Advanced C Programming & Lab

13. Advanced Expressions / Functions / Data Types

Sejong University

Outline

1) **Bit Expressions and Operators**

2) **Recursive Functions**

3) **Library**

1) Bit Expressions and Operators

- **Bit Expressions and Operators?**
  - Bitwise operations
  - Bit: True 1, False 0

  - Bitwise logical operations
  - Ex) Bitwise AND
    - ✓ True if both bits are true (1)

```
          0010 1010
          1010 1101
          ---------
(result)  0010 1000
```

1) Bit Expressions and Operators

- **Bitwise logical operations**
  - Ex) Bitwise AND : x **&** y
    - ✓ Logical AND operation on each pair of the bits in x and y
  - Convenient to represent as hexadecimal numbers

```
int x = 0X2A;  // x = 0000 0000 … 0010 1010
int y = 0XAD;  // y = 0000 0000 … 1010 1101
int z = x & y; // z = 0000 0000 … 0010 1000

printf("%#X", z);

Result:
0X28
```

1) Bit Expressions and Operators

- **Bitwise shifts**
  - Ex) Bitwise left shift : x << k
    - ✓ Bits are shifted by k places
    - ✓ The blank spaces (on the right) are filled by 0s

```
int x = 0X2A01234C;    // x = 0010 1010 … 0000 1100
int z = x << 4;        // z = 1010 0000 … 1100 0000

printf("%#X", z);

Result:
0XA01234C0
```

1) Bit Expressions and Operators

- **Bitwise operations in C**
  - Available: int, char

Operator	Operation	Example (8bits)
&	Bitwise AND	0110 1100 & 0100 1010 → 0100 1000
|	Bitwise OR	0110 1100 | 0100 1010 → 0110 1110
^	Bitwise XOR	0110 1100 ^ 0100 1010 → 0010 0110
~	Bitwise NOT	~0110 1100 → 1001 0011
<<	Left shift	0100 1010 << 2 → 0010 1000
>>	Right shift	0100 1010 >> 2 → 0001 0010

1) Bit Expressions and Operators

- **[Practice 1] Perform logical operations.**

  - Declare unsigned char type variables,
    Assign 0100 1100 and 0100 1010 to the variables as hexadecimal numbers

  - Perform the logical operations (previous slide), store the results in unsigned char type variables, and print them as hexadecimal numbers

1) Bit Expressions and Operators

- **Shift right(>>): signed vs. unsigned**
  - If the operand is signed and leftmost bit is 1, filled by 1s
    - ✓ Leftmost bit is 1 → negative number
    - ✓ After shift operation, still be a negative number

```
int x, zs, zu;

x = 0XA1234567;                 // x = 1010 0001 …
zs = (signed) x >> 4;           // result: 1111 1010 0001 …
zu = (unsigned) x >> 4;         // result: 0000 1010 0001 …
printf("signed = %#X, unsigned = %#X\n", zs, zu);

result:
signed = 0XFA123456, unsigned = 0XA123456
```

1) Bit Expressions and Operators

- Otherwise, filled by 0s
  - ✓ The operand is unsigned or leftmost bit is 0

```
int x, zs, zu;

x = 0X71234567;                 // x = 0111 0001 …
zs = (signed) x >> 4;           // result: 0000 0111 0001 …
zu = (unsigned) x >> 4;         // result: 0000 0111 0001 …
printf("signed = %#X, unsigned = %#X\n", zs, zu);

result:
signed = 0X7123456, unsigned = 0X7123456
```

1) Bit Expressions and Operators

- **(Example 1) Represent an integer N as a binary number. Print the bit in the 10th place from the right side**
  - Assume: rightmost bit is in the 0th place

```
int x = 0X71234567;           // … 0101 …

x = x >> 10;       // 10th bit is placed
                   // in the rightmost place
x = x & 1;         // AND between x and … 0001
                   // ok to combine them (x >> 10) & 1

printf("%d\n", x);

Result:
1
```

Outline

1) Bit Expressions and Operators

2) **Recursive Functions**

3) Library

2) Recursive Functions

- **Does the following program work properly? YES**
  - No problem at all
    - ✓ Call dec( ) in dec()??
    - ✓ What happened to the current dec( )??

```
void dec(int x)
{
   printf("x: %d\n", x);
   if( x > 1) dec(x-1);
}
void main()
{
   dec(3);
}
```

