

一、第一个C语言程序编写

1.1 QT的安装

1.2 C语言工程创建

1.3 第一个C语言程序

1.4 编译并运行代码

二、关键字

2.1 数据类型相关的关键字

2.2 存储相关关键字

2.3 控制语句相关的关键字

2.4 其他关键字

三、数据类型

3.1 基本类型

3.2 构造类型

3.3 常量和变量

3.3.1 常量

3.3.2 变量

3.3.3 格式化输出字符

3.4 类型转换

3.4.1 自动转换

3.4.2 强制转换

3.4.3 案例

四、运算符

4.1 运算符的概念以及分类

4.2 算术运算符

4.3 关系运算符

4.4 逻辑运算符

4.5 位运算符

4.6 条件运算符

4.7 逗号运算符

4.8 自增自减运算符

4.8.1 将++或者--放在变量的后面

4.8.2 将++或者--放在变量的前面

4.9 运算符优先级表

五、控制语句

5.1 选择控制语句

5.1.1 if语句

5.1.2 switch语句

5.2 循环控制语句

5.2.1 for循环

5.2.2 while循环

5.2.3 goto循环

一、第一个C语言程序编写

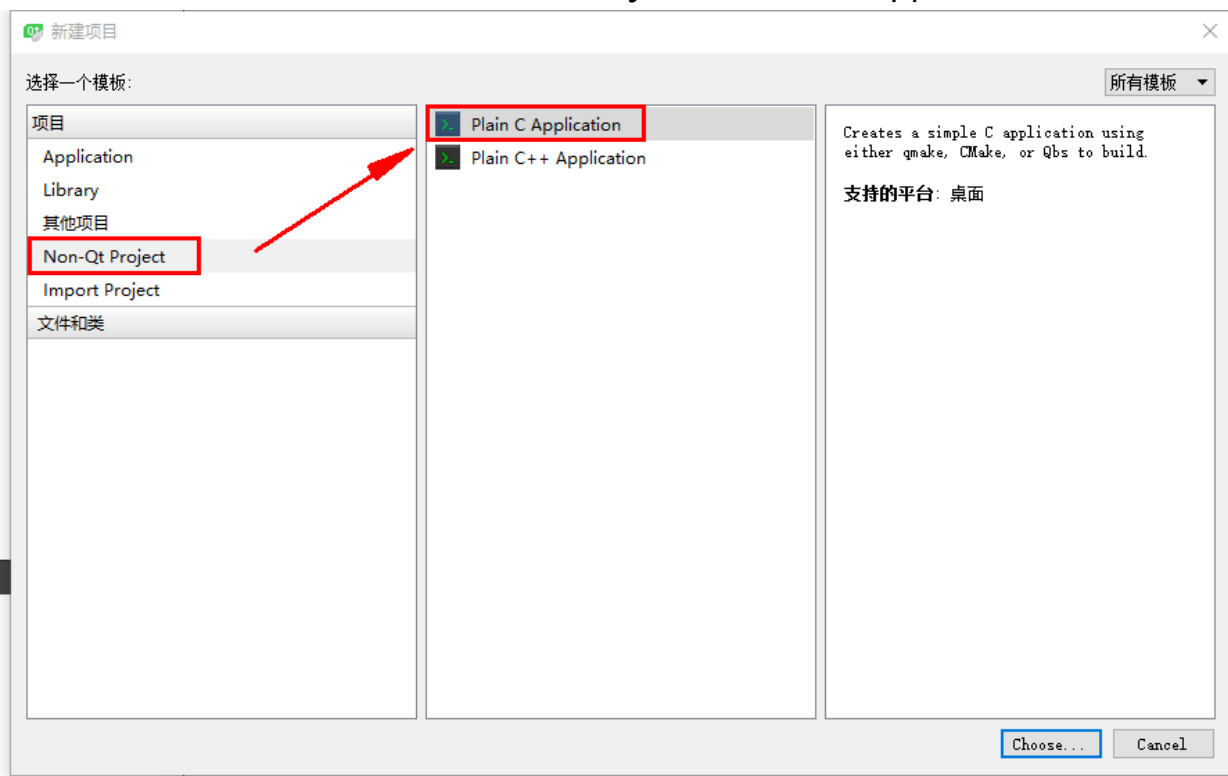
1.1 QT的安装

qt安装的时候要求安装路径必须是全英文的，其他下一步默认即可

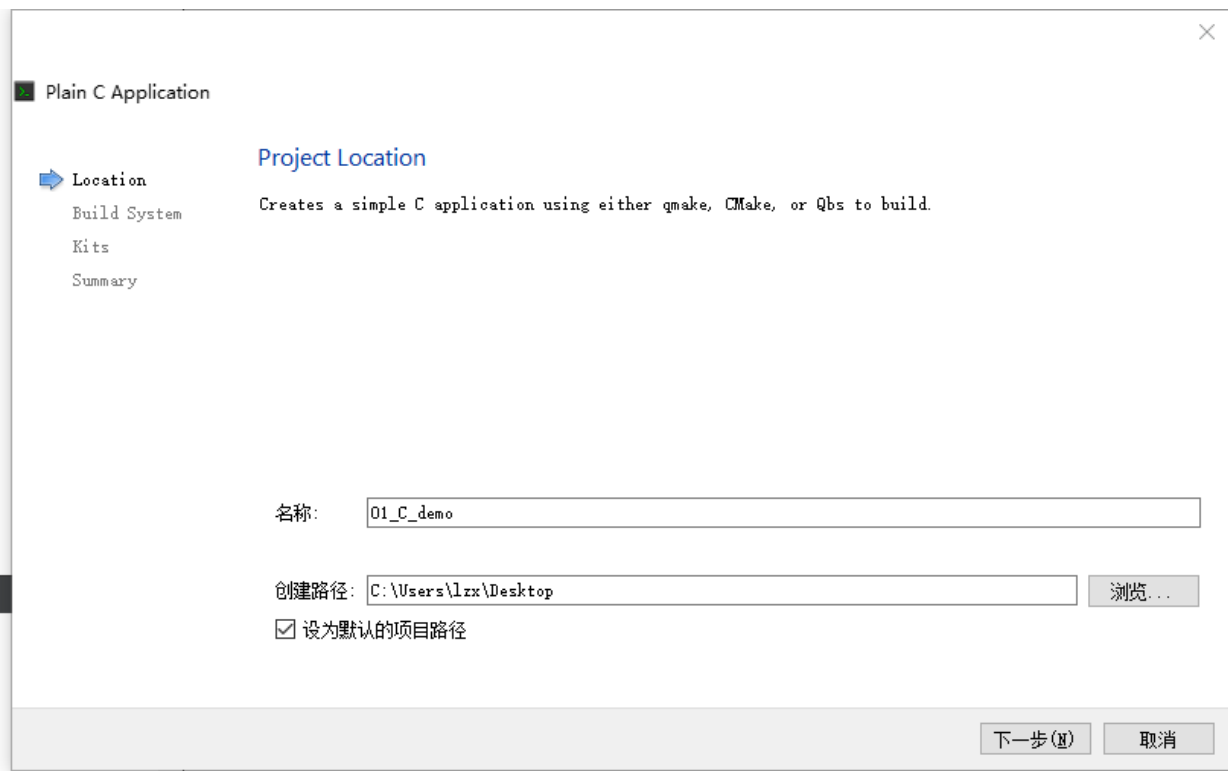
1.2 C语言工程创建

第一步：点击new project按钮，或者在左边“编辑”中，项目里面鼠标右键选择“新建项目”，或者点击左上角“文件”，选择新建文件或项目

第二步：在弹出的对话框中选择Non-Qt Project中的Plain C Application



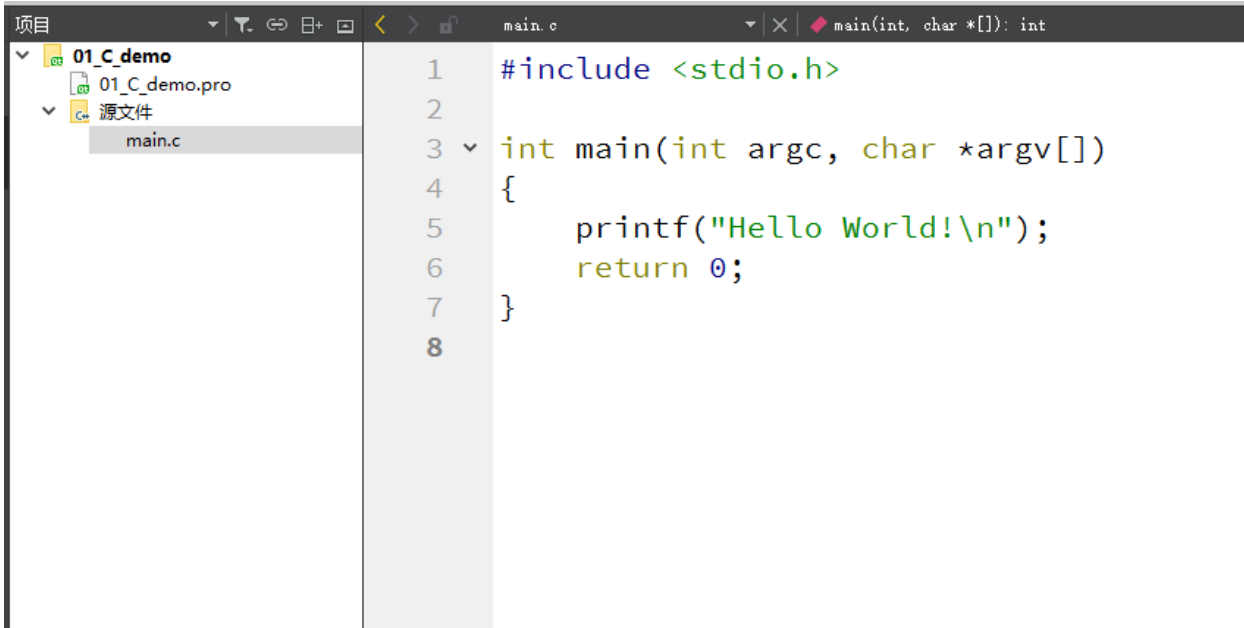
