NETWORK PROGRAMMING

Assoc. Prof. Truong Dieu Linh
Data Communications & Computer Networks
SoICT, HUST

Course information

- IT4062E: Network programming
- Webpage of the course
  - https://soict.hust.edu.vn/~linhtd/courses/NetworkProg/
- Instructor email: linhtd@soict.hust.edu.vn
  - For making appointment only.
- What we study in this course
  - How to build network applications using socket programming paradigm.
  - Socket programming using C (in details)
  - Socket programming in Java (briefly)
- Reference:
  - UNIX® Network Programming Volume 1, Third Edition: The Sockets Networking API, W. Richard Stevens,Bill Fenner,Andrew M. Rudoff
  - https://notes.shichao.io/unp/ch7/

Course contents

- Lecture contents
  - Review of C programming language
  - Review of related concept in Computer Networks
  - Introduction to Socket API
  - Basic TCP socket: server side, client side
  - UDP socket
  - Multi-thread TCP server
  - Socket programming with Java.
- Exercises in class
  - After each lecture
- Final project
  - Development of network applications in groups
  - 3 members/ group.
  - Used for mid-term and final evaluations

REVIEW C PROGRAMMING

Truong Dieu Linh
SoICT, HUST

Content

- Data type
- Condition and Loop statement
- Function
- Command line argument
- Pointer
- Structure
- Link listed
- I/O function

Data type

- Integer
  - int, char, short, long
- Floating
  - double, float
- Array
  - Collection of A data type
  - Declaration : int a[10];

Size of Type

size of char: 1 bytes

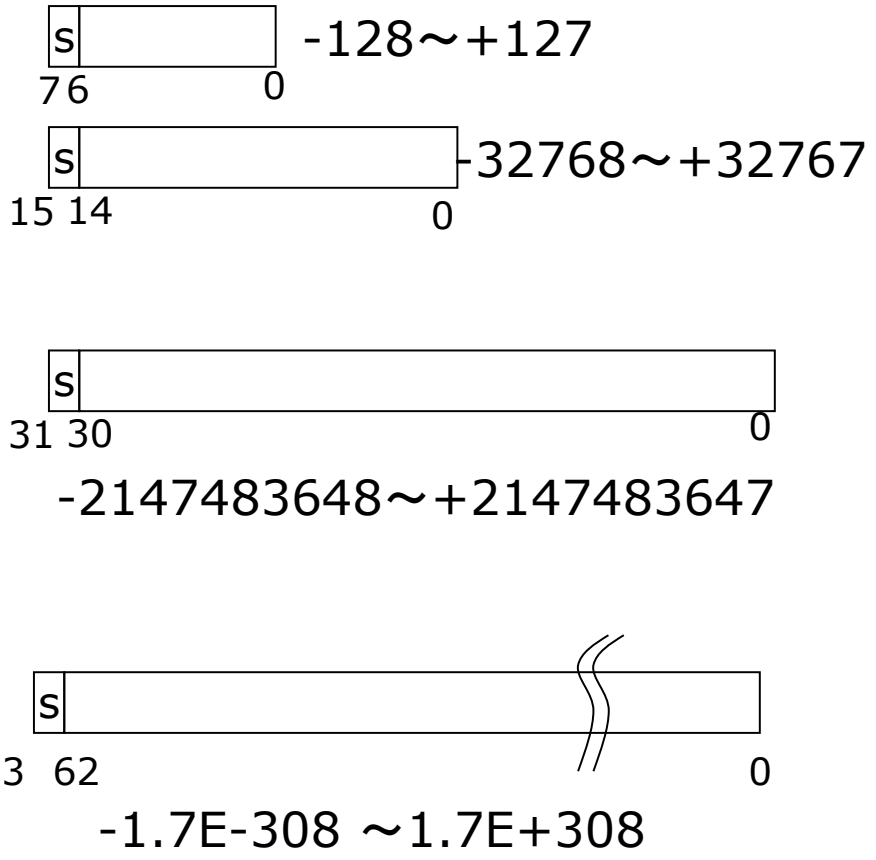
size of short: 2 bytes

size of int: 4 bytes

size of long: 4 bytes

size of float: 4 bytes

size of double: 8 bytes



Condition and Loop statement

- if … else
- switch
- for
- while, do … while

Condition

- a == b
  - b equals to a
- a != b
  - b is different to a
- a > b
  - b is smaller than a
- a >= b
  - b isn’t greater than a
- a < b
  - b is greater than a
- a <= b
  - b isn’t smaller than a

if … else

```
if (condition){
   statement1 ;
   …
}
else{
   statement2 ;
   …
}
```

