ECE 551D Recitation
Reading Code, Types, Writing code and Compiling

Yang Deng, Daniel Park, Cecilé Sadler, Mutian Wang, Yang Zhong

August 28, 2020

Outline

Outline

Emacs

Great Reference Card

`https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf`

Files and buffers

Open file

`C-x C-f`

Switch buffer

`C-x b`

Leaving Emacs

Save file

`C-x C-s`
(do compulsively)

Close Emacs

`C-x C-c`

Suspend Emacs

`C-z`
(from Bash, do `fg` to return to Emacs)

Git basics

Turning in assignments

- `git add`
- `git commit -m ‘‘Here is my meaningful commit message describing the changes I have made.’’`
- `git push`

Outline

Reading function calls

```
int min(int a, int b) {
  if (a < b) {
    return a;
  }
  return b;
}
int max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

```
int euclid(int a, int b) {
  printf("euclid(%d, %d)\n", a, b);
  int larger = max(a, b);
  int smaller = min(a, b);
  if (smaller == 0) {
    return larger;
  }
  return euclid(smaller, larger % smaller);
}
int main(void) {
  int x = euclid(9135, 426);
  printf("x = %d\n", x);
  return EXIT_SUCCESS;
}
```

Scope practice

```
int main(void) {
  int a = 0;
  {
    printf("a = %d\n", a);
    int a = 1;
    printf("a = %d\n", a);
  }
  printf("a = %d\n", a);
  return 0;
}
```

Scope practice

```
int main(void) {
  int a = 0;
  {
    printf("a = %d\n", a);
    int a = 1;
    printf("a = %d\n", a);
  }
  printf("a = %d\n", a);
  return 0;
}
```

```
a = 0
a = 1
a = 0
```

Outline

Basic data types in C

char

size: 1 byte (8 bits)
interpretation: one ASCII character, example: 'f'

Basic data types in C

char

size: 1 byte (8 bits)
interpretation: one ASCII character, example: 'f'

int

size: 4 bytes (32 bits)
interpretation: integer, example: 42

Basic data types in C

char

size: 1 byte (8 bits)
interpretation: one ASCII character, example: 'f'

int

size: 4 bytes (32 bits)
interpretation: integer, example: 42

float

size: 4 bytes (32 bits)
interpretation: floating point number, example: 3.141592

Basic data types in C

char

size: 1 byte (8 bits)
interpretation: one ASCII character, example: 'f'

int

size: 4 bytes (32 bits)
interpretation: integer, example: 42

float

size: 4 bytes (32 bits)
interpretation: floating point number, example: 3.141592

double

size: 8 bytes (64 bits)
interpretation: floating point number, example: 3.141592653589793

Signed and unsigned

For a short int (assume 16 bits)

What is the smallest number that it can represent?
What is the largest number that it can represent?

Signed and unsigned

For a short int (assume 16 bits)

What is the smallest number that it can represent?
What is the largest number that it can represent?

For an *unsigned* short int (assume 16 bits)

What is the smallest number that it can represent?
What is the largest number that it can represent?

Signed and unsigned

For a short int (assume 16 bits)

What is the smallest number that it can represent?
What is the largest number that it can represent?

For an *unsigned* short int (assume 16 bits)

What is the smallest number that it can represent?
What is the largest number that it can represent?

What is the bit pattern for 41362 stored as an unsigned short?
What number is this interpreted as a signed short?

Signed and unsigned

For a short int (assume 16 bits)

What is the smallest number that it can represent?
What is the largest number that it can represent?

For an *unsigned* short int (assume 16 bits)

What is the smallest number that it can represent?
What is the largest number that it can represent?

What is the bit pattern for 41362 stored as an unsigned short?
What number is this interpreted as a signed short?

1010000110010010
interpreted as a signed short: -24174 (0101111001101101+1=0101111001101110)

Why two's complement?

Why two's complement?

So the operations for addition and subtraction can be unified!

Why two's complement?

So the operations for addition and subtraction can be unified!

Example 1

$69 + 12$

Why two's complement?

So the operations for addition and subtraction can be unified!

Example 1

$69 + 12$

Example 2

69 - 12
(can take negative of 12 and add!)

Floating-point numbers

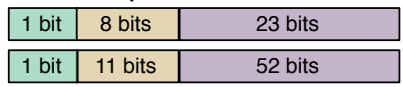
Sample Code

```
int main (void) {
    float p1 = 3.141592;
    double p2 = 3.141592653589793;
    ...
}
```

Conceptual Representation

p1	3.141592
p2	3.141592653589793

Hardware Representation $-1^{s} \times m \times 2^{e}$



Floating-point numbers

Sample Code

