #define 宏是在预处理阶段展开。
- const 常量是编译运行阶段使用。

(2) 类型和安全检查不同

- #define 宏没有类型，不做任何类型检查，仅仅是展开。
- const 常量有具体的类型，在编译阶段会执行类型检查。

(3) 存储方式不同

- #define宏仅仅是展开，有多少地方使用，就展开多少次，不会分配内存。（宏定义不分配内存，变量定义分配内存。）
- const常量会在内存中分配(可以是堆中也可以是栈中)。

(4) const 可以节省空间，避免不必要的内存分配。 例如：

```
#define NUM 3.14159 //常量宏
const doulbe Num = 3.14159; //此时并未将Pi放入ROM中 ......
double i = Num; //此时为Pi分配内存，以后不再分配！
double I= NUM; //编译期间进行宏替换，分配内存
double j = Num; //没有内存分配
double J = NUM; //再进行宏替换，又一次分配内存！
```

const 定义常量从汇编的角度来看，只是给出了对应的内存地址，而不是象 #define 一样给出的是立即数，所以，const 定义的常量在程序运行过程中只有一份拷贝（因为是全局的只读变量，存在静态区），而 #define 定义的常量在内存中有若干个拷贝。

(5) 提高了效率。 编译器通常不为普通const常量分配存储空间，而是将它们保存在符号表中，这使得它成为一个编译期间的常量，没有了存储与读内存的操作，使得它的效率也很高。

(6) 宏替换只作替换，不做计算，不做表达式求解;

宏预编译时就替换了，程序运行时，并不分配内存。