Example :

```
if (x == 1){
   y = 3;
   z = 2;
}
else{
   y = 5;
   z = 4;
}
```

switch

```
switch (condition)
{
   case value1: statement1;…; break;
   case value2: statement2;…; break;
   …
   default: statementn;…; break;
}
```

Example :

```
int monthday( int month ){
switch(month)
{
   case 1: return 31;
   case 2: return 28;
   …
   case 12: return 31;
   }
}
```

for

```
for (condition1 ; condition2 ; condition3)
{
  statements ;
  …
}
```

Example :

```
for (x = 0; x < 10; x = x +1)
{
  printf(“%d\n”, x);
}
```

while

```
while(condition){
   statement;
   …
}
```

Example:

```
x = 0;
while( x < 10 ){
   printf(“%d\n”,x);
   x = x + 1;
}
```

break and continue

- break
  - Terminates the execution of the nearest enclosing loop or conditional statement in which it appears.
- continue
  - Pass to the next iteration of then nearest enclosing do, for, while statement in which it appears
- Example

```
for( i=0; i<100; i++ ){
   statement 1;
   if ( i==90 )    continue;
   statement 2;
}

for( i=0; i<100; i++ ){
   statement 1;
   if ( i==90 )    break;
   statement 2;
}
```

Function

- A function is a group of statements that is executed when it is called from some point of the program.
- Function format:

  type function_name (parameter1, parameter2, ...)
  { statements }
- where:
  - *type* is the type of the data returned by the function.
  - *function_name*.
  - *parameters*
  - Statements: function's body.

Example of function

```
#include <stdio.h>

int squaresub(int a)
{
      return a*a;
}

int main()
{
      int   b = 10;
      printf("%d\n", squaresub(5));
      return 0;
}
```

Data type of function

Return value statement

Use function

Usage of command line arguments

- main(int argc, char **argv)
- main(int argc, char *argv[])

- Argc : number of arguments
- argv[0] : command name
- argv[1] : 1st argument
- argv[2] : 2nd argument

Example :