第三步：设置工程的名称和路径



注意：名称和路径都必须是英文的，并且名称只能使用数字、字母、下划线

第四步：接着一直下一步

第五步：工程创建好之后，会显示整个工程的目录结构以及基本代码



```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     printf("Hello World!\n");
6     return 0;
7 }
8
```

1.3 第一个C语言程序

```
1 #include <stdio.h> //头文件
2
3 //main: 主函数，任意一个C程序中必须有且只能有一个主函数，是程序的入口函数
4 int main()
5 {
6     printf("Hello World!\n"); //使用printf函数输出括号里面的字符串
7
8     return 0;
9 }
10
```

- 1.#include<stdio.h> 头文件包含，一定要有
- 2.每一个c语言的程序有且只有一个main函数，这是整个程序的开始位置
- 3.C语言中()、[]、{}、""、''、都必须成对出现,必须是英文符号
- 4.C语言中语句要以分号结束。
- 5.//为注释

1.4 编译并运行代码

点击



或者使用快捷键ctrl+r即可编译运行

```
应用程序输出
01_C_demo
Starting C:\Users\lzx\Desktop\build-01_C_demo-Desktop_Qt_5_8_0_MinGW_32bit-Debug\debug\01_C_demo.exe...
Hello World!
C:\Users\lzx\Desktop\build-01_C_demo-Desktop_Qt_5_8_0_MinGW_32bit-Debug\debug\01_C_demo.exe exited with code 0
```

