

# Day05 A

Young W. Lim

2017-10-07 Sat

- 1 Based on
- 2 Structured Programming (2) Conditions and Loops
  - Conditional Statements
  - Loop Statements
  - Type Cast

## "C How to Program", Paul Deitel and Harvey Deitel

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

# The if selection statement

- a decision can be based on any expression
  - if the expression evaluates to zero : false condition
  - if the expression evaluates to non-zero : true condition
- a single entry / single exit structure

# The `if ... else` selection statement

- a double selection statement
- a single entry / double exit structure
- the *conditional operator* ( `A ? B : C` )
  - A: conditional expression
  - B: if the condition A is true, then B is executed
  - C: if the condition A is false, then C is executed

# The nested if ... else selection statements

- the values in a conditional expression can also be actions to be executed
- several statements in the body of if or else statement, must be enclosed by braces  
: a compound statement / block { ... }
- Nested if ... else statement for a multiple case selection

```
if (A) { S1; }  
  else if (B) { S2; }  
    else if (C) { S3; }  
      else { S4; }
```

# The while repetition statement

- specifies that an action is to be repeated as long as a condition is true
  - eventually the condition will become false the repetition terminates
- 1 counter controlled repetition
  - 2 sentinel controlled repetition (guard or sentry)

# The counter controlled repetition

- a *counter variable*  
to specify the number of times  
a set of statements should execute
- *definite* repetition : the number is know in advance
- a *counter variable* initialized with zero or one  
(counting from zero or one)  
the conditions must be different
- an *accumulation variable* for a total  
must be initialized with zero
- an uninitialized variable contains a garbage value
- must write before read



# The sentinel controlled repetition

- indefinite repetition  
the number of repetition is not known in advance
- a sentinel / dummy / flag value  
the end of data entry : the end of repetition
- a sentinel value must be carefully chosen  
must be different from the normal input data

```
while (grade != -1)                                // 0 <= grade <= 100
    S = S + grade;                                // negative grade : not a data
    i = i + 1;                                    // sentinel value
}
```

# The float type

- to represent the numbers with decimal point (floating point numbers)
- *precision* : the number of digits below the decimal point
- format / conversion specifier %f  
the default precision : 6 digits below the decimal point
- format / conversion specifier %.2f  
the precision of value 2 : 2 digits below the decimal point
- %f and %.2f use rounding off  
in order to print 6 and 2 digits below the decimal point

# Type Cast - Explicit

- to produce a floating point calculation with integer values must *cast* (*convert*) the integers into floating point numbers
- the unary cast operator (float)
- explicit conversion : cast operators

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

```
int main(void) {
```

```
    int    a;
```

```
    float  x;
```

```
    printf("2 /3  = %d\n", 2 /3 );
```

```
    printf("2 /3. = %f\n", 2 /3.);
```

```
    printf("2./3  = %f\n", 2./3 );
```

```
    printf("2./3. = %f\n", 2./3.);
```

```
    a = 2.12345;
```

```
    x = 3;
```

```
    printf("a = %d\n", a);
```

```
    printf("x = %f\n", x);
```

```
}
```

```
2 /3  = 0
```

```
2 /3. = 0.666667
```

```
2./3  = 0.666667
```

```
2./3. = 0.666667
```

```
a = 2
```

```
x = 3.000000
```

# Type Cast - Implicit

- implicit conversion

int / float, float / int ==> float / float

int = float (truncation)

float = int (add .0000)

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

```
int main(void) {
```

```
    int    a; float x;
```

```
    printf("2 /3  = %d\n", 2 /3 );           2 /3  = 0
    printf("2 /3. = %f\n", 2 /3.);          2 /3. = 0.666667
    printf("2./3  = %f\n", 2./3 );          2./3  = 0.666667
    printf("2./3. = %f\n", 2./3.);          2./3. = 0.666667
```

```
    a = 2
    x = 3.000000
```

```
    a = 2.12345; x = 3;
    printf("a = %d\n", a);
    printf("x = %f\n", x);
}
```

# Type Cast Examples

```
#include <stdio.h>

int main(void) {
    int a = 2, b = 3;
    float x, x1, x2, x3, x4 ;

    printf("a= %d\nb= %d\n", a, b);

    x =          (a / b);
    x1 = (float) (a / b);
    x2 = (float) a / (float) b;
    x3 = (float) a / b;
    x4 =          a / (float) b;

    printf("          a / b          = %f\n", x);
    printf("(float) (a / b)          = %f\n", x1);
    printf("(float) a / (float) b = %f\n", x2);
    printf("(float) a / b          = %f\n", x3);
    printf("          a / (float) b = %f\n", x4);
}
```

```
$ gcc -m32 t.c
```

```
$ ./a.out
```

```
a= 2
```

```
b= 3
```

```
          a / b          = 0.000000
```

```
(float) (a / b)          = 0.000000
```

```
(float) a / (float) b = 0.666667
```

```
(float) a / b          = 0.666667
```

```
          a / (float) b = 0.666667
```

# Type Cast Operators

- `(float)`, `(int)`, `(char)` ...
- `(float *)`, `(int *)`, `(char *)` ... pointer type cast
- unary operators
- from right to left associativity
- the same precedence level as `+` and `-` unary operators
- higher precedence level than `+`, `-`, `*`, `/` binary operators