%./a.out 123 456 789

arg[0]: ./a.out

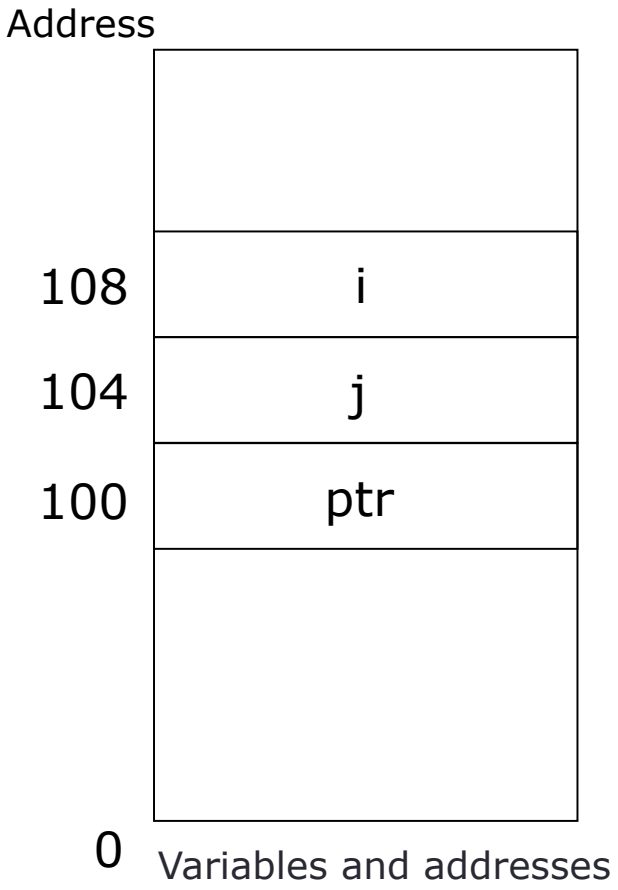
arg[1]: 123

arg[2]: 456

arg[3]: 789

Pointer

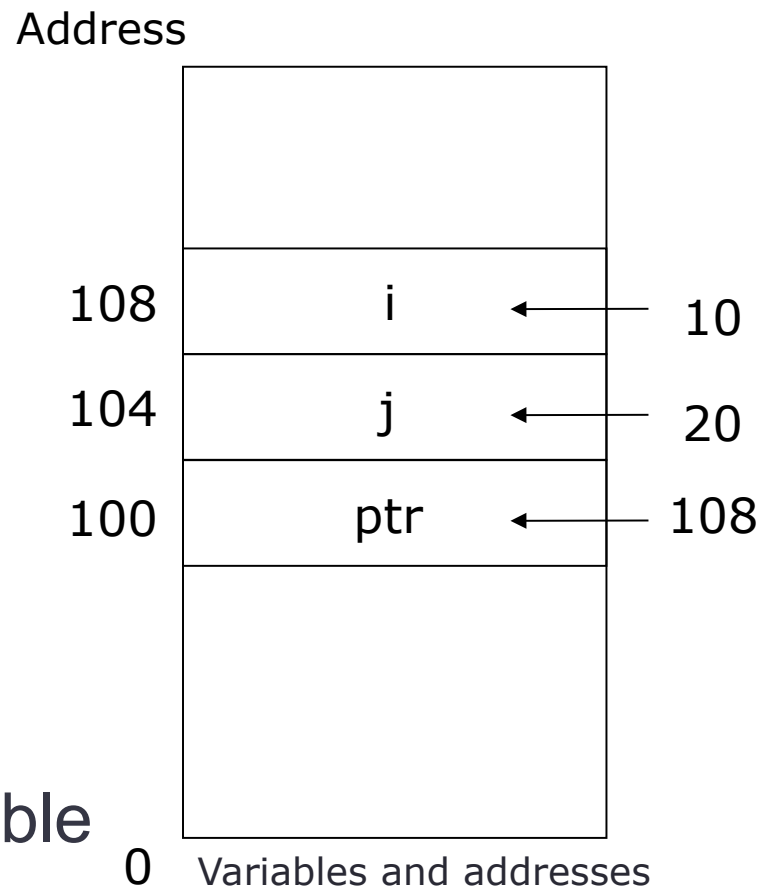
- Pointer variable
  - "Variable" refers to variable
  - Value of the pointer is the address of the variable in the memory
- int i = 10;
- int j = 20;
- int *ptr



Pointer (cont)

- int i = 10;
- int j = 20;
- int *ptr = &i;

- printf("i=%d\n", &i)
- printf("ptr=%d\n", ptr)
- printf("i=%d\n", i)
- printf("*ptr=%d\n",*ptr)
- Ptr refers to the pointer variable



Pointer (cont)

- int x=1, y=5;
- int z[10];
- int *p;
- p=&x; /* p refers to x   */
- y=*p; /*y is assigned the value of x*/
- *p = 0; /* x = 0   */
- p=&z[2]; /* p refer to z[2] */