二、关键字

关键字就是c语言已经定义好的名字，直接可以拿过来使用，不需要再次定义

2.1 数据类型相关的关键字

用于定义变量或者类型

定义变量的语法结构：

类型 变量名；

拓展：变量名属于标识符，标识符（变量名、函数名、重命名和取别名）有命名规则

标识符的命名规则：

标识符只能以数字、字母和下划线命名

首字母不能是数字

不能与关键字相同

char、short、int、long、float、double、
struct、union、enum、signed、unsigned、void

1、char 字符型，用char定义的变量是字符型变量，占1个字节

有符号： $-2^7 \sim 2^7-1$

无符号： $0 \sim 2^8-1$

2、short 短整型，使用short定义的变量是短整型变量，占2个字节

有符号： $-2^{15} \sim 2^{15}-1$

无符号： $0 \sim 2^{16}-1$

3、int 整型，用int定义的变量是整型变量，在32位以上的系统下占4个字节

有符号： $-2^{31} \sim 2^{31}-1$

无符号： $0 \sim 2^{32} - 1$

4、long 长整型 用long定义的变量是长整型的，在32位系统下占4个字节，在64位系统下占8个字节

5、float 单浮点型（实数），用float定义的变量是单浮点型的实数，占4个字节

6、double 双浮点型（实数），用double定义的变量是双浮点型的实数，占8个字节

7、struct 这个关键字是与结构体类型相关的关键字，可以用它来定义结构体类型，以后讲结构体的时候再讲

8、 union 这个关键字是与共用体（联合体）相关的关键字，以后再讲

9、 enum 与枚举类型相关的关键字 以后再讲

10、 signed 有符号(正负)的意思

在定义char、整型（short、int、long）数据的时候用signed修饰，代表咱们定义的数据是有符号的，可以保存正数，也可以保存负数

注意：默认情况下 signed 可以省略 即 int a=-10;//默认a就是有符号类型的数据

11、 unsigned 无符号的意思

在定义char、整型（short、int、long）数据的时候用unsigned修饰，代表咱们定义的数据是无符号类型的数据

无符号类型的变量只能保存正数和0。

12、 void 空类型的关键字

char、int、float 都可以定义变量

void不能定义变量，没有void类型的变量

void是用来修饰函数的参数或者返回值，代表函数没有参数或没有返回值

测试数据类型所占内存大小

```
1 #include <stdio.h>
2
3 //测试基本数据类型的所占内存大小
4
5 int main(int argc, char *argv[])
6 {
7     char a;
8     short b;
9     int c;
10    long d;
11    float e;
12    double f;
13
14    //sizeof: 是一个运算符，可以获取数据类型所占内存的大小
15    printf("%d\n", sizeof(a));
16    printf("%d\n", sizeof(b));
17    printf("%d\n", sizeof(c));
18    printf("%d\n", sizeof(d));
19    printf("%d\n", sizeof(e));
20    printf("%d\n", sizeof(f));
21
22    return 0;
```

执行结果

```

应用程序输出
02_data_type_size
Starting C:\Users\lzx\Desktop\Debug\debug\02_data_type_s
1
2
4
4
4
8
C:\Users\lzx\Desktop\build\02_data_type_size.exe exi

```

2.2 存储相关关键字

register、static、const、auto、extern

1、**register**是寄存器的意思，用register修饰的变量是寄存器变量，

即：在编译的时候告诉编译器这个变量是寄存器变量，**尽量**将其存储空间分配在寄存器中。

注意：

- (1)：定义的变量不一定真的存放在寄存器中。
- (2)：cpu取数据的时候去寄存器中拿数据比去内存中拿数据要快
- (3)：因为寄存器比较宝贵，所以不能定义寄存器数组
- (4)：register只能修饰 字符型及整型的，不能修饰浮点型
- (5)：因为register修饰的变量可能存放在寄存器中不存放在内存中，所以不能对寄存器变量取地址。因为只有存放在内存中的数据才有地址

2、**static** 是静态的意思

static可以修饰全局变量、局部变量、函数

使用static修饰的变量，此变量保存在内存的静态区空间中

3、**const**

const 是常量的意思

用const修饰的变量是只读的，不能修改它的值

```
const int a=101;//在定义a的时候用const修饰，并赋初值为101
```

从此以后，就不能再给a赋值了

```
a=111;//错误的
```

const可以修饰指针，这个在以后课程中重点讲解

4、auto

auto int a和int a是等价的，auto关键字现在基本不用

5、extern

是外部的意思，一般用于函数和全局变量的声明

2.3 控制语句相关的关键字

if、else、break、continue、for、while、do、switch case
goto、default

条件控制语句：

if语句：if else

switch语句：switch case default

循环控制语句：

for while do goto

辅助控制语句：

break continue

2.4 其他关键字

sizeof、typedef、volatile

1、sizeof

用来测变量、数组的占用存储空间的大小（字节数）

```
1 int a = 10;
2 int num;
3 num = sizeof(a);
4 printf("num = %d", num);
```

2、typedef 重命名相关的关键字

关键字，作用是给一个已有的类型，重新起个类型名，并没有创造一个新的类型

typedef 定义方式：

- 1、用想起名的类型定义一个变量

```
short int a;
```

- 2、用新的类型名替代变量名

```
short int INT16;
```

- 3、在最前面加typedef

```
typedef short int INT16;
```

- 4：就可以用新的类型名定义变量了

```
INT16 b 和 short int b 是一个效果
```

3、volatile 易改变的意思

用volatile定义的变量，是易改变的，即告诉cpu每次用volatile变量的时候，重新去内存中取

保证用的是最新的值,而不是寄存器中的备份。

volatile 关键字现在较少适用

三、数据类型

3.1 基本类型

char、short、int、long、float、double

```
1 #include <stdio.h>
2
3 //基本数据类型的学习和使用
4 //char short int long float double
5
6 int main(int argc, char *argv[])
7 {
8     //定义一个char类型的变量并赋值，输出字符使用%c
9     char a = 'w';
10    printf("a = %c\n", a);
11
12    //定义一个short类型的变量并赋值
13    short b = 100;
14    printf("b = %d\n", b);
15
16    //定义一个int类型的变量并赋值，输出int类型变量的值使用%d
17    int c = 9999;
```

```

18  printf("c = %d\n", c);
19
20  //定义一个long类型的变量并赋值，输出long类型变量的值使用%ld
21  long d = 34536453;
22  printf("d = %ld\n", d);
23
24  //定义一个float类型的变量并赋值，输出float类型变量的值使用%f
25  //默认保留小数点后六位，并且可以四舍五入
26  float e = 3.1415926;
27  printf("e = %f\n", e);
28
29  //定义一个double类型的变量并赋值，输出double类型变量的值使用%lf
30  double f = 3452.2345324523452;
31  printf("f = %lf\n", f);
32
33  return 0;
34 }

```

执行结果

```

Starting C:\Users\lzx\Desktop\build-03_base_data_type\Debug\debug\03_base_data_type.exe...

