Day06 A

Young W. Lim

2017-10-07

Outline

1. Based on

2. Program Control
   - Overview
   - for, while, do ... while
   - break and continue
   - Relational and Logical Operators
   - Assignment Operators
   - Increment / Decrement Operators

Based on

"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.

Repetition Essentials

- loop : a group of instructions which are executed repeatedly while the loop continuation condition is true
- counter controlled repetition : definite repetition
  - the number of repetition is known in advance
  - counter variable
- sentinel based repetition : indefinite repetition
  - the end of data

Counter Controlled Repetition

- counter variable (loop counter)
  - the initial value
  - the increment / decrement
  - the test condition

for Repetition Statement

1. the control variable is initialized
2. the loop continuation condition is checked
3. if the condition is met, the loop body is executed
4. the control variable is incremented / decremented
5. the loop continuation condition is checked
6. repeat 3~5 until the loop continuation condition fails

```
for (init; cond; update) statement;
```

for Repetition Statement Examples (1)

1. `for (i=1; i<=100; ++i)`
2. `for (i=100; i>=1; --i)`
3. `for (i=1; i<100; i+=2)`
4. `for (i=99; i>=1; i-=2)`
5. `for (i=1; i<=1000; i*=2)`
6. `for (i=1000; i>=1; i/=2)`

```
for (init; cond; update) statement;
```

for Repetition Statement Examples (2)

```
#include <stdio.h>

int main(void) {
  int i=0;

  for (i=1; i<=1024; i*=2)
    printf("i=%d  \n", i);
  printf("----------------------\n");

  for (i=1024; i>=1; i/=2)
    printf("i=%d  \n", i);
  printf("----------------------\n");

  for (i=0; i<=10; i++)
    printf("1<<%2d= %d  \n", i, 1<<i);
  printf("----------------------\n");

  for (i=10; i>=0; i--)
    printf("1<<%2d= %d  \n", i, 1<<i);
  printf("----------------------\n");

}
```

for and while Repetition Statements

- init : initialize the loop control variable
- cond : the loop continuation condition
- update : increments / decrements the loop control variable

```
for (init; cond; update)
  statement;
```

```
init;
while (cond) {
  statement;
  update;
}
```

The optional three statements in for

- init expression is omitted

```
init; for (    ; cond; update) statement;
```

- cond expression is omitted

```
for (init;    ; update) statement;
for (init; 1 ; update) statement;
```

- update expression is omitted

```
for (init; cond;  _) { statement; update; }
```

- infinite loop

```
for (  ;   ;   ) {
  ...
  if (...) break;
  ...
}
```

The comma operator in `for` statement

```
for (exp1, exp2; exp3, exp4; exp5, exp6) statments
```

- the comma operator:
  evaluates the expressions from left to right
- the value of the entire expression :
  the value of the rightmost expression
- condition is checked for exp4,
  after exp3 is executed

The comma operator examples (1)

- for (S=0, i=1; i<11; ++i, S+=i) ;
  - not working
  - when S=0, S+=2 instead of S+=1
- for (S=0, i=1; i<11; S+=i, ++i) ;
  - the final i value is 11
  - S is incremented by 10, before incrementing i to 11
  - after `for` loop, S contains the correct value 55
- for (S=0, i=1; i<11; S+= i++) ;
  - combine two statements S+=i, ++i into one statement
  - S is incremented by old i, before incrementing i

The comma operator examples (2)

```
#include <stdio.h>

int main(void) {
  int i=0, S=0;

  S=0;
  for (i=1; i<11; ++i) {
    S += i;
    printf("i=%d S=%d \n", i, S);
  }

  printf("----------------------\n");
  for (S=0, i=1; i<11; ++i, S+=i) ;   printf("i=%d S=%d (X) \n", i, S);

  printf("----------------------\n");
  for (S=0, i=1; i<11; S+=i, ++i) ;   printf("i=%d S=%d \n", i, S);

  printf("----------------------\n");
  for (S=0, i=1; i<11; S+=i++) ;      printf("i=%d S=%d \n", i, S);

  printf("----------------------\n");
}
```

do ... while Repetition Statement

- if the initial loop continuation condition is false
  - do ... while executes the loop body at least once
  - for and while never execute the loop body

```
init2;
do {
  statement;
  update;
} while (cond2);
```

```
init;
while (cond) {
  statement;
  update;
}
```

```
for (init; cond; update)
  statement;
```

break and continue statements

- break in for, while, do ... while, and switch statements
  - immediate exits from the body
- continue in for, while, do ... while statements
  - skips the remaining statement after continue in the body
  - continue the next iteration of the loop