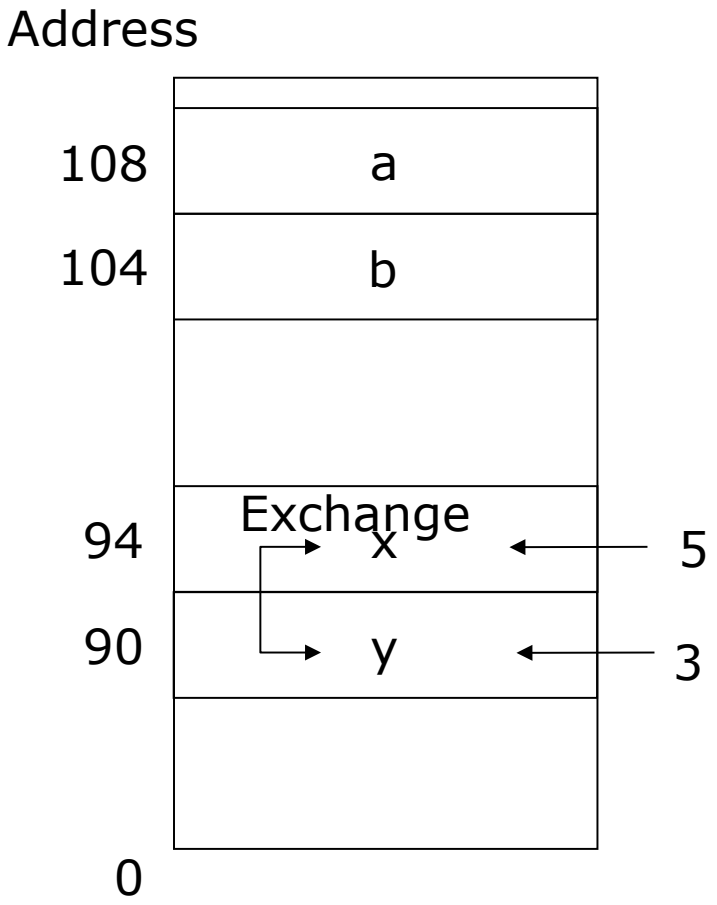
Pointer and function

```
#include <stdio.h>
void swap(int x, int y)
{
      int temp;
      temp = x;
      x = y;
      y = temp;
}

int main(){
      int a = 5;
      int b = 3;
      swap (a,b);
      printf("a=%d\n", a);
      printf("b=%d\n",b);
      return 0;
}
```

Result ?

Pointer and function (cont)

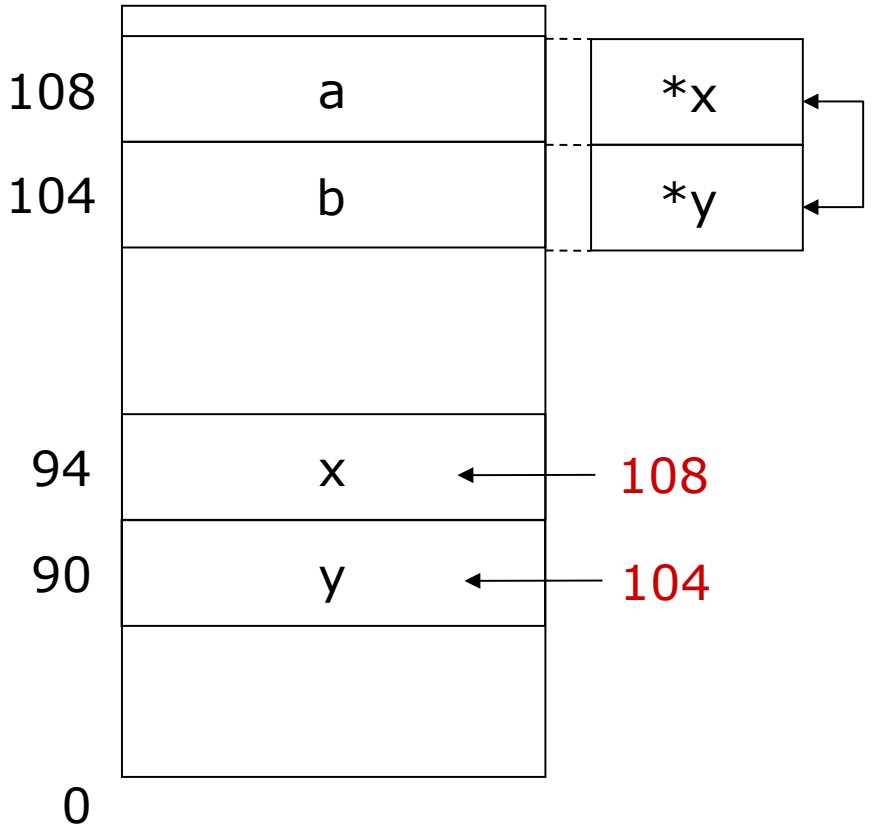
```
#include <stdio.h>
void swap(int x, int y)
{
      int temp;
      temp = x;
      x = y;
      y = temp;
}

int main(){
      int a = 5;
      int b = 3;
      swap (a,b);
      printf("a=%d\n", a);
      printf("b=%d\n",b);
      return 0;
}
```