```

```

a = w
b = 100
c = 9999
d = 34536453
e = 3.141593
f = 3452.234532

```

```

C:\Users\lzx\Desktop\build-03_base_data_type\Debug\debug\03_base_data_type.exe exited with code 0

```

3.2 构造类型

概念：由若干个相同或不同类型数据构成的集合，这种数据类型被称为构造类型

例：int a[10];

数组、结构体、共用体、枚举

3.3 常量和变量

3.3.1 常量

常量：在程序运行过程中，其值不可以改变的量

例：100 'a' "hello"

常量的分类：

整型 100, 125, -100, 0

实型 3.14, 0.125f, -3.789

字符型 'a', 'b', '2'

字符串 "a", "ab", "1232"

ASCII 码表：对于计算机而言，只能识别二进制数，也就是数字，对于非数值型数据，如果要使用，就需要将其用一个数值型数据进行标识，就称之为ASCII码表

编码	字符	编码	字符	编码	字符	编码	字符
0	NUL	32	Space	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	TAB	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     //注意在使用字符类型的数据时，如果用%c输出就是输出字符，如果用%d就是输出字符的a
    scii值
6     char ch1 = 'w';

```

```
7 printf("ch1 = %c %d\n", ch1, ch1);
8
9 char ch2 = 97;
10 printf("ch2 = %c %d\n", ch2, ch2);
11
12 return 0;
13 }
```

执行结果

```
Starting C:\Users\lzx\Desktop\build-04_ascii.exe...
ch1 = w 119
ch2 = a 97
C:\Users\lzx\Desktop\build-04_ascii.exe exited with code 0
```

3.3.2 变量

变量：其值可以改变的量被称为变量

定义变量的方式：

存储类型 数据类型 变量名；

存储类型 数据类型 变量名 = 变量或者常量；

变量在定义的时候要满足标识符的命名规则

- (1) 只能由字母、数字和下划线组成
- (2) 首字母不能是数字
- (3) 不能关键字相同

整型数据

整型常量: (按进制分)：

十进制：以正常数字1-9开头，如457 789

八进制：以数字0开头，如0123

十六进制：以0x开头，如0x1e

整型变量:

有/无符号短整型(un/signed) short(int) 2个字节

有/无符号基本整型(un/signed) int 4个字节

有/无符号长整型(un/signed) long (int) 4个字节 (32位处理器)

实型数据(浮点型)

实型常量

实型常量也称为实数或者浮点数

十进制形式: 由数字和小数点组成:0.0、0.12、5.0

指数形式: 123e3代表 123×10^3 的三次方

123e-3代表 123×10^{-3} 的负三次方

不以f结尾的常量是double类型

以f结尾的常量(如3.14f)是float类型

实型变量

单精度(float)和双精度(double)3.1415926753456

float型: 占4字节, 7位有效数字,指数-37到38

double型: 占8字节, 16位有效数字,指数-307到308

字符数据

字符常量:

直接常量: 用单引号括起来, 如: 'a'、'b'、' 0' 等.

转义字符: 以反斜杠 "\ " 开头, 后跟一个或几个字符、如'\n','\t'等, 分别代表换行、横向跳格.

'\ ' 表示的是 \ '%%' '\ ' '

字符变量:

用char定义, 每个字符变量被分配一个字节的内存空间

字符值以ASCII码的形式存放在变量的内存单元中;

注: char a;

a = 'x';

a变量中存放的是字符'x'的ASCII :120

即a=120跟a='x'在本质上是一致的.

字符串常量

是由双引号括起来的字符序列, 如 "CHINA"、" 哈哈"

"C program" , "\$12.5" 等都是合法的字符串常量.

字符串常量与字符常量的不同

'a' 为字符常量, " a" 为字符串常量

每个字符串的结尾, 编译器会自动的添加一个结束标志位'\0',

即 "a" 包含两个字符 'a' 和 '\0'

3.3.3 格式化输出字符

3.3.3.1 普通字符

%d 十进制有符号整数

%ld 十进制long有符号整数

%u 十进制无符号整数

%o 以八进制表示的整数

%x 以十六进制表示的整数

%f float型浮点数

%lf double型浮点数

%e 指数形式的浮点数

%c 单个字符

%s 字符串

%p 指针的值

```
1 #include <stdio.h>
2
3 //格式化输出字符的使用
4
5 int main(int argc, char *argv[])
6 {
7     //输出整数
8     int a = 100;
9     //输出十进制数, 用%d
10    printf("a = %d\n", a);
11    //输出八进制数, 用%o
12    //printf("a = %o\n", a);
13    //使用%#o, 可以输出八进制数的前导符
14    printf("a = %#o\n", a);
15    //输出十六进制数
16    //printf("a = %x\n", a);
17    // 使用%#x, 可以输出十六进制数的前导符
18    printf("a = %#x\n", a);
19
20    //输出浮点型数据,float使用%f, double使用%lf
21    //默认小数点后保留六位, 并且可以四舍五入, 如果不够六位自动补0
22    float b = 3.1415926;
23    double c = 2345.2345;
24    printf("b = %f\n", b);
25    printf("c = %lf\n", c);
26
```

