Lecture04: Basic Types & Function

Slides modified from Yin Lou, Cornell CS2022: Introduction to C

1

Administrative things

- Assignment #3 is on the course webpage due next week.

2

Review from Last Week

- if, else-if, else
- < condition> ? <expression1> : <expression2>
- switch, case, default, break
- for (<exp1>; <exp2>; <exp3>) <statement>
- while (<condition>) <statement>
- do <statement> while (<condition>);
- break, continue

3

Basic Types

- Have already used two C basic types; int and double
- Learn the rest of the basic types
  - Integer type: short int, int, long int, long long int + signed, unsigned
  - Floating types: float, double, long double
  - Character types: char

4

Integer Type: short int

- Different integer types have different sizes
- For examples: size of (short int) is 2 bytes
- Declare variable with short int:
  - short int a; // short a;
- What is a byte? A byte = 8 bits. What is a bit?
- How to represent an integer 5?
- How to represent a positive/negative sign?
  - The sign bit: 0xxxxxx (positive or zero), 1xxxxxx (negative)
- What is the largest number for short int? 0111111
- What is the smallest number for short int? 00000000

5

Integer Type: unsigned short int

- An integer with no sign bit (zero or positive)
- Declare variable with unsigned short int:
  - unsigned short int a;
- What is the largest/smallest number for unsigned short int?

6

Integer Type:
short int, int, long int, long long int

- Size rule: short int < int <= long int <= long long int
- How do you find out these sizes? sizeof()

```
#include <stdio.h>
int main()
{
   printf("sizes: short int(%d), int(%d), long int(%d), long long int(%d)\n",
        sizeof(short int), sizeof(int), sizeof(long int), sizeof(long long int));
}
```

- On my machine:
  - short int: 2 bytes
  - int: 4 bytes
  - long int: 4 bytes
  - long long int: 8 bytes

7

Integer Type:
short int, int, long int, long long int

- Size rule: short int < int <= long int <= long long int
- How do you find out these sizes? sizeof()

```
#include <stdio.h>
int main()
{
   printf("sizes: short int(%d), int(%d), long int(%d), long long int(%d)\n",
        sizeof(short int), sizeof(int), sizeof(long int), sizeof(long long int));
}
```

- On my machine:
  - short int(2B): -32,768 ~ 32,767
  - int(4B): -2,147,483,648 ~ 2,147,483,648
  - long long int(8B): -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,808

8

Octal and Hexadecimal Numbers

• Decimal(base 10)	15	255	32767
• Octal(base 8)	017	0377	077777
• Hexadecimal(base 16)	0xf	0xff	0x7fff

9

Reading & Writing Integers

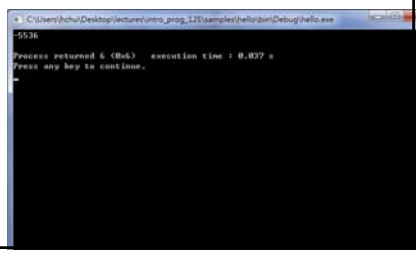
```
unsigned int u;

scanf("%u", &u);    // read u in base 10
printf("%u", u);    // write u in base 10
scanf("%o", &u);    // read u in base 8
printf("%o", u);    // write u in base 8
scanf("%x", &u);    // read u in base 16
printf("%x", u);    // write u in base 16

short int s;
scanf("%hd", &s);   // add h
printf("%hd", s);
```

```
long int l;
scanf("%ld", &l);    // add l
printf("%ld", l);

long long int ll;
scanf("%lld", &ll);   // add ll
printf("%lld", ll);
```

10

Common Pitfall: Integer Overflow

```
short i;

i = 10000 + 20000 + 30000;
printf("%d\n", i);
```



Floating Types

- float: 4B, one bit for sign, 8 bits for exponent, 23 bits for fraction
  - Range: 1.17549 * 10^-38 ~ 3.40282 * 10^38
  - 6 digits precision
- double: 8B
  - Range: 2.22507*10^-308 ~ 1.79769*10^308
  - 15 bits precision
- long double: 12B

