Review C Programming

Bùi Trọng Tùng
Viện CNTT-TT, Đại học BKHN

Content

- Data type
- Condition and Iteration
- 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
  - Declare : int a[10];

Size of Type

size of char: 1 bytes

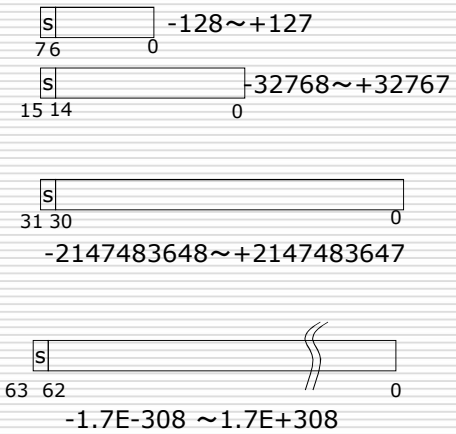
size of short: 2 bytes

size of int: 4 bytes

size of long: 4 bytes

size of float: 4 bytes

size of double: 8 bytes



Condition and Loop Structure

- 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){
        statement 1  ;
        …
}
else{
        statement 2  ;
        …
}
```

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( condition 1  ;     condition 2  ;     condition 3  ){
    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 next iteration of 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. The following is its format:

  type name (parameter1, parameter2, ...) { statements }
- where:
  - Type is the data type specifier of the data returned by the function.
  - Name is the identifier by which it will be possible to call the function.
  - Parameters (as many as needed): Each parameter consists of a data type specifier followed by an identifier
  - Statements is the function's body. It is a block of statements surrounded by braces { }.

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] : argument 0
- argv[1] : argument 1
- argv[2] : argument 2

Example :

%./a.out 123 456 789

arg[0]: ./a.out

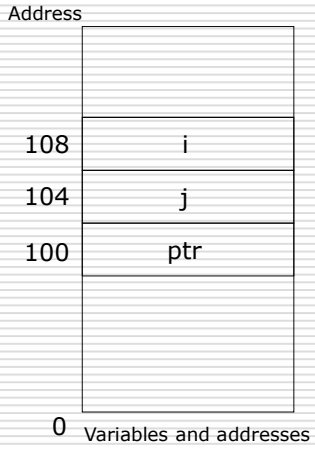
arg[1]: 123

arg[2]: 456

arg[3]: 789

Pointer

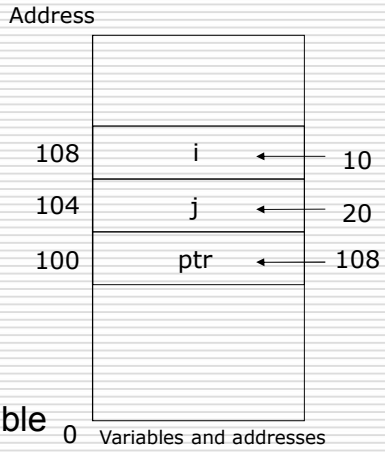
- Pointer variable
  - "Variable" refers to variable
- int i = 10;
- int j = 20;
- int *ptr



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] */

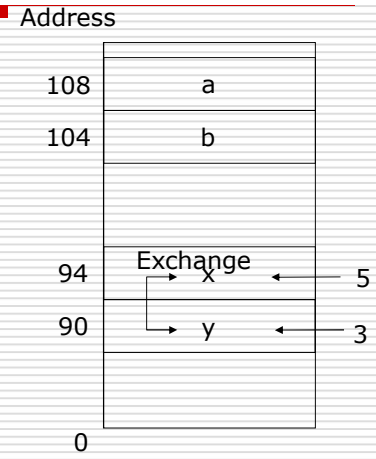
Pointer and function

```
#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)

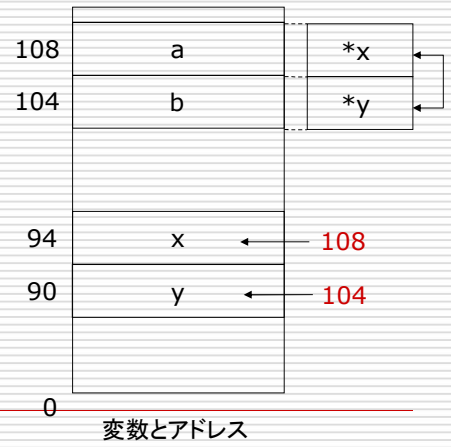
```
#include <stdio.h>
void swap(int x, int y)
{
int temp;
temp = x;
x = y;
y = temp;
}

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)

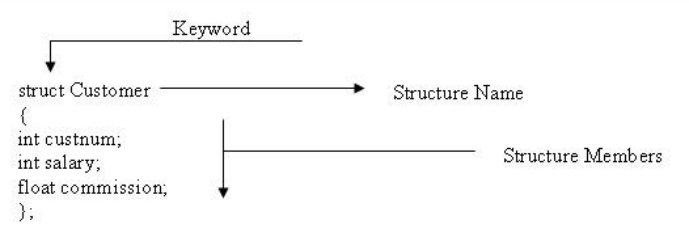
```
#include <stdio.h>
void swap(int *x, int *y)
{
int temp;
temp = *x;
*x = *y;
*y = temp;
}