```

27 //输出字符，使用%c输出字符，使用%d可以输出字符的ascii码值
28 char d = 'y';
29 printf("d = %c %d\n", d, d);
30
31 //输出字符串，使用%s
32 //没有专门的变量保存字符串，一般使用数组来保存
33 char e[] = "hello world";
34 printf("e = %s\n", e);
35
36 //输出地址，使用%p
37 int f = 999;
38 //&: 取一个变量的地址，一般地址用十六进制数标识
39 printf("&f = %p\n", &f);
40
41 return 0;
42 }

```

执行结果

```

Starting C:\Users\lzx\Desktop\src\build-
\debug\05_printf.exe...

```

```

a = 100
a = 0144
a = 0x64
b = 3.141593
c = 2345.234500
d = y 121
e = hello world
&f = 0029FE8C

```

```

C:\Users\lzx\Desktop\src\build-
\05_printf.exe exited with code

```

3.3.3.2 特殊应用

`%3d` `%03d` `%-3d` `%5.2f`

`%3d` 要求宽度为3位，如果不足3位，前面空格补齐;如果足够3位，此语句无效

`%03d` 要求宽度为3位，如果不足3位，前面0补齐;如果足够3位，此语句无效

`%-3d` 要求宽度为3位，如果不足3位，后面空格补齐;如果足够3位，此语句无效

`%.2f` 小数点后只保留2位

```

1 int m = 456;
2 printf("%d%d\n", m, m);
3 //%5d: 输出的宽度为5，右对齐，如果实际数据的宽度小于5，则左边位置补空格，如果大
   于5，则没有用
4 printf("%5d%5d\n", m, m);

```



```

5 // %05d: 输出的宽度为5, 右对齐, 如果实际数据的宽度小于5, 则左边位置补0, 如果大于
5, 则没有用
6 printf("%05d%05d\n", m, m);
7 // %-5d: 输出的宽度为5, 左对齐, 如果实际数据的宽度小于5, 则右边补空格, 如果大于
5, 则没有用
8 printf("%-5d%-5d\n", m, m);
9
10 float n = 3.678;
11 printf("n = %f\n", n);
12 // %.2f: 小数点后保留两位并且可以四舍五入
13 printf("n = %.2f\n", n);

```

执行结果

```

456456
  456  456
0045600456
456  456
n = 3.678000
n = 3.68
C:\Users\lzx\Desktop\src\b

```

3.4 类型转换

数据有不同的类型, 不同类型数据之间进行混合运算时必然涉及到类型的转换问题.

转换的方法有两种:

自动转换:

遵循一定的规则, 由编译系统自动完成.

强制类型转换:

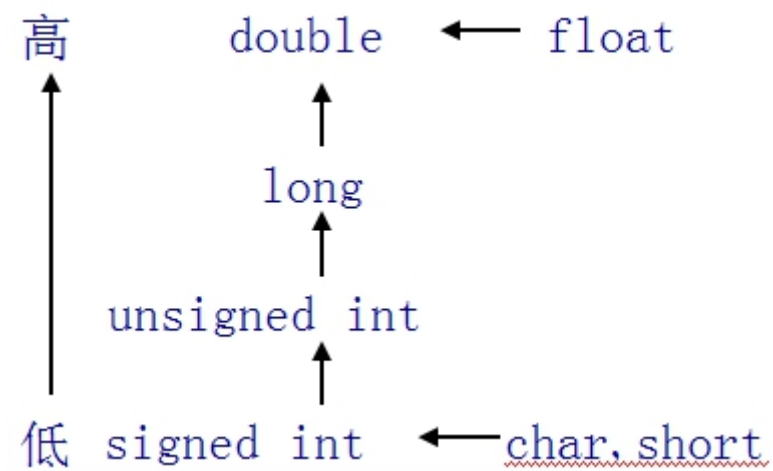
把表达式的运算结果强制转换成所需的数据类型

3.4.1 自动转换

自动转换原则

1、占用内存字节数少(值域小)的类型, 向占用内存字节数多(值域大)的类型转换, 以保证精度不降低.

2、转换方向:



3.4.2 强制转换

通过类型转换运算来实现

(类型说明符) (表达式)

功能：

把表达式的运算结果强制转换成类型说明符所表示的类型

例如：

`(float)a;` // 把a的值转换为实型

`(int)(x+y);` // 把x+y的结果值转换为整型

注意:

类型说明符必须加括号

3.4.3 案例

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     //*****强制类型转换之自动转换*****
6     //参加运算的成员全部变成int类型的参加运算，结果也是int类型的
7     printf("%d\n",5/2);
8
9     //当表达式中出现了带小数点的实数，参加运算的成员全部变成double类型
10    //参加运算，结果也是double型
11    printf("%lf\n",5.0/2);
12
13    //当表达式中有有符号数 也有无符号数，参加运算的成员变成无符号数参
14    //加运算结果也是无符号数。(表达式中无实数)
15    int a = -8;

```

```

16 unsigned int b=7;
17 if(a + b > 0)
18 {
19     printf("a+b>0\n");
20 }
21 else
22 {
23     printf("a+b<=0\n");
24 }
25
26 //在赋值语句中等号右边的类型自动转换为等号左边的类型
27 int m;
28 float n=5.8f;//5.8后面加f代表5.8是float类型，不加的话，认为是double类型
29 m = n;
30 printf("m = %d\n",m);
31 printf("n = %f\n", n); //注意自动类型转换都是在运算的过程中进行临时性的转换，并不会影响自动类型转换的变量的值和其类型
32
33 //*****强制类型转换之强制转换*****
34 int x = 10;
35 int y = 4;
36 float w;
37 w = (float)x / (float)y;
38 printf("w = %f\n", w);
39
40 return 0;
41 }

```

执行结果

```

Desktop_Qt_5_8_0_MinC
2
2.500000
a+b>0
m = 5
n = 5.800000
w = 2.500000
C:\Users\lzx\Desktop\