```
int main (void) {
    float p1 = 3.141592;
    double p2 = 3.141592653589793;
    ...
}
```

Conceptual Representation

p1	3.141592
p2	3.141592653589793

Hardware Representation $-1^{s} \times m \times 2^{e}$

1 bit	8 bits	23 bits
1 bit	11 bits	52 bits
sign	exponent	mantissa

float fRoot = sqrt(2.0);
float fSquared=fRoot*fRoot;
You can find the values: fRoot=1.4142135381698608, fSquared=1.9999998807907104
double dRoot = sqrt(2.0);
double dSquared=dRoot*dRoot;
You can find the values: dRoot=1.4142135623730951, dSquared=2.0000000000000004

Type conversion

Assume that we have executed `int a = 4;` and `int b = 5;`. What are the type and value of each of the following expressions:

1. `a / b`
2. `(double)(a / b)`
3. `a / (double)b`
4. `(double)a / b`
5. `a - b / 2`
6. `a- b / 2.0`

Type conversion

Four ways to convert type

1. Sign extend (smaller signed int to longer signed int)
2. Zero extend (smaller unsigned to longer unsigned)
3. Truncate (longer int to shorter int)
4. Fully calculate (int to real)

typedef

Suppose you are writing software in which you need a unique sequence numbers, and you decide that unsigned long is sufficiently large as a type to work with them.

How could you give this type a name (e.g., `seq_t`) so that you can use that name throughout your program? Write the C statement which would accomplish this goal.

Outline

Writing Code

7 Steps!

- 1 Work an Example Yourself
- 2 Write Down What You Just Did
- 3 Generalize Your Steps
- 4 Test Your Algorithm
- **5 Translate Your Algorithm to Code**
- 6 Test
- 7 Debug

Translate to Code

Repetition

```
for (int i = 0; i < max; ++i) {
  // do something
}

while () {}
```

Translate to Code

Decision Making

```
if () {}
else {}

switch(expression) {
  case constant-expression-1:
    ...;
  case constant-expression-2:
    ...;
  default:
    ...;
}
```

Outline

Writing Code Practice

Write a function `myRound` which takes a `double d`, and returns an integer. This function should round `d` to the nearest integer and return the rounded result.

(Hint: to get the fractional portion of `d` (the part after the decimal), think about how do you get the integral portion (the part before the decimal) using what you learned in Chapter 3——then think about what mathematical operation you can use to compute the fractional portion from the information you have).

Writing Code Practice

```
int myRound(double d) {
  int int_part = (int)d;
  double frac_part = d - int_part;
  if (frac_part >= 0.5) {
    return int_part + 1;
  }
  return int_part;
}
```

Outline

7 Steps!

- 1 Work an Example Yourself
- 2 Write Down What You Just Did
- 3 Generalize Your Steps
- 4 Test Your Algorithm
- 5 Translate Your Algorithm to Code
- **6 Test**
- **7 Debug**

Compiling

How to make your .c file into an executable binary file?

Suppose I have a myCode.c file, what command should I type so that I could have an executable binary file named myCode?

Compiling

How to make your .c file into an executable binary file?

Suppose I have a myCode.c file, what command should I type so that I could have an executable binary file named myCode?

```
gcc -std=gnu99 -pedantic -Wall -Werror -o my-code my-code.c
```

Compiling

```
gcc -std=gnu99 -pedantic -Wall -Werror -o my-code my-code.c
```

-std=gnu99: This tells the compiler should use the C99 standard with GNU extensions.

-pedantic: This tells the compiler to be adhere strictly to the standard, rejecting any code which is not compliant.

-Wall: This tells the compiler to issue warnings for a wide range of questionable behavior.

-Werror: This tells the compiler to treat all warnings as errors–make it refuse to compile until programmer fix all the warnings.

-o: Change the output name (default is a.out), and specify the output file name after it.

Dealing with Compiling Errors

tip 1

Compiler can get confused by earlier errors. If later errors are confusing, fix the first error and try to recompile before you attempt to fix them.

Dealing with Compiling Errors

tip 1

Compiler can get confused by earlier errors. If later errors are confusing, fix the first error and try to recompile before you attempt to fix them.

tip 2

If parts of an error message are completely unfamiliar, try to ignore them and see if the rest of the error messages make sense. If not, search for the confusing parts on Google.

Dealing with Compiling Errors

tip 1

Compiler can get confused by earlier errors. If later errors are confusing, fix the first error and try to recompile before you attempt to fix them.

tip 2

If parts of an error message are completely unfamiliar, try to ignore them and see if the rest of the error messages make sense. If not, search for the confusing parts on Google.

tip 3

Programmer's editors are good at helping you find mismatched braces and parentheses.

Dealing with Compiling Errors

tip 1

Compiler can get confused by earlier errors. If later errors are confusing, fix the first error and try to recompile before you attempt to fix them.

tip 2

If parts of an error message are completely unfamiliar, try to ignore them and see if the rest of the error messages make sense. If not, search for the confusing parts on Google.

tip 3

Programmer's editors are good at helping you find mismatched braces and parentheses.

tip 4

Be confident in your fix for an error. If you do not understand what is wrong and how to fix it, find out and be sure rather than randomly changing things.