NWEN 241
More C Fundamentals

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui
CAPITAL CITY UNIVERSITY

This Lecture

- Data types
- Operators
- Data input/output
- Control constructs

Data Types

- Programming is about describing data and algorithms
- How data is represented in memory?

Data Types

- Programming is about describing data and algorithms
- How data is represented in memory?
- Four basic data types:
  - int (integer quantity)
  - char (single character)
  - float (floating-point number)
  - double (double-precision floating-point number)

  Note: There are also qualifiers associated with the types: short / long, and signed / unsigned.
- Data types for Java (any difference?)

Data Types

- Two groups of types
  - Integral types: int and char
    - Can be used to hold integer values
  - Floating types: float and double
    - Can be used to hold real values

Data Types

- Integral types

```
int i;
char c;
for (i = 65; i <= 90; i++)    /* 'A'=65, 'Z'=90 */
  printf("%c ", i);     /* what is i? */
                        /* what is printed? */

for (c = 'A'; c <= 'Z'; c++)  /* 'A'=65, 'Z'=90 */
  printf("%d ", c);     /* what is c? */
                        /* what is printed? */
```

Data Types

- Integral types

```
int i;
char c;
for (i = 65; i <= 90; i++)    /* 'A'=65, 'Z'=90 */
  printf("%c ", i);     /* print an int into a char */
                        /* A B C … Z is printed */

for (c = 'A'; c <= 'Z'; c++)  /* 'A'=65, 'Z'=90 */
  printf("%d ", c);     /* print a char into an int */
                        /* 65 66 67 … 90 is printed */
```

Data Types

- Floating types
  - How floating-point number represented in memory
  - 123.45 = 1111011.01110011 = 0.111101101110011 * 2^7
    - Mantissa: 111101101110011
    - Exponent: 7
    - Mantissa and exponent are stored separately
  - 123.75 = 1111011.11000000 = 0.111101111000000 * 2^7
  - 123.45 cannot be perfectly expressed in binary notation

Data Types

- Floating types
  - How floating-point number represented in memory
  - 123.45 = 1111011.01110011 = 0.111101101110011 * 2^7
    - Mantissa: 111101101110011
    - Exponent: 7
    - Mantissa and exponent are stored separately
  - 123.75 = 1111011.11000000 = 0.111101111000000 * 2^7
  - 123.45 cannot be perfectly expressed in binary notation
    - float t = 123.45
    - t = 123.449997
    - Use double

Data Types

- Sizes of different types
  - Use sizeof() to find out
  - The sizes may vary from machine to machine
  - The following rules are always guaranteed:
    - sizeof(char) = 1
    - sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)
    - sizeof(signed) = sizeof(unsigned) = sizeof(int)
    - sizeof(float) <= sizeof(double) <= sizeof(long double)
  - Does Java have varied sizes between systems?

Data Types

- Type casting
  - C does automatic type casting

```
int i = 2;
double d = 2.5;
i = (int)d;  /* explicit type casting */

i = d;
```

Data Types

- Type casting
  - C does automatic type casting

```
int i = 2;
double d = 2.5;
i = (int)d;  /* explicit type casting */

i = d;       /* d is converted to an int
              * and then assigned to i.
              */
```

  - Info losing type casting must be made explicitly in Java

Data Types

- Constants
  - integer constants
  - floating-point constants
  - character constants
  - string constants
  - enumeration constants (does Java have this?)
- Naming constants
  - Use the const qualifier (Java uses the **final** keyword)

```
const float pi = 3.14;  /* declares a “read-only” variable
  */
```

  - Use the preprocessor (Java does not have this)

```
#define PI 3.14  /* macro definition
                  * PI to be substituted by 3.14
                  */
```

Data Types

- Problems with micros

```
#define SQ(x) x * x
(int)SQ(r);       /* (int)r * r */
SQ(r1 + r2);      /* r1 + r2 * r1 + r2 */
```

Data Types

- Problems with micros

```
#define SQ(x) x * x
(int)SQ(r);       /* (int)r * r */
SQ(r1 + r2);      /* r1 + r2 * r1 + r2 */
```

  - Solution: `#define SQ(x) ((x) * (x))`

Data Types

- Problems with micros

```
#define SQ(x) x * x
(int)SQ(r);       /* (int)r * r */
SQ(r1 + r2);      /* r1 + r2 * r1 + r2 */
```

  - Solution: `#define SQ(x) ((x) * (x))`
  - Is it safe now?

Data Types

- Problems with micros

```
#define SQ(x) x * x
(int)SQ(r);      /* (int)r * r */
SQ(r1 + r2);     /* r1 + r2 * r1 + r2 */
```

  - Solution: `#define SQ(x) ((x) * (x))`
  - Is it safe now?

```
SQ(++r);    /* r is incremented twice? */
SQ(f());    /* f() called twice before the
             * multiplication
             */
```

  - Be careful when defining and calling macros

Data Types

- More data types later on ….