```

四、运算符

4.1 运算符的概念以及分类

运算符的概念

用运算符将运算对象(也称操作数)连接起来的、符合C语法规则的式子,称为C表达式
运算对象包括常量、变量、函数等

运算符的分类

1、双目运算符:即参加运算的操作数有两个

例: +

a+b

2、单目运算符:参加运算的操作数只有一个

++自增运算符 给变量值+1

--自减运算符

```
int a=10;
```

```
a++;
```

3、三目运算符:即参加运算的操作数有3个

()?:()

4.2 算术运算符

+ - * / % += -= *= /= %=

10%3 表达式的结果为1

复合运算符:

a += 3 相当于a=a+3

a*=6+8 相当于a=a*(6+8)

```
1 #include <stdio.h>
2
3 //算术运算符的使用
4
5 int main(int argc, char *argv[])
6 {
7     int a = 40;
8     int b = 6;
9     printf("%d + %d = %d\n", a, b, a + b);
10    printf("%d - %d = %d\n", a, b, a - b);
11    printf("%d * %d = %d\n", a, b, a * b);
12    printf("%d / %d = %d\n", a, b, a / b);
13    //printf如果要输出%, 则需要使用%%
14    printf("%d %% %d = %d\n", a, b, a % b);
```

```

15
16 float m = 10.32;
17 float n = 4.5;
18 printf("%.4f + %.4f = %.4f\n", m, n, m + n);
19 printf("%.4f - %.4f = %.4f\n", m, n, m - n);
20 printf("%.4f * %.4f = %.4f\n", m, n, m * n);
21 printf("%.4f / %.4f = %.4f\n", m, n, m / n);
22 //注意: 只有整数才能够取余, 浮点型数据不能取余
23 //printf("%.4f %% %.4f = %.4f\n", m, n, m % n);
24
25 return 0;
26 }

```

执行结果

Desktop_Qt_5_8_0_MinGW_32bit-Debug

40 + 6 = 46

40 - 6 = 34

40 * 6 = 240

40 / 6 = 6

40 % 6 = 4

10.3200 + 4.5000 = 14.8200

10.3200 - 4.5000 = 5.8200

10.3200 * 4.5000 = 46.4400

10.3200 / 4.5000 = 2.2933

C:\Users\lzx\Desktop\src\build-07_

4.3 关系运算符

>、<、==、>=、<=、!=

!=为不等于

一般用于判断条件是否满足或者循环语句

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5 //关系运算符连接的表达式的最终结果只有两个, 真和假
6 //一般返回的结果使用int保存, 如果为假则为0, 为真则为非0
7 //非0即为真
8 int a = 10 > 5;
9 int b = 10 < 5;
10 printf("a = %d, b = %d\n", a, b);
11

```

```
12 return 0;
13 }
```

执行结果

```
Starting C:\Users\lzx\
Desktop_Qt_5_8_0_MinG
a = 1, b = 0
C:\Users\lzx\Desktop\
```

注意：关系运算符中==用于判断左右两边是否相等，不能使用=，=用于赋值，将右值赋给左值

4.4 逻辑运算符

1、&& 逻辑与

两个条件都为真，则结果为真

```
if((a>b) && (a<c))
```

if(b<a<c)//这种表达方式是错误的

2、|| 逻辑或

两个条件至少有一个为真，则结果为真

```
if((a>b) || (a<c))
```

3、! 逻辑非

如果原本表达式为真，则为假，原本表达式为假，则为真

```
if(!(a>b))
```

```
{
}
```

注意：

在C语言不能这样写表达式 $10 < a < 100$ ，需要通过逻辑运算符

```
a > 10 && a < 100
```

数学中： $a < 10$ 或者 $a > 100$

C语言中： $a < 10 || a > 100$

逻辑与和逻辑或的短路原则：

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
```

```

4 {
5  int a = 20;
6  //逻辑与两边的表达式都为真，整体才为真，否则为假
7  int ret = a > 10 && a < 19;
8  printf("ret = %d\n", ret);
9
10 //逻辑或两边的表达式只要有一个为真，则整理为真，否则都为假才为假
11 ret = a > 10 || a < 19;
12 printf("ret = %d\n", ret);
13
14 //逻辑与的短路原则：如果第一个表达式的结果为假，则整体表达式为假，则后面所有的
    表达式都不会执行
15 int b = 100;
16 ret = (a < 19) && (b += 10);
17 printf("b = %d\n", b);
18
19 //逻辑或的短路原则：如果第一个表达式的结果为真，则整体表达式为真，所有后面所有的
    表达式都不会执行
20 ret = (a > 19) || (b += 10);
21 printf("b = %d\n", b);
22
23 return 0;
24 }

```

执行结果

```

starting C:\Users\
Desktop_Qt_5_8_0_M
ret = 0
ret = 1
b = 100
b = 100
C:\Users\lzx\Deskt
\debug\00_00000000

```

4.5 位运算符

1、&按位与

任何值与0得0，与1保持不变

使某位清0

0101 1011 &

1011 0100

0001 0000

2、| 按位或

任何值或1得1，或0保持不变

0101 0011 |

1011 0100

1111 0111

3、~ 按位取反

1变0，0变1

0101 1101 ~

1010 0010

4、^ 按位异或

相异得1，相同得0

1001 1100 ^

0101 1010

1100 0110

5、位移

>> 右移

<< 左移

注意右移分：逻辑右移、算数右移

(1)、右移

逻辑右移 高位补0，低位溢出

算数右移 高位补符号位，低位溢出（有符号数）

-15

1000 1111

1111 0000

1111 11 00 -4

A)、逻辑右移

低位溢出、高位补0

0101 1010 >>3

0000 1011

B)、算数右移：

对有符号数来说

低位溢出、高位补符号位。

1010 1101 >> 3

1111 010 1

0101 0011 >>3

0000 101 0

总结 右移：

1、逻辑右移 高位补0，低位溢出

注：无论是有符号数还是无符号数都是高位补0，低位溢出

2、算数右移 高位补符号位，低位溢出（有符号数）

注：对无符号数来说，高位补0，低位溢出

对有符号数来说，高位补符号位，低位溢出

在一个编译系统中到底是逻辑右移动，还是算数右移，取决于编译器

```
1 //判断右移是逻辑右移还是算数右移
2 #include <stdio.h>
3 int main(int argc, char *argv[])
4 {
5     printf("%d\n", -1>>3);
6     return 0;
7 }
8 //如果结果还是-1 证明是算数右移
```

