CSE320 System Fundamentals II
Hello World

YoungMin Kwon

Some UNIX commands

- Directory
  - ls: list directory contents.
    e.g. ls -al
  - mkdir: make a directory.
    e.g. mkdir abc
  - cd: change directory.
    e.g. cd abc, cd ..
  - rmdir: remove a directory.
    e.g. rmdir abc
  - pwd: print working directory.
    e.g. pwd

Some UNIX commands

- File
  - `cp`: copy files.
    `e.g. cp * abc/, cp a.txt b.txt`
  - `mv`: move files.
    `e.g. mv abc/* bcd/*, mv a.txt b.txt`
  - `cat`: print the contents of a file.
    `e.g. cat a.txt`
  - `grep`: looking for a pattern.
    `e.g. grep hello *`
- man (manual page)
  - section number 2 is for system calls, 3 is for library routines
  - `man 3 printf`
  - `man 2 fork`
  - `man sin`

Hello.c

```c
// #include tells the compiler to copy the contents of
// the include file to this file.
// The line below will copy the contents of stdio.h to hello.i
//     try run  gcc -E hello.c -o hello.i
#include <stdio.h>

// #define creates a macro
#define HELLO "Hello world\n"

// main is the function that starts the program
int main() {

    // printf prints out the macro string to the screen
    printf(HELLO);

    // returning 0 from main indicates a normal completion.
    // returning non-zero means abnormal termination.
    return 0;
}
```

Compiling Hello.c

- To compile hello.c

  ```
  gcc hello.c
  ```

  - It will make `a.out.`
  - To make hello instead of a.out run

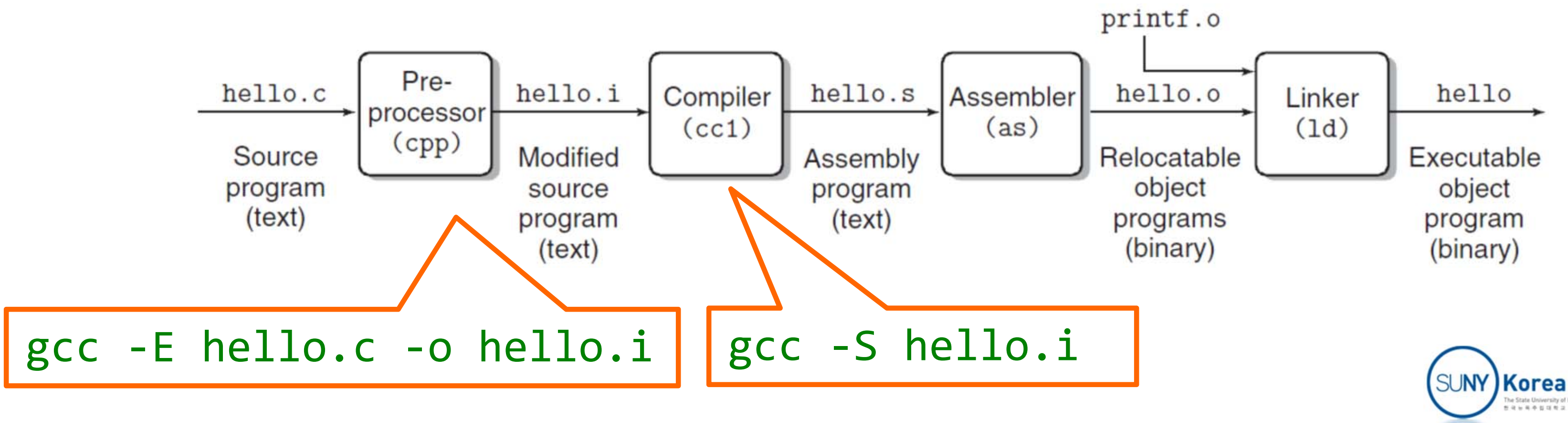
    ```
    gcc -o hello hello.c
    ```

- To run a.out

  ```
  ./a.out
  ```

- It will print out

  ```
  Hello world
  ```



```
gcc -E hello.c -o hello.i
```

```
gcc -S hello.i
```