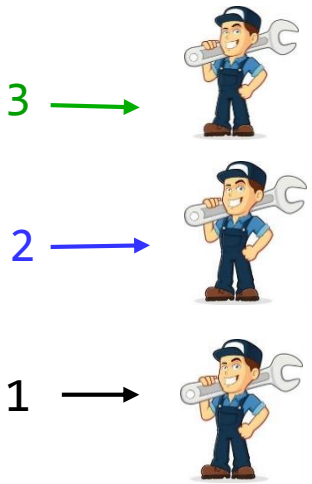
Result

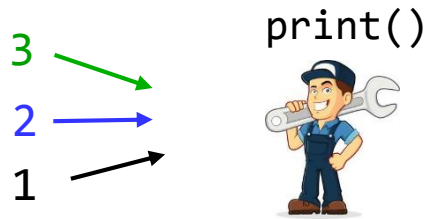
```
x: 3
x: 2
x: 1
```

2) Recursive Functions

- **Procedure**
  - How does it work?

```
void print(int x){
    printf("x: %d\n", x);
}
void main(){
    print(3);
    print(2);
    print(1);
}
```





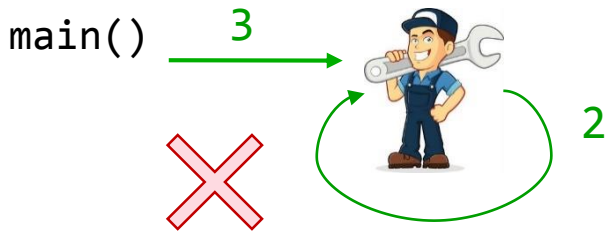
3 (identical) functions process the 3 jobs??

1 function processes the 3 jobs??

2) Recursive Functions

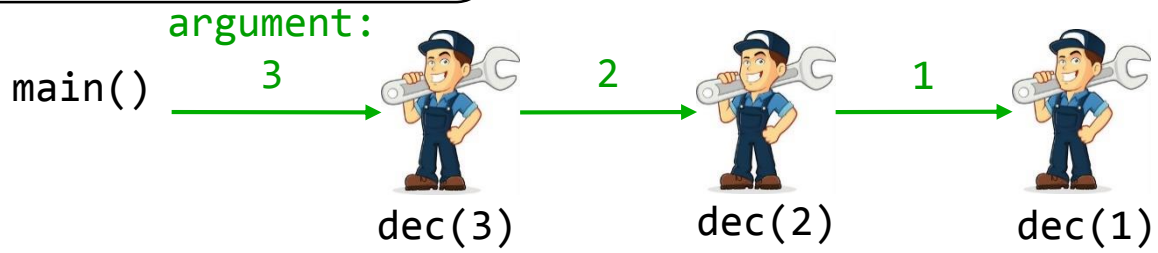
- **Procedure**
  - How does it work?

```
void dec(int x)
{
   printf("x: %d\n", x);
   if( x > 1) dec(x-1);
}
void main()
{
   dec(3);
}
```



Do not call itself

A function dec() calls another identical function dec()



2) Recursive Functions

- **Recursive functions?**
  - A procedure to calls (or references) itself

  - Similar: Recurrence relation
    - ✓ $A_n = A_{n-1} + 2$

    - ✓ A : Function name
    - ✓ subscript n : Function argument

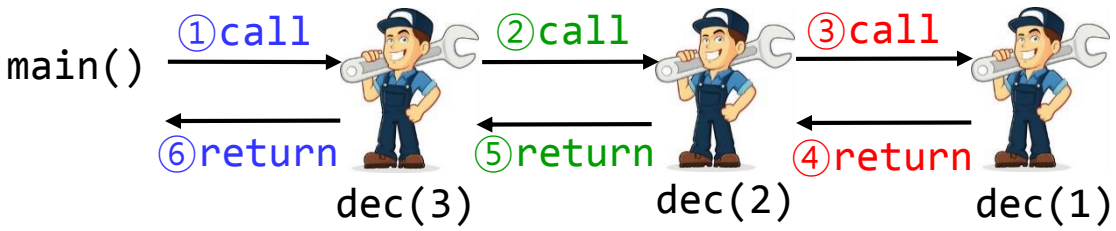
2) Recursive Functions

- **Procedure: Check the internal processes**

```
void dec(int x)
{
   printf("+start: x=%d\n",x);
   if( x > 1) dec(x-1);
   printf("-end: x=%d\n",x);
}
void main()
{
   dec(3);
}
```