Pointer and function (cont)

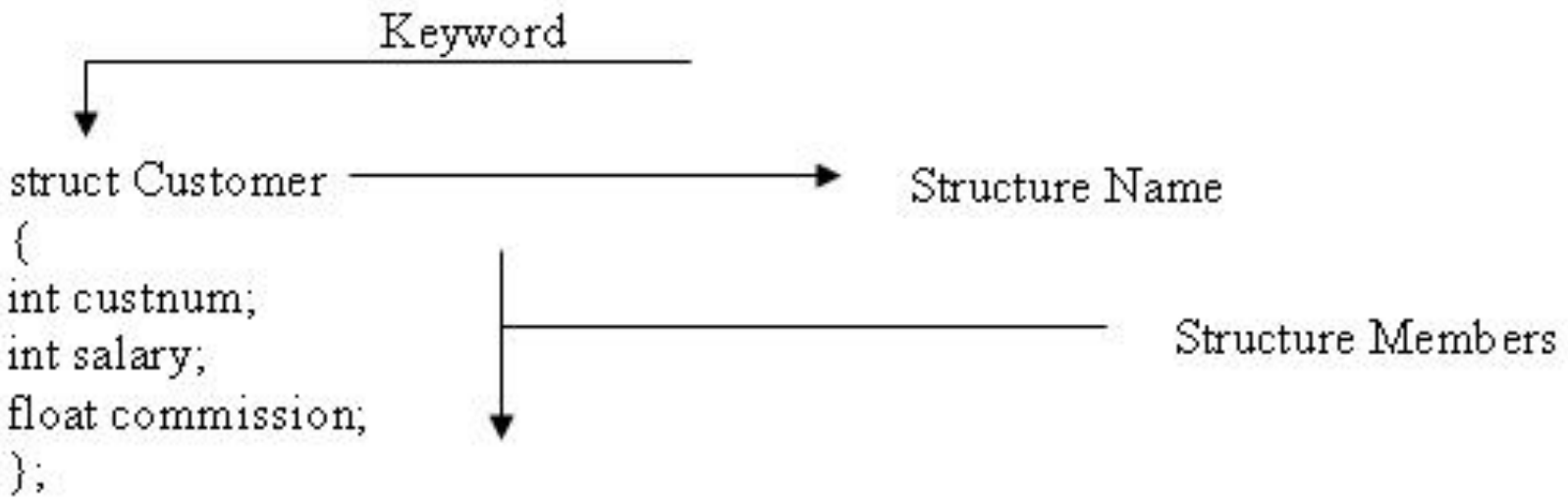
```
#include <stdio.h>
void swap(int *x, int *y)
{
      int temp;
      temp = *x;
      *x = *y;
      *y = temp;
}

int main(){
      int a = 5;
      int b = 3;
      swap (&a,&b);
      printf("a=%d\n", a);
      printf("b=%d\n",b);
      return 0;
}
```

Program to exchange 2 value of variables



Structure

- Structure is a collection of variables under a single name. Variables can be of any type: int, float, char etc.
- ***Declaring a Structure:***



Using variable structure

- **How to declare Structure Variable?**
  - This is similar to variable declaration.
- Example :

```
int a;
struct Customer John;
```

Access structure members

- Use “dot” operator denoted by (.).
- Syntax:

structure-variable-name.member-name

Ex:

John.salary;

John.commission;

Access structure members (cont)

- Access to members of a pointer to the variable structure → using operators (→)
- Example :
  - struct student b = {70000000,70};
    struct student *c = &b;
    printf("Score of student : \n", c->score);

Example (Structure)