Values in C

- In C, literal values are numbers
  - ‘a’: ASCII code 97
  - 20: integer value 20
  - 20L: long value of 20
  - 3.14: double value of 3.14
  - 3.14F: floating point value of 3.14
  - “hello”: the address of the string “hello”
  - main: the address of the function main

printf

```
int main() {
    printf("char:       %c, %d\n", 'a', 'a');
    printf("int:        %c, %d\n", 97,  97);
    printf("hex number: %x\n", 97);
    printf("float:      %f\n", 3.14f);
    printf("double:     %lf\n", 3.14);
    printf("string:     %p, %s\n", "hello", "hello");
    printf("function:   %p\n", main);
}
```

```
youngmin.kwon@momgoose:~/home/cse320$ ./a.out
char:       a, 97
int:        a, 97
hex number: 61
float:      3.140000
double:     3.140000
string:     0x5590f1808061, hello
function:   0x5590f1807139
```

More on printf

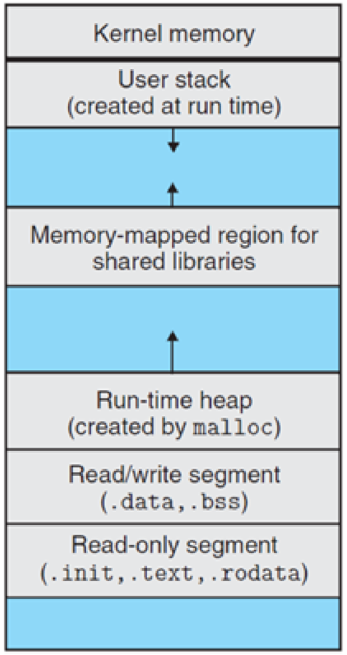
```
//the value of string literals are their addresses
printf("string:    %ld, %s\n", (long)"hello", "hello");

//more about strings
printf("%p, %c, %c, %c, %c, %c\n", "hello", "hello"[0],
        "hello"[1], "hello"[2], "hello"[3], "hello"[4]);

char *str = "hello"; //str points to the address of "hello"
printf("%p, %c, %c, %c, %c, %c\n", str, str[0], str[1],
                                str[2], str[3], str[4]);

long adr  = (long)"hello"; //cast an address to long
printf("%ld, %c, %c, %c, %c, %c\n", adr, ((char*)adr)[0],
                        ((char*)adr)[1], ((char*)adr)[2],
                        ((char*)adr)[3], ((char*)adr)[4]);
```

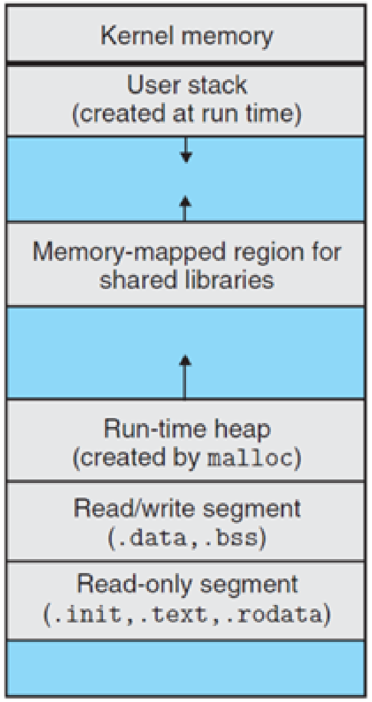
```
string:     94836899717217, hello
0x5640efb82061, h, e, l, l, o
0x5640efb82061, h, e, l, l, o
94836899717217, h, e, l, l, o
```



More on printf

```
//function pointers
printf("main:   %p\n", main);
printf("printf: %ld\n", (long)printf);

main:   0x5640efb81139
printf: 139928006843920
```



C: Call by Value

- Parameter passing modes
  - Call by value: values of the parameters are passed
  - Call by reference: addresses of the parameters are passed
  - Call by name: parameters are passed as literal substitution (lazy evaluation, e.g. lambda calculus)
  - Call by need: call by name + memorization

- In C, for a callee to change the caller's parameter
  - Parameter's address needs to be passed
  - Callee needs to change the content of the address