(2)、左移<< 高位溢出，低位补0

5<<1

4.6 条件运算符

$()?():0$

如果？前边的表达式成立，整个表达式的值，是？和：之间的表达式的结果

否则是：之后的表达式的结果

$A?B:C;$

首先判断表达式A的结果，如果结果为真，则执行表达式B，如果表达式A为假，则执行表达式C

条件运算符其实就是一个简单的if else语句

if(A)

{

 B;

}

else

```
{  
    C;  
}
```

案例：

```
1 #include <stdio.h>  
2  
3 int main(int argc, char *argv[])  
4 {  
5     int a = 10, b = 20;  
6     int c;  
7     c = (a > b) ? (a += 10) : (b += 10);  
8     printf("c = %d\n", c);  
9     printf("a = %d, b = %d\n", a, b);  
10  
11     return 0;  
12 }
```

执行结果

```
Starting C:\Users\lzx\Desktop\src\build-11_operi  
Desktop_Qt_5_8_0_MinGW_32bit-Debug\debug\11_opei  
c = 30  
a = 10, b = 30  
C:\Users\lzx\Desktop\src\build-11_operator_term  
\debug\11_operator_term.exe exited with code 0
```

4.7 逗号运算符

(... , ... , ...)

使用逗号隔开的表达式从左向右依次执行，最后的表达式的值是整个运算的结果

例如：A = (B, C, D)

先运行表达式B，再运行表达式C，最后运行表达式D，最终变量A的值为表达式D的值

案例：

```
1 #include <stdio.h>  
2  
3 int main(int argc, char *argv[])  
4 {  
5     int a = 10, b = 20;  
6     int c;
```

```

7
8 //如果要使用逗号运算符，必须要加括号，如果不加，则会歧义
9 c = (a += 10, b += 10, a += b);
10 printf("a = %d, b = %d, c = %d\n", a, b, c);
11
12 return 0;
13 }

```

执行结果

```

Starting C:\Users\lzx\Desktop\src\build-12_operator_
Desktop_Qt_5_8_0_MinGW_32bit-Debug\debug\12_operator
a = 50, b = 30, c = 50
C:\Users\lzx\Desktop\src\build-12_operator_comma-Des
\debug\12_operator_comma.exe exited with code 0

```

4.8 自增自减运算符

++ --

4.8.1 将++或者--放在变量的后面

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5 //将++或者--放在变量的后面
6 //先使用，后自增或者自减
7 int a = 100;
8 int b;
9 b = a++;
10 printf("a = %d, b = %d\n", a, b);
11
12 return 0;
13 }

```

执行结果

```

Starting C:\Users\lzx\Desktop\src\bu
Desktop_Qt_5_8_0_MinGW_32bit-Debug\d
a = 101, b = 100
C:\Users\lzx\Desktop\src\build-13_op
\debug\13_operator_add_sub.exe exite

```

4.8.2 将++或者--放在变量的前面

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {

```

```

5 //将++或者--放在变量的前面
6 //先自增或者自减，后使用
7 int a = 100;
8 int b;
9 b = ++a;
10 printf("a = %d, b = %d\n", a, b);
11
12 return 0;
13 }

```

执行结果

```

Starting C:\Users\lx\Desktop\src\build-Desktop_Qt_5_8_0_MinGW_32bit-Debug\debug
a = 101, b = 101
C:\Users\lx\Desktop\src\build-13_operat\debug\13_operator_add_sub.exe exited wi

```

4.9 运算符优先级表

优先级	运算符	运算形式	结合方向	名称或含义
1	()	(e)	自左至右	圆括号
	[]	a[e]		数组下标
	.	x.y		成员运算符
	->	p->x		用指针访问成员的指向运算符
2	- +	-e	自右至左	负号和正号
	++ --	++x 或 x++		自增运算和自减运算
	!	!e		逻辑非
	~	~e		按位取反
	(t)	(t)e		类型转换
	*	*p		指针运算，由地址求内容
	&	&x		求变量的地址
sizeof	sizeof(t)	求某类型变量的长度		
3	* / %	e1 * e2	自左至右	乘、除和求余
4	+ -	e1 + e2	自左至右	加和减
5	<< >>	e1 << e2	自左至右	左移和右移
6	< <= > >=	e1 < e2	自左至右	关系运算(比较)
7	== !=	e1 == e2	自左至右	等于和不等比较
8	&	e1 & e2	自左至右	按位与
9	^	e1 ^ e2	自左至右	按位异或
10		e1 e2	自左至右	按位或
11	&&	e1 && e2	自左至右	逻辑与(并且)
12		e1 e2	自左至右	逻辑或(或者)
13	?:	e1 ? e2 : e3	自右至左	条件运算
14	=	x = e	自右至左	赋值运算
	+ = - = * =	x + = e		复合赋值运算
	/ = % = >> =			
	<< = & = ^ =			
15	,	e1, e2	自左至右	顺序求值运算

五、控制语句

5.1 选择控制语句

5.1.1 if语句

形式：

1) if(条件表达式)

```
{
    //复合语句，若干条语句的集合
    语句1;
    语句2 ;
}
```

2) if(条件表达式)

```
{
    语句块1
}
```

else

```
{
    语句块2
}
```

3) if(条件表达式1)

```
{
    语句块1
}
```

else if(条件表达式2)

```
{
    语句块2
}
```

else if(条件表达式3)

```
{
    语句块3
}
```

...