12

Reading & Writing Floats

```
float f;
scanf("%f", &f);
printf("%f", f);

double d;
scanf("%lf", &d);          // add l
printf("%lf", d);

long double ld;
scanf("%Lf", &ld);         // add L
printf("%Lf", ld);
```

13

In-Class Exercise 3-1

Write a program that computes the factorial of a positive integer. Note that factorial function F(n) = n*(n-1)*(n-2)*…1

Enter a positive integer: 6
Factorial of 6: 720

(a) Use a short variable to store the value of the factorial. What is the largest value of n for which the program correctly prints the factorial of n?
(b) Repeat (a) using an int variable instead.
(c) Repeat (a) using a long variable instead.
(d) Repeat (a) using a long long variable instead.
(e) Repeat (a) using a float, double, long double variable instead.

14

Character Types

- char: 1B
  - 256 characters

```
char ch;
ch = 'a';
ch = 'A';
ch = '0';
ch = ' ';
```

15

C treats characters as small integers

```
char ch;
int i;

i = 'a';            // i is now 97
ch = 65;            // ch is now 'A'
ch = ch + 1;        // ch is now 'B'
ch++;               // ch is now 'C'

if ('a' <= ch && ch <= 'z')   // ch can also be compared like numbers
        ch = ch – 'a' + 'A';

for (ch = 'A'; ch <= 'Z'; ch++) …
```

16

Reading & Writing Characters

```
char ch;
scanf("%c", &ch);           // read a single character, including a space!
printf("%c", ch);

scanf(" %c", &ch);          // skip white spaces, then read ch

do {
        scanf("%c", &ch);
} while (ch != '\n');       // detec the end of line char

putchar(ch);                // write a char
ch = getchar();             // read a char and store it in ch

while ((ch = getchar()) == ' ');        // skip spaces
```

17

In-Class Exercise 3-2

Write a program that takes a first name and last name entered by the user and displays the last name, a comma, and the first initial, followed by a period:

Enter a first and last name: Lloyd Fosdick
Fosdick, L.

18

Type Conversion & Casting

```
char c;
short s;
int I;
unsigned int u
long int l;
usigned long int ul;
float f;
double d;
long double ld;

i = i + c; // c is converted to int
i = i + s; // s is converted to int
u = u + I; // l is converted to int
l = l + u; // u is converted to long int
```

```
i = (int) f;  // f is converted to int
f = (float) i; // I is converted to float

c = 10000; // wrong
s = 999999999999999; // wrong
```

19

Math Functions

- Many math functions are defined in <math.h>
  - pow(a, b) - Compute a^b
  - exp(a) - Compute e^a
  - log(a) - Compute natural logarithm
  - log10(a) - Compute common logarithm
  - sqrt(a) - Compute square root
  - fabs(a) - Compute absolute value
  - ceil/floor - Round up/down value
  - cos, sin, tan
  - acos, asin, atan

20

Functions

- Purpose of functions
  - **Break a program into pieces that are easier to write and understand**
    - Break down a **big** complex problem into **small** sub-problems that are easy to solve
    - What if the subproblems are still difficult to solve?
    - More about this in the next lecture.
  - Makes *recursive* algorithms easier to implement
  - Promotes code reuse
- Disadvantage of functions
  - Function calls add some memory and time overhead

21

A Simple Function

Compute base^{exp}

```
int power(int base, int exp)
{
        int i, p = 1;
        for (i = 1; i <= exp; ++i)
        {
                p *= base;  // p = p * base;
        }
        return p;
}
```

22

Simple Function in Context

```
#include <stdio.h>

int power(int base, int exp);                   // function prototype or declaration

void main()
{
    int i = 3, j = 4;
    printf("%d^%d is %d.\n", i, j, power(i, j));    // function call
}

int power(int base, int exp)                    // function definition
{
    int i, p = 1;
    for (i = 1; i <= exp; ++i)
    {
        p *= base;
    }
    return p;
}
```