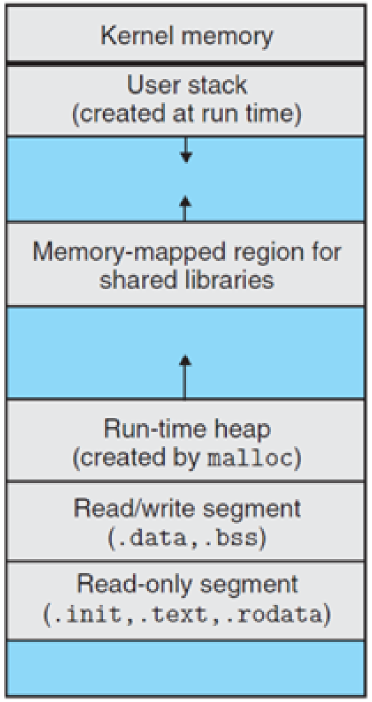
scanf: to read user’s input

```c
#include<stdio.h>
int main() {
    // read a string
    char name[100];
    printf("Enter your name: ");
    scanf("%99s", name); // name: address of the array
    printf("hello %s.\n", name);

    // read an integer number
    int num;
    printf("Enter a number: ");
    scanf("%d", &num); // &num is the address of num
    printf("read %d.\n", num);

    return 0;
}
```

```
Enter your name: youngmin
hello youngmin.
Enter a number: 1
read 1.
```



More on scanf

```c
#include<stdio.h>
#include<math.h>
int main() {
    // read a floating point number
    float fnum;
    printf("Enter a floating point number: ");
    scanf("%f", &fnum);
    printf("read %f.\n", fnum);
    printf("sin(%f) = %f.\n", fnum, sin(fnum));
    return 0;
}
```

- How to compile the program above.

```
youngmin.kwon@momgoose:~/home/cse320/intro$ gcc scan.c
/tmp/ccNDavtQ.o:scan.c:function main: error: undefined reference to 'sin'
collect2: ld returned 1 exit status
```

-lm option will fix the error (link math library)

```
gcc scan.c -lm
```

```
Enter a floating point number: 0.21
read 0.210000.
sin(0.210000) = 0.208460.
```

Arithmetic Operators

```c
#include<stdio.h>
int main() {
    int a = 0xff, b = 0x05, c = 0x50;
    printf("a: %5d, b: %5d, a + b: %5d\n", a, b, a + b);
    printf("a: %5d, b: %5d, a - b: %5d\n", a, b, a - b);
    printf("a: %5d, b: %5d, a * b: %5d\n", a, b, a * b);
    printf("a: %5d, b: %5d, a / b: %5d\n", a, b, a / b);
    printf("a: %5d, b: %5d, a %% b: %5d\n", a, b, a % b);
    return 0;
}
```

```
a:   255, b:     5, a + b:   260
a:   255, b:     5, a - b:   250
a:   255, b:     5, a * b:  1275
a:   255, b:     5, a / b:    51
a:   255, b:     5, a % b:     0
```

Bitwise Operators

```
printf("a: %5d, b: %5d, a & b: %5d\n", a, b, a & b); // and
printf("c: %5d, b: %5d, c | b: %5d\n", c, b, c | b); // or
printf("a: %5d, b: %5d, a ^ b: %5d\n", a, b, a ^ b); // xor

// One’s complement vs Two’s complement
printf("b: %5d, ~b: %5d (%x)\n", b, ~b, ~b);    // one’s complement
printf("-1: %x, -2: %x, -3: %x\n", -1, -2, -3); // two’s complement

printf("b: %5d, b << 1: %5d\n", b, b << 1); // shift left
printf("b: %5d, b >> 1: %5d\n", b, b >> 1); // shift right

a:   255, b:     5, a & b:     5
c:    80, b:     5, c | b:    85
a:   255, b:     5, a ^ b:   250
b:     5, ~b:    -6 (fffffffa)
-1: ffffffff, -2: fffffffe, -3: fffffffd
b:     5, b << 1:    10
b:     5, b >> 1:     2
```