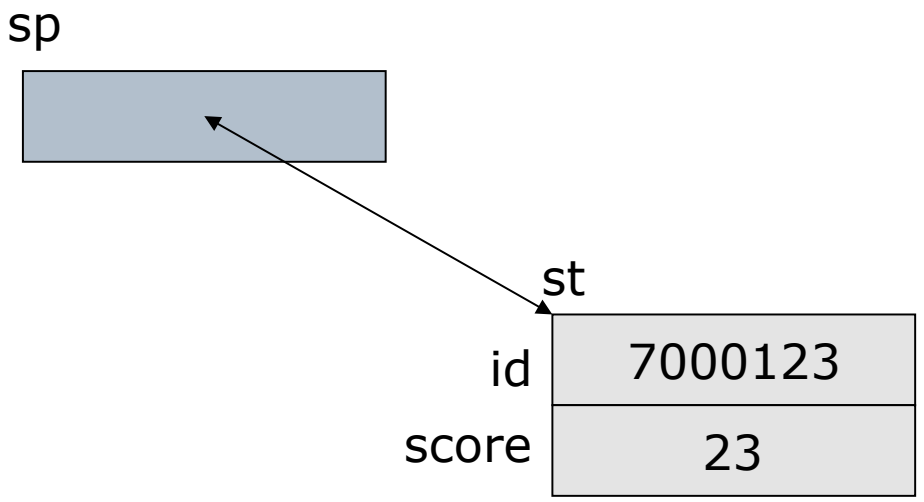
```
struct student{
    int id;
    int score;
};

int main()
{
    int i;
    struct student students[5];
    for(i=0; i<5; i++){
        students[i].id = i;
        students[i].score = i;
    }
    for(i=0;i<5;i++){
        printf("student id:%d, score:%d\n",
    students[i].id, students[i].score);
    }
}
```

Use ‘typedef’

```
typedef struct student{
  int id;
  int score;
} STUDENT;
STUDENT students[5];
```

Structure and Pointer

```
struct student st;
struct student *sp;
sp = &st;
sp->id = 7000123;
(*sp).score = 23;
```



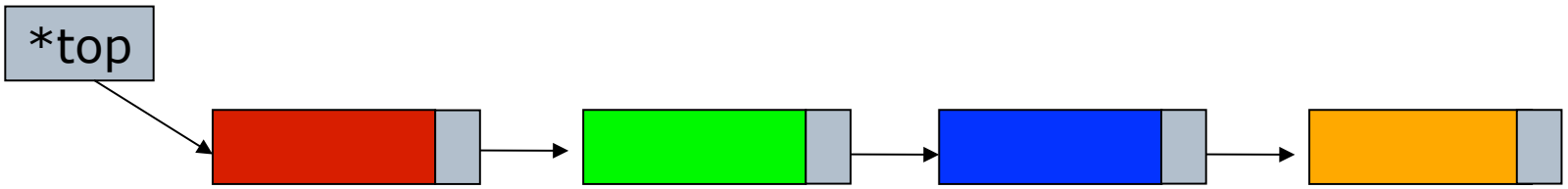
printf("%d\n", sp->score);

Link list

- Store a pointer to the next structure in the structure

```
struct student {
   int id;
   int score;
   struct student *next;
}
```