Result

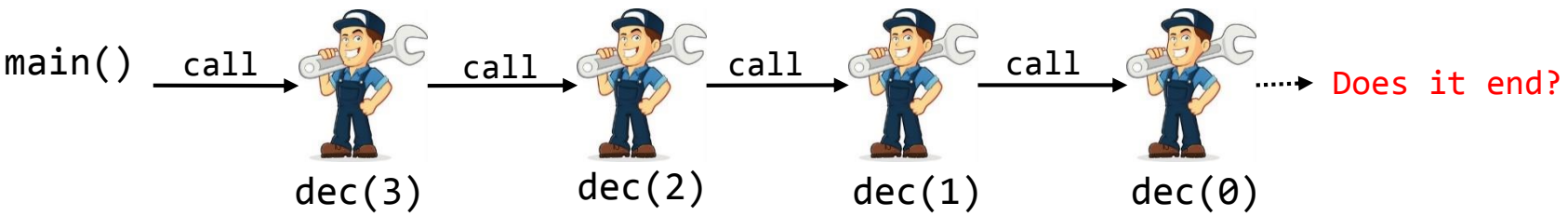
```
+start: x=3
+start: x=2
+start: x=1
-end: x=1
-end: x=2
-end: x=3
```



2) Recursive Functions

- **Notes: If missing the terminating condition, it won't work properly**
- **Recurrence relation without the initial value**
  - ✓ $A_n = A_{n-1} + 2$, $A_1 = 1$

```
void dec(int x){
    printf("x: %d\n", x);
    if( x > 1) dec(x-1);
}
void main(){
    dec(3);
}
```



2) Recursive Functions

- **(Example 2) Compute the sum from 1 to n using recursive functions**
  - sum(n) = 1+2+3+ … + (n-1) + n: Recurrence relation?
    - ✓ sum(n) = sum(n-1) + n (n > 1)
    - ✓ sum(1) = 1; (n==1)

```
int sum(int n){
   int s;      // sum
   if( n == 1)  s = 1;
   else   s = sum(n-1) + n;
   return s;
}
void main()
{
   printf("%d", sum(10) );
}
```

```
// simple version

int sum(int n){
   if( n == 1)  return 1;
   return sum(n-1) + n;
}
void main()
{
   printf("%d", sum(10) );
}
```

2) Recursive Functions

- **[Practice 2] Compute n! using recursive functions.**
  - ✓ n! = n*(n-1)! (n > 1)
  - ✓ 1! = 1; (n==1)

Outline

1) Bit Expressions and Operators

2) Recursive Functions

3) **Library**

3) Library

- **Random number generation**
  - Random number? A number chosen as if by chance
  - C provides functions to generate random numbers
  - Functions: rand( ) , srand( ) , time ( )

- **Measure the running time**
  - C provides time functions: can obtain time information
  - Functions: clock( ),

3) Library

- **Random number generator**
  - rand( ): available in <stdlib.h>
    - ✓ Return a rand number in the range 0~RAND_MAX
    - ✓ RAND_MAX is a constant (32767) defined in stdlib.h

```
#include <stdio.h>
#include <stdlib.h>   // rand()

void main()
{  int i;

   for( i=0 ; i < 5 ; ++i)   // 5 random numbers
      printf(" %d", rand() );
}
Result:
 41 18467 6334 26500 19169
```

3) Library

- Execute the previous code multiple times. Results are the same?
- srand( ): change “seed”
  - ✓ Available in<stdlib.h>
  - ✓ Modify function arguments: srand()

```
#include <stdio.h>
#include <stdlib.h>   // rand()

void main()
{  int i;
   srand(10);          // seed: 10
   for( i=0 ; i < 5 ; ++i)   // 5 random numbers
printf(" %d", rand() );
}
Result:
 71 16899 3272 13694 13697
```

3) Library

- Use time( ) to change seed each time it is executed
  - ✓ Return the current system time
  - ✓ Available in <time.h>

```
#include <stdio.h>
#include <stdlib.h>   // rand()
#include <time.h>     // time()

void main()
{  int i;
   srand( time(NULL) );   // seed: current time
for( i=0 ; i < 5 ; ++i)       // 5 random numbers
      printf(" %d", rand() );
}
Result: different each time
```

3) Library

- **[Practice 3] Generate random numbers in the range from 0 to 100**
  - Should generate different numbers each time it is executed
  - (hint) modulo (%)

3) Library

- **(Example 3) Generate one random number in the range [min, max)**

```
#include <stdio.h>
#include <stdlib.h>   // rand ()
#include <time.h>     // time ()

int random_num(int min, int max)
{
   int rand_num;
   rand_num =
    (double) rand() / (RAND_MAX +1) * (max-min) + min;
   return rand_num;
}
```

3) Library

- **Running time**
  - clock( ): Available in <time.h>
    - ✓ Return the current system time when it is called, in the unit of CLOCKS_PER_SEC (defined in time.h)
    - ✓ To get time in seconds, should divide the returned time by CLOCKS_PER_SEC

3) Library

- **Example code**

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

void main( void )
{  clock_t start, finish;
   double  duration;

   start = clock();   // Starting time
    // Some code lines….
   finish = clock();  // Finishing time

   duration = (double)(finish-start) / CLOCKS_PER_SEC;

   printf(“Running time: %lf seconds\n", duration);
}
```