Flags and Masks

```
#include <stdio.h>

//gender
#define MALE    (0)
#define FEMALE  (1)
#define SET_GENDER(data, gender)    ((data) | (gender) << 31)
#define GET_GENDER(data)            ((data) >> 31 & 1)

//role
#define STUDENT (0)
#define STAFF   (1)
#define FACULTY (2)
#define SET_ROLE(data, role)        ((data) | (role) << 29)
#define GET_ROLE(data)              ((data) >> 29 & 3)

//department
#define AMS     (0)
#define BUS     (1)
#define CS      (2)
#define DTS     (3)
#define ECE     (4)
#define MEC     (5)
#define SET_DEPT(data, dept)        ((data) | (dept) << 26)
#define GET_DEPT(data)              ((data) >> 26 & 7)
```

Flags and Masks

```
//id (20bit, upper 6 bits are reserved)
#define SET_ID(data, id)            ((data) | (id))
#define GET_ID(data)                ((data) & 0xfffff)

int main() {
    char* str_gndr[] = {"male", "female"};
    char* str_role[] = {"student", "staff", "faculty"};
    char* str_dept[] = {"AMS", "BUS", "CS", "DTS", "ECE", "MEC"};
    unsigned int a = 0;
    a = SET_GENDER(a, FEMALE);
    a = SET_ROLE(a, STUDENT);
    a = SET_DEPT(a, CS);
    a = SET_ID(a, 30);
    printf("gender:     %s\n", str_gndr[GET_GENDER(a)]);
    printf("role:       %s\n", str_role[GET_ROLE(a)]);
    printf("department: %s\n", str_dept[GET_DEPT(a)]);
    printf("id:         %d\n", GET_ID(a));
}
```

```
Output
gender:     female
role:       student
department: CS
id:         30
```

Logical Operators

```
printf("a: %5d, b: %5d, a == b: %5d\n", a, b, a == b);
printf("a: %5d, b: %5d, a != b: %5d\n", a, b, a != b);
printf("a: %5d, b: %5d, a > b:  %5d\n", a, b, a > b);
printf("a: %5d, b: %5d, a >= b: %5d\n", a, b, a >= b);
printf("a: %5d, b: %5d, a < b:  %5d\n", a, b, a < b);
printf("a: %5d, b: %5d, a <= b: %5d\n", a, b, a <= b);

printf("a: %5d, b: %5d, a && b: %5d\n", a, b, a && b);
printf("a: %5d, b: %5d, a || b: %5d\n", a, b, a || b);
printf("a: %5d, !a: %4d, !!a: %5d\n", a, !a, !!a);
```

```
a:   255, b:     5, a == b:     0
a:   255, b:     5, a != b:     1
a:   255, b:     5, a > b:      1
a:   255, b:     5, a >= b:     1
a:   255, b:     5, a < b:      0
a:   255, b:     5, a <= b:     0
a:   255, b:     5, a && b:     1
a:   255, b:     5, a || b:     1
a:   255, !a:    0, !!a:        1
```

Side Effects from a Compiler Optimization

```
a > b && printf("a > b && print\n");
a < b && printf("a < b && print\n");
a > b || printf("a > b || print\n");
a < b || printf("a < b || print\n");

a > b && print
a < b || print

// gcd: Euclidean algorithm
int gcd(int a, int b)
{
    return  a > b && gcd(a - b, b) ||
            a < b && gcd(b - a, a) ||
            printf("gcd: %d\n", a);
}
```

Programming Assignment 1

- Implement hexstr_to_num and num_to_binstr functions in the next page
  - hexstr_to_num converts a hex string to an integer number (e.g. “aB” to 171)
  - num_to_binstr converts a number to a binary string (e.g. 6 to “110”)
- Use Blackboard to submit the assignment.
- Due date: 9/8/2022, 11:59pm

```c
//printnum.c
#include <stdio.h>
#include <stdlib.h>

int hexstr_to_num(char *hex) {
    //hint: 'c' - 'a' + 10 = 12
}

void num_to_binstr(int num, char *bin) {
    //hint: num & 1 << i checks the i-th bit of num
}

int main() {
    char hexstr[9];
    char binstr[33];
    int num;
    printf("Enter a hexadesimal number: ");
    scanf("%8s", hexstr);

    num = hexstr_to_num(hexstr);
    printf("dec: %d\n", num);

    num_to_binstr(num, binstr);
    printf("bin: %s\n", binstr);

    return 0;
}
```

Expected result

```
./a.out
Enter a hexadesimal number: a1
dec: 161
bin: 10100001
```