Type Conversions and Operators

T. Howell
CS 49C

Type Conversions

- Floating point to integer
- Integer to floating point
- For an expression i + f, the integer is automatically converted to float, since float is "wider"
- Note: The header file `limits.h` contains the limits of the different types. Each compiler is allowed to set the limits. During compiler installation, your compiler may check your machine to set the limits in `limits.h`.

```
#include <stdio.h>

int x;
float f2;

int main() {
   f2 = 1.2345;
   x = (int)f2; // same as x = f2
   printf("The values are %f and %d\n", f2, x);
   f2 = (float)x; // same as f2 = x
   printf("Now f2 is %f\n", f2);
   return 0;
}
```

Type Conversions

- Character to integer
- Integer to character

```
#include <stdio.h>

int x;
char r;

int main() {
   r = 65; // integer to character
   x = 'r' + 2; // can perform arithmetic on character: 'r' has
                // the value 114
   printf("x is %d\n", x);
   r = x; // integer to character
   printf("r is %c\n", r);
   x = r + 2; // character to integer
   printf("x is %d\n", x);
   return 0;
}
```

Exercise

- Write a program to convert a character into the integer it represents, not its ASCII value.
- A numerical character, that is '0' through '9' should be converted to 0 through 9 respectively.
- A non-numerical character should be converted to 0.
- Assume that the character to be converted has been put in variable ***c***, and the converted integer should be written to variable ***i***.
- The ASCII value of the character '0' is 48. The ASCII value of the character '9' is 57.

Fill in code here

```
#include <stdio.h>

int i;
char c;

int main() {




}
```

Assignment Operator

- The '=' symbol is used as the assignment operator.

```
#include <stdio.h>

float f2;

int main() {
   f2 = 1.2345; // f2 is assigned the value 1.2345
                // in other words, f2 gets the value 1.2345
   f2 = f2 + 15.0;
   return 0;
}
```

Increment Operator

- For integers, we have an increment operator '++'

```
#include <stdio.h>

int p;

int main() {
   p = 10;
   printf("At point 0 p is %d\n", p);
   p = p + 1;
   printf("At point 1 p is %d\n", p);
   p++; // use the increment operator
   printf("At point 2 p is %d\n", p);
   return 0;
}
```

Prefix or postfix

- Increment operator can be put either before or after the variable.
- They both increment the value of the variable.

```
#include <stdio.h>

int p;

int main() {
   p = 10;
   printf("At point 0 p is %d\n", p);
   p = p + 1;
   printf("At point 1 p is %d\n", p);
   ++p; // use the increment operator
   printf("At point 2 p is %d\n", p);
   return 0;
}
```

Prefix and postfix: Difference

- Postfix changes the value of the variable after it is used.
- Prefix changes the value of the variable before it is used.

```
#include <stdio.h>

int i0, i1, i2, i3;

int main() {
   i0 = 20;
   i1 = ++i0; // use the prefix operator
   printf("The values are %d and %d\n", i0, i1);
   i2 = 20;
   i3 = i2++; // use the postfix operator
   printf("The values are %d and %d\n", i2, i3);
   return(0);
}
```

```
The values are 21 and 21
The values are 21 and 20
```

Bitwise Operators

- Integer constants expressed in decimal, octal and hexadecimal forms.

```
#include <stdio.h>

unsigned int i0, i1, i2;   //unsigned for logic
                           //signed for arithmetic

int main() {
   i0 = 20;
   i1 = 020;  // leading 0 means octal
   i2 = 0x20; // leading 0x means hexadecimal
   printf("The numbers are %u %u %u\n", i0, i1, i2);
   // %u in printf means unsigned decimal integer
   return(0);

}      The numbers are 20 16 32
```

Bitwise Operators

- Use bitwise AND (&) and OR (|) operators

```
#include <stdio.h>

unsigned int i0, i1, i2;

int main() {
   i0 = 0xbfa5;
   i1 = 63;
   i2 = i0 & i1;
   printf("The numbers are %u %u %u\n", i0, i1, i2);
   return 0;
}
```

```
The numbers are 49061 63 37
```

Bitwise Operators

- Use bitwise shift left (<<) and shift right (>>) operators

```
#include <stdio.h>

unsigned int i0, i1, i2;

int main() {
   i0 = 63;
   i1 = i0 << 2; // shift left by two bits
   printf("The numbers are %u %u\n", i0, i1);
   return 0;
}
```

```
The numbers are 63 252
```

Bitwise Operators

- Use bitwise not operator (~)

```
#include <stdio.h>

unsigned int i0, i1, i2;

int main() {
   i0 = 0xff00;
   i1 = ~i0; // bitwise NOT
   printf("The numbers are %u %u\n", i0, i1);
   printf("The numbers are %x %x\n", i0, i1);
   // %x in printf means unsigned hexadecimal integer
   return 0;
}
```

```
The numbers are 65280 and 4294902015
The numbers are ff00 ffff00ff
```

Precedence

- ( ) [ ]
- Unary + - ++ -- ! ~ (*type*)
- * / %
- Binary + -
- < <= > >= == !=
- &
- ^
- |
- = += *= /= %= &= ^= !=
- See Table 2-1 for more details