- *Warning : allocate memory before use and release memory after use*

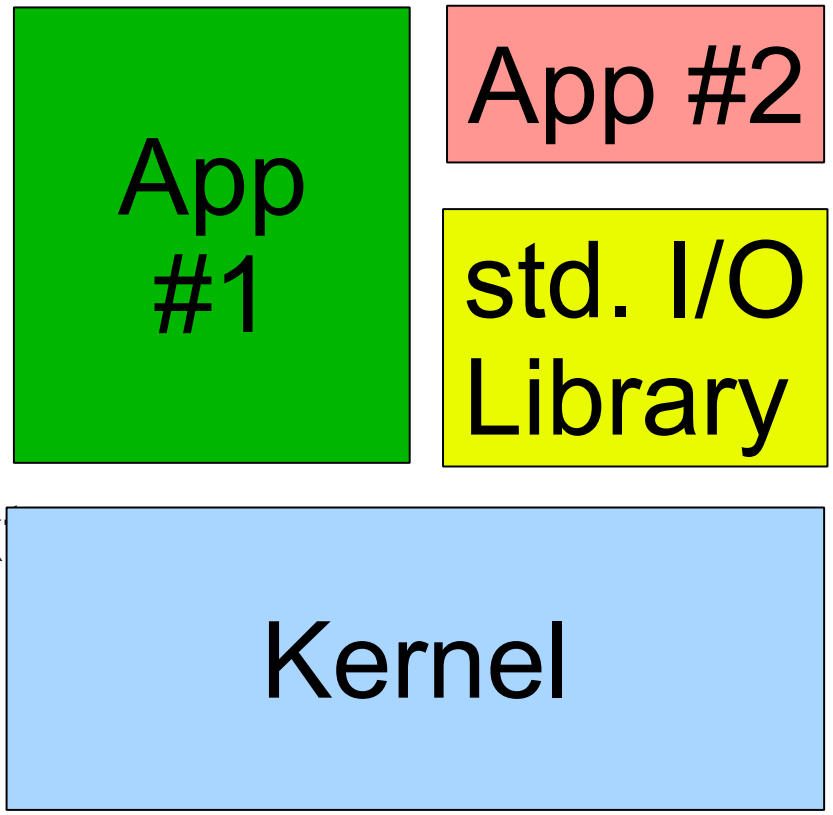


Link list (cont)

```
char *cp;
struct student *sp;

（1）
cp = (char *)malloc(64);
sp = (struct student *)malloc(64);

（2）
cp = (char *)malloc(sizeof(ch));
sp = (struct student *)malloc(sizeof(struct student)*10);
  → struct student sp[10]
```

I/O function

- All I/O calls ultimately go to the kernel
- I/O library helps with buffering, formatting, interpreting (esp. tex strings & conversions)



Input function (include in stdio.h)

- Functions
  - printf( )
    - Print formatted data to stdout
  - fprintf( )
    - Write formatted output to stream
  - gets( )
    - Read one line from standard input
    - Get warning by compilers
  - fgets( )
    - Get string from stream, a newline character makes fgets stop reading
    - **USE THIS INSTEAD of gets()**
  - getc( )
    - Character read from standard input
  - putc( )
    - Export one character to standard output
- Deprecated functions
  - scanf( )
    - Read formatted data from stdin
  - fscanf( )
    - Read formatted data from stream

File handling functions

- FILE * fopen(char *filename, char *mode)
  - r,w,a,r+,w+,a+
- char * fgets(char *s,int length,FILE *fd)
- int fgetc(FILE *fd)
- fclose(FILE *fd)

Example

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  FILE *fp;
  char buf[1024];
  int c;
  fp = fopen(argv[1],"r");
     while((fgets(buf, sizeof(buf),fp)) != NULL){
                    fputs(buf,stdout);
    }
  fclose(fp);
  exit(0);
}
```

Exercise

We need a small study schedule management program for a single student. The program read the list of registered course and its schedule from a text file. The structure of the file is as following:

```
IT3080   Computer Network        1,523,526,22,25-31,33-40,TC-502;
IT4560   Computer Literacy       1,221,224,22,25-31,33-40,TC-211;
IT4590   Database                1,524,526,22,25-31,33-40,D6-101;
IT4935   Database Lab            1,615,616,22,25-31,D6-303;
```

Required functionalities:

- Read study schedule from file, represent the information under the form of a list of structures, each structure instance corresponds to a course.
- Display the schedule under the form of a table on terminal

```
=====================================================
Code      |Course              | Week Day| AM/PM  |Period  |Week                  | Room
IT3080    |Computer Network | Thursday| Afternoon| 3-6    |22,25-31, 33-40       |TC-502
```

Exercise

- Display the busy schedule in the following format
- ==========
  - |Monday | Tuesday |Wednesday |Thursday |Friday
- ----------
- 1 | | | | |
- 2 | | | | |
- 3 | | | | |
- 4 | | | | |
- 5| | | | |
- 6 | | | | |
- 7 | TC-201 | | | |
- 8 | TC-201| | | |
- 9 | TC-201 | | |TC-502|
- 10 | TC-201 | | |TC-502|
- 11 | | | |TC-502|
- 12 | | | |TC-502|