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

- the operator used is the dot operator denoted by (.). The dot operator for accessing structure members is used thusly:

  structure variable name.member name

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)

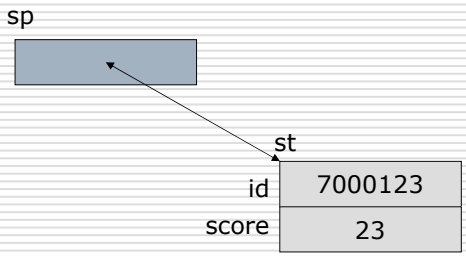
```
struct student{
    int id;
    int score;
};

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;
```



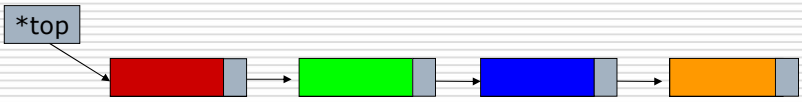

```
printf("%d\n", sp->score);
```

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*

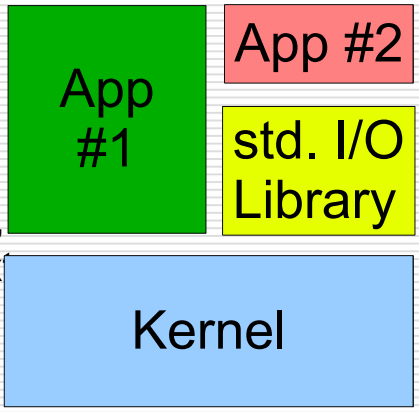


Link list (cont)

```
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. text 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
    - **NEVER EVER USE THIS!**
  - fgets( )
    - Get string from stream, a newline character makes fgets stop reading
    - **USE THIS INSTEAD**
  - 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

Input function (include in unistd.h)

- Function
  - read()
    - Argument : number of bytes read and target
  - write()
    - Argument : the number of bytes to write to output
  - open()
  - close()
  - start()

open()/read()/write()/close()

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#define BUFSIZE 1024
int main()
{
        char buf[BUFSIZE];
        int fd;
        int nbyte;
        fd = open("test.txt", O_RDONLY, 0);
        while((nbyte = read(fd, buf, BUFSIZE)) > 0) {
                write(1, buf, nbyte);
        }
        close(fd);
        exit(0);
}
```

File handling functions

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

Example

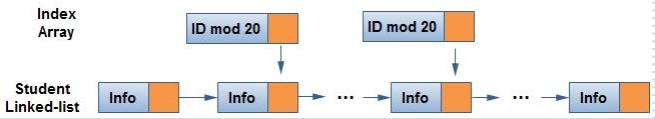
```
#include <stdio.h>
int main(int argc, char *argv[])
{
  FILE *fp;
  char buf[1024];
  int c;
  fp = fopen(argv[1],"r");
     while((fgets(buf, sizeof(buf),fp)) != NULL){
                    fputs(buf,stdout);
    }
  fclose(fp);
  exit(0);
}
```

Practice

I. Xây dựng chương trình tính giá trị biểu thức

1. Cho phép người dùng nhập từ bàn phím hàm f(x) và giá trị biến x.
2. Biểu diễn biểu thức dưới dạng cây. Tính giá trị biểu thức
3. Biểu diễn biểu thức dưới dạng ký pháp Ba Lan. Tính giá trị biểu thức.

Practice



II. Chương trình quản lý thông tin SV

Cơ sở dữ liệu về sinh viên được mô tả như sau:

- Các thông tin về sinh viên bao gồm:
  - StudentID : integer
  - Name : char[30]
  - Class : char[10]
  - Department : char[10]
- CSDL quản lý SV bao gồm 1 danh sách liên kết và một mảng con trỏ. Trong đó
  - Thông tin của mỗi SV được lưu trong một nút của danh sách liên kết Student Linked-list
  - Mảng Index Array được sử dụng để việc tìm kiếm nhanh và thuận lợi hơn. Mỗi nút của mảng liên kết gồm trường :
    - (1) Trường 1 : ghi nhận giá trị R = 0 ÷ 19
    - (2) Trường 2 : trỏ vào phần tử đầu tiên trong danh sách thỏa mãn StudentID mod 20 = R

Practice

II. Chương trình quản lý thông tin SV (tiếp)

Xây dựng chương trình quản lý thông tin SV thực hiện những công việc sau:

1. Đưa dữ liệu từ một file văn bản students.txt vào Student Linked-list

* Mỗi dòng trên file văn bản chứa các thông tin của SV, mỗi trường cách nhau bởi dấu ":"

2. Cho phép người dùng thêm, xóa, tìm kiếm theo các tiêu chí tùy ý, sửa thông tin. Cập nhật sự thay đổi vào file studens.txt và ghi nhận thao tác đó vào file log.

* File log là một file văn bản, mỗi dòng chứa các thông tin sau:

- Thời điểm thay đổi
- Tên thao tác : delete, append, modify
- Mã số SV bị thay đổi thông tin.