23

Function Return Values

- If a function returns type void, then no return statement is needed.
- If a function returns another type, then a return statement is required along all possible execution paths.
- What's wrong with the left?

```
#include <stdio.h>

int foo(int arg)
{
        if (arg == 1)
        {
                return 1;
        }
}

void main()
{
        printf("%d\n", foo(0));
}
```

24

Call by Value

- Function arguments in C are passed by *value*
  - The *value* of the argument is passed, not a reference
  - Functions are given a new copy of their arguments
  - So a function can't modify the value of a variable in the calling function (unless you use **pointers**)
  - Pointers in the next lecture

```
#include <stdio.h>

int foo(int a)
{
   a = 3;
   return a;
}

void main()
{
   int a = 1, b;
   b = foo(a);
   printf("%d %d\n", a, b);  // Output 1 3
}
```

25

Call by Value: More example

```
#include <stdio.h>

void swap(int a, int b)
{
    int t = a;
    a = b;
    b = t;
}
void main()
{
    int a = 1, b = 2;
    swap(a, b);
    printf("%d %d\n", a, b); // Output 1 2
}
```

26

Call by Value: Tradeoff

- Call by value has advantages and disadvantages
- Advantage: some functions are easier to write

```
int power(int base, int exp)
{
        int result = 1;
        for (; exp >= 1; exp = exp - 1)
        {
                result *= base;
        }
        return result;
}
```

27

Call by Value: Tradeoff

- Disadvantage: sometimes you'd like to modify an argument (e.g. swap() function)
  - What's wrong with the left?
  - We'll see how to do this using pointers later

```
void swap (int x, int y)
{
        int temp = x;
        x = y;
        y = temp;
}
```

28

In-Class Exercise 3-3

Write a function that prints the following diamond shape. Your program first reads an odd number in the range 1 to 19 to specify the number of rows in the diamond and a character. You then call a function void printline(int ns, int nc, char c) that print one line with ns number of spaces, nc number of characters, and then ns number of spaces.

```
Input the number of rows and the symbol char in the diamond> 5 *
  *
 ***
*****
 ***
  *
```

29

Recursion

```
int fact(int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * fact(n - 1);
    }
}
```

```
// non-recursive version

int fact(int n)
{
   int f = 1;
   int i;
    for (i=n; i>1; i--)
    {
     f = f*i;
    }
    return f;
}
```

30

In-Class Exercise 3-4

Write a **recursive** function to compute the Fibonacci number, namely, besides main function, you must implement another Fibonacci function. Note that the Fibonacci number sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21. Your program should have the following input & output.

```
Enter an integer> 0
Fib(0) = 0

Enter an integer> 6
Fib(6) = 8
```

31

Declaration vs. Definition

- Declaration
  - A declaration announces the properties of a variable (primarily its type).
  - Example:

    ```
    extern int n;
    extern double val[];
    ```
- Definition
  - A definition also causes storage to be set aside.
  - Example:

    ```
    int n;
    double val[MAX LEN];
    ```

32

Manage Your Project

- It's always recommended to modularize your project. How?
- Write functions and paste them in new file?
- Definitions and declarations are shared among many source files. How to centralize this, so that there is only one copy to get and keep right as the program evolves?
- We could use header files.

33

Header File

- Place common material in a header file.
- Can be *included* as needed.

Example: mymath.h

```
int fact(int n);
int power(int power, int exp);
```

34

Example

power.c

```
#include "mymath.h"

int power(int base, int exp)
{
    int result = 1;
    int i;
    for (i = 1; i <= exp; ++i)
    {
        result *= base;
    }
    return result;
}
```

fact.c

```
#include "mymath.h"

int fact(int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * fact(n - 1);
    }
}
```

35

Example

main.c

```
#include <stdio.h>
#include "mymath.h"

void main()
{
    printf("%d\n", power(5, 3));
    printf("%d\n", fact(5));
}
```

36

How to compile in CBs?

- TA will talk about this

37