else

```
{
    语句块n
}
```

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     int n = 40;
6
```

```
7 //形式1: 只有if
8 //首先 判断if后面括号里面的表达式是否为真,
9 //如果为真, 则执行大括号里面的语句
10 //如果为假, 则不执行
11 // if(n >= 50)
12 // {
13 // printf("%d >= 50\n", n);
14 // }
15
16 //形式2: 有if和else
17 //先判断if后面的表达式, 如果为真, 则执行if后大括号里面的语句
18 //如果为假, 则执行else后面大括号里面的语句
19 //注意: 如果if和else后面只有一条语句, 则可以不加大括号
20 //但是如果有多条语句, 必须加大括号, 否则else找不到与之最近的if, 编译会报错
21 // if(n >= 50)
22 // {
23 // printf("%d >= 50\n", n);
24 // }
25 // else
26 // {
27 // printf("%d < 50\n", n);
28 // }
29
30 //形式3: if...else if...else
31 if(n > 50)
32 {
33     printf("%d > 50\n", n);
34 }
35 else if(n == 50)
36 {
37     printf("%d = 50\n", n);
38 }
39 else
40 {
41     printf("%d < 50\n", n);
42 }
43
44 return 0;
45 }
```

5.1.2 switch语句

```
switch ( 表达式 ) //表达式只能是字符型或整型的(short int  int  long int)
{
case 常量表达式1 :
    语句1 ;
    break ;
case 常量表达式2 :
    语句2;
    break ;
...
default :
    语句3 ;
    break;
}
```

运行顺序：将常量表达式的值与switch后面的表达式的值对比，如果表达式的值刚好等于case后面的某一个值，就会立即去执行case后的语句，如果都不是，则会执行default后面的语句

注意事项：

- (1) switch后面的表达式不能是浮点型，只能是整形的
- (2) 如果case后面的常量表达式与switch的表达式的值都不同，则执行default后面的语句
- (3) 每一个case执行结束后理论上必须跟一个break，作用就是跳出整个switch语句
- (4) case后面如果语句很多，不需要加大括号

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     int num = 2;
6
7     switch(num)
8     {
9         case 1:
10            printf("111111111111\n");
11            //如果不加break, 当要执行当前case语句时, 执行完毕后悔接着下一个
12            //case后的语句执行, 直到遇到break为止, 否则会一直执行
```

```
13 break;
14 case 2:
15 printf("222222222222\n");
16 break;
17 case 3:
18 printf("333333333333\n");
19 break;
20 default:
21 printf("hahahahaha\n");
22 break;
23 }
24
25 return 0;
26 }
```

5.2 循环控制语句

5.2.1 for循环

```
for(表达式1;表达式2;表达式3)
{
    //复合语句，循环体
    语句块
}
```

执行顺序：

先执行表达式1，然后执行表达式2，如果表达式2成立，则执行语句块
当语句块执行完毕之后，接着执行表达式3，然后再执行表达式2，
如果表达式2成立，则继续执行语句块，以此类推，直到表达式2不成立，循环结束

例如：

```
int i;
for(i = 1; i <= 10; i++)
{

}
```

案例：使用for循环求1到100的累加和

```
1 #include <stdio.h>
2
```



```

3 int main(int argc, char *argv[])
4 {
5     int i = 1;
6     int sum = 0;
7
8     for(i = 1; i <= 100; i++)
9     {
10        sum += i;
11    }
12
13    printf("1 + 2 + 3 + ... + 100 = %d\n", sum);
14
15    return 0;
16 }

```

执行结果

```

Starting C:\Users\lzx\Desktop\src\build-1
\debug\16_for.exe...
1 + 2 + 3 + ... + 100 = 5050
C:\Users\lzx\Desktop\src\build-16_for-Des
\16_for.exe exited with code 0

```

5.2.2 while循环

1)形式1：

```

while(条件表达式)
{
    //循环体，复合语句
    语句块
}

```

执行顺序：

首先判断while后面的条件表达式，如果表达式成立（为真），执行语句块，执行完语句块

接着再次执行条件表达式，如果表达式为真，则继续执行语句块，直到条件表达式为假，循环结束

案例：使用while循环求1到100的累加和

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {

```

```

5  int i = 1;
6  int sum = 0;
7
8  while(i <= 100)
9  {
10     sum += i;
11     i++;
12 }
13
14 printf("1 + 2 + 3 + ... + 100 = %d\n", sum);
15
16 return 0;
17 }

```

执行结果

Starting C:\Users\lzx\Desktop\src\build
 \debug\17_while.exe...

1 + 2 + 3 + ... + 100 = 5050

C:\Users\lzx\Desktop\src\build-17_while
 \17_while.exe exited with code 0

2) 形式2 : do...while...

```

do{
    //循环体
    语句块
}while(条件表达式);

```

执行顺序：

先执行do后面的语句块，然后判断while后面的条件表达式是否成立，

如果成立，则继续执行do后面的语句块，执行完毕后接着执行while后面的条件表达式，

当条件表达式不成立时，循环结束

注意：

不管条件是否成立，do后面的语句块都会执行一次，所以尽量少用do...while
 在while后面必须加一个分号

5.2.3 goto循环

goto主要用于在一个函数里面实现代码的跳转

```

1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {

```

```

5 //使用goto实现跳转
6 printf("1111111111111111\n");
7 goto NEXT;
8
9 printf("2222222222222222\n");
10
11 printf("3333333333333333\n");
12
13 NEXT:
14 printf("4444444444444444\n");
15 printf("hello world\n");
16
17 return 0;
18 }

```

执行结果

```

Starting C:\Users\lzx\Desktop\src
\debug\18_goto.exe...
1111111111111111
4444444444444444
hello world
C:\Users\lzx\Desktop\src\build-18

```

案例：使用goto实现求1到100的累加和

```

1 int i = 1;
2 int sum = 0;
3
4 JOOP:
5 sum += i;
6 i++;
7
8 if(i <= 100)
9 {
10 goto JOOP;
11 }
12 printf("1 + 2 + 3 + ... + 100 = %d\n", sum);

```

执行结果

```

1 + 2 + 3 + ... + 100 = 5050

```

注意：

在平时编写代码时，尽量少使用goto，因为会使得代码逻辑混乱，可读性差