- while and do ... while
  the loop continuation condition is checked immediately
- for
  the update expression is executed and then
  the loop continuation condition is executed

break example

```
#include <stdio.h>

int main(void) {

  int i;
  int S;


  S=0;
  for (i=0; i<10 ; ++i) {

    if (i==5) break;

    S += (i+1);
    printf("i=%d S=%d \n", i,S);
  }

}
```

continue example

```
#include <stdio.h>

int main(void) {

  int i;
  int S;


  S=0;
  for (i=0; i<10 ; ++i) {

    printf("i%%2 = %d --- ", i%2);

    if ((i%2) == 1) {
      printf("\n");
      continue;
    }

    S += (i+1);
    printf("i=%d S=%d \n", i,S);
  }

}
```

```
#include <stdio.h>

int main(void) {

  int i;
  int S;


  S=0;
  for (i=1; i<10 ; i=i+2) {

    printf("i=%d old S=%d --- ", i,S);
    S += (i+1);
    printf("i=%d S=%d \n", i,S);
  }

}
```

True and False Values

- the result of the following operators : 0 for false, 1 for _true
  - relational operators (>, <, >=, <=)
  - equality operators (==, !=)
  - logical operators (&&, ||, !)
- accepts nonzero value as a true

The results of relational operators

```
int main(void) {
   int a =10;
   float x =3.14;

   printf(" a = %d \n", a);
   printf("(a > 1) = %d \n", (a>1) );
   printf("(a < 1) = %d \n", (a<1) );

   printf(" x = %f \n", x);
   printf("(x > 1) = %d \n", (x>1) );
   printf("(x < 1) = %d \n", (x<1) );
}
```

```
int main(void) {
   if (0) printf("0 is true \n");
   else   printf("0 is false \n");

   if (1) printf("1 is true \n");
   else   printf("1 is false \n");

   if (-999) printf("-999 is true \n");
   else      printf("-999 is false \n");

   if (3.14999) printf("3.14999 is true \n");
   else         printf("3.14999 is false \n")

}
```

Logical Operators

- to form a complex conditions
- logical AND : && (true when all operands are true)
- logical OR : || (true when any operand is true)
- logical NOT : !

logical AND	logical OR
true when all operands are true	true when any operand is true
false when any operand is false	false when all operands are false

Evaluation of relational operators

- `&&` is higher precedence over `||`
- `A && B`
  - if A is true, then B is evaluated
  - if A is false, then B needs not be evaluated (false)
- `A || B`
  - if A is true, then B needs not be evaluated (true)
  - if A is false, then B is evaluated

Assignment Operators

	first compute	then assign
a += b	a + b	a = a + b
a -= b	a - b	a = a - b
a *= b	a * b	a = a * b
a /= b	a / b	a = a / b
a %= b	a % b	a = a % b

Assignment Operator Examples (1)

```
#include <stdio.h>

int main(void) {
  int m = 100;
  int n = 20;

  printf("m=100, n=20 \n");

  m=100; n=20;  m = m + n; printf("m = m + n = %d \n", m);
  m=100; n=20;  m = m - n; printf("m = m - n = %d \n", m);
  m=100; n=20;  m = m * n; printf("m = m * n = %d \n", m);
  m=100; n=20;  m = m / n; printf("m = m / n = %d \n", m);

  m=100; n=20;  m += n; printf("m += n = %d \n", m);
  m=100; n=20;  m -= n; printf("m -= n = %d \n", m);
  m=100; n=20;  m *= n; printf("m *= n = %d \n", m);
  m=100; n=20;  m /= n; printf("m /= n = %d \n", m);
```

Assignment Operator Examples (2)

```
  m=100; n=20;  printf("m = m + n --> %d \n", m = m + n);
  m=100; n=20;  printf("m = m - n --> %d \n", m = m - n);
  m=100; n=20;  printf("m = m * n --> %d \n", m = m * n);
  m=100; n=20;  printf("m = m / n --> %d \n", m = m / n);

  m=100; n=20;  printf("m += n --> %d \n", m += n);
  m=100; n=20;  printf("m -= n --> %d \n", m -= n);
  m=100; n=20;  printf("m *= n --> %d \n", m *= n);
  m=100; n=20;  printf("m /= n --> %d \n", m /= n);

  m = n = 999;
  printf("m=%d, n=%d \n", m, n );

}
```

Increment / Decrement Operators

	first update	then access
b = ++a;	a = a + 1;	b = a
b = – –a;	a = a – 1;	b = a

	first access	then update
b = a++;	b = a;	a = a + 1;
b = a– –;	b = a;	a = a – 1;

Pre-increment / Post-increment Examples

```
#include <stdio.h>

int main(void) {
  int i;

  i = 100;
  printf("++i= %d \n", ++i); // pre-increment
  printf("i  = %d \n",   i);

  i = 100;
  printf("i++= %d \n", i++); // post-increment
  printf("i  = %d \n",   i);

  i = 100;
  printf("--i= %d \n", --i); // pre-decrement
  printf("i  = %d \n",   i);

  i = 100;
  printf("i--= %d \n", i--); // post-decrement
  printf("i  = %d \n",   i);
}
```