Built-in Operators

- arithmetic
- relational
- logical
- increment/decrement
- bitwise
- assignment
- others including type casting

Data Input and Output

- Functions for data input and output
  - getchar() / putchar()

```
char c;
c = getchar();    /* input a char */
putchar(c);       /* output a char */
```

  - gets() / puts()

```
char line[80];
gets(line);       /* input a line/string */
puts(line);       /* output a line */
```

  - **scanf() / printf()**

Data Input and Output

- scanf() / printf()

```
int i;
float f;
char c;
char s[80];
scanf("%d", &i);            /* %d is format information
                             * d is conversion character
                             */
scanf("%f", &f);            /* &f is f’s memory address
                             * input is sent to &f
                             */
```

Data Input and Output

- scanf() / printf()

```
int i;
float f;
char c;
char s[80];
scanf("%d", &i);            /* %d is format information
                             * d is conversion character
                             */
scanf("%f", &f);            /* &f is f’s memory address
                             * input is sent to &f
                             */
printf("\nYou typed in \"%f\"\n", f);
                            /* \n starts new line. \” treats “
                             * as an ordinary character
                             */
```

Data Input and Output

- scanf() / printf()

```
int i;
float f;
char c;
char s[80];
scanf("%d", &i);            /* %d is format information
                             * d is conversion character
                             */
scanf("%f", &f);            /* &f is f’s memory address
                             * input is sent to &f
                             */
printf("\nYou typed in \"%f\"\n", f);
                            /* \n starts new line. \” treats “
                             * as an ordinary character
                             */
scanf(" %c", &c);           /* blank space preceding %c to
                             * ignore \n typed in earlier
                             */
scanf("%s", s);             // a seq. of nonwhite space char
scanf("%[^\n]", s);         // [^\n] means \n is the end of
                            // input. s = &s[0]
```

Control Constructs

- Loops: for, while and do-while

```
#include <stdio.h>     /* each loop runs 4 times */

int main(void)
{ int i = 0, x =0;
  for (; i <4; i++)    /* starting and ending conditions */
  { x += i;
    printf("for loop: x = %d, i = %d\n", x, i);
  }

  while (i < 2*4)      /* only given ending condition */
  { x += i;
    printf("while loop: x = %d, i = %d\n", x, i);
    i++;
  }

  do                   /* do at least once */
  { x += i;
    printf("do-while loop: x = %d, i = %d\n", x, i);
    i++;
  } while (i < 3*4);   /* ending condition */
  return 0;
}
```

Control Constructs

- Blocks

```
int main(void)
{ int i = 0, x =0;

  for (int i=-4; i < 4; i++)  /* Only for C99. i is re-declared. */
  { x += i;
  }


  while (i < 2*4)
  { x += i;
    i++;
  }


  do
  { x += i;
    i++;
  } while (i < 3*4);

  return 0;
}
```

Control Constructs

- Blocks

```
int main(void)
{ int i = 0, x =0;            /* i will be used by the */
                              /* while and do-while loops, */
                              /* but not the for loop */
  for (int i=-4; i < 4; i++)  /* Only for C99. i is re-declared. */
  { x += i;                   /* only valid within this block. */
  }


  while (i < 2*4)             /* The 2nd i has no effects */
  { x += i;                   /* in this and next block */
    i++;
  }


  do
  { x += i;
    i++;
  } while (i < 3*4);

  return 0;
}
```

Control Constructs

- Conditionals: if-else and switch
  - Let us write a program to check if a character is an upper-case alphabetic letter

Control Constructs

- Conditionals: if-else and switch

```
int main(void)            /* to test if it is an upper-case alphabetic letter */
{ char i, c;
  printf("\nPlease enter an alphabetic character:\n");
  c = getchar();



}
```

Control Constructs

- Conditionals: if-else and switch

```
int main(void)            /* to test if it is an upper-case alphabetic letter */
{ char i, c;
  printf("\nPlease enter an alphabetic character:\n");
  c = getchar();

  if (isalpha(c))         /* true = nonzero, false = zero */
    ;                     /* empty is ok, but “;” must be there */
  else
    return(printf("You did not enter an alphabetic character\n"));









}
```

Control Constructs

- Conditionals: if-else and switch

```
int main(void)            /* to test if it is an upper-case alphabetic letter */
{ char i, c;
  printf("\nPlease enter an alphabetic character:\n");
  c = getchar();

  if (isalpha(c))         /* true = nonzero, false = zero */
    ;                     /* empty is ok, but “;” must be there */
  else
    return(printf("You did not enter an alphabetic character\n"));

  if (isupper(c) ? 1 : 0)                  /* true = 1, false = 0 */
    printf("if-else: it is an upper-case letter\n");
  else
    printf("if-else: it is a lower-case letter\n");





}
```

Control Constructs

- Conditionals: if-else and switch

```
int main(void)            /* to test if it is an upper-case alphabetic letter */
{ char i, c;
  printf("\nPlease enter an alphabetic character:\n");
  c = getchar();

  if (isalpha(c))         /* true = nonzero, false = zero */
    ;                     /* empty is ok, but “;” must be there */
  else
    return(printf("You did not enter an alphabetic character\n"));

  if (isupper(c) ? 1 : 0)                  /* true = 1, false = 0 */
    printf("if-else: it is an upper-case letter\n");
  else
    printf("if-else: it is a lower-case letter\n");

  i = (isupper(c) != 0 ? 'T' : 'F');        /* true = ‘T’, false = ‘F’ */
  switch(i) {
  case 'T':
    printf("switch: it is an upper-case letter\n");
    break;                /* break must be there, otherwise it will go through */
  case 'F':
    printf("switch: it is a lower-case letter\n");
  }
  return 0;
}
```

Control Constructs

- break, continue and goto
  - **break**: jumps out of the loop
  - **continue**: stops current iteration and starts next iteration
  - **goto** jumps to a labelled statement
  - Java support labelled **continue** and **break** statement
  - Java does not support **goto**

Control Constructs

- break, continue and goto
  - **break**: jumps out of the loop
  - **continue**: stops current iteration and starts next iteration
  - **goto** jumps to a labelled statement
  - Java support labelled **continue** and **break** statement
  - Java does not support **goto** (**goto is bad**)

Next Week

- Functions, pointers and arrays
- Next lecture: functions