Internal Memory:

ROM (read-only memory) permanent

RAM (random access memory) volatile(不定的)

Secondary Storage:

permanent

Magnetic disk/CD-ROMs/Magnetic tapes

I/O Units:

The interface(接口) that peripheral(外部设备) devices(keyboards/console screen/printers) are attached to

Mainframe:

Mainboard/CPU/Internal Memory/External Storage/Extended Cards/Interfaces/Power

Software:

== Programs + Data + Documents

Computer Environment:

Time-sharing Environment

Client/Server Environment

Distributed Computing

Webcast teaching environment is

Data:

Bit 0/1

Byte = 8 Bit (1B = 8 bits)

Word: sequence of bytes as a single unit

Machine Language:

The only language understood by computer hardware

Opcode (operation code)

Types of Language:

Assembly Language: 汇编

Object-oriented language: Smalltalk C++ Java C#

Procedural language: C Fortran

Compiled language: C C++

Interpreted language: Java? JavaScript Python PHP MATLAB

High-Level language / Assembly language

Imperative language: C C++ Java Python Fortran Basic

**C**

Function:

Header: type name arguments

Cannot define a function inside a function

Indentation (缩进):

C compiler ignores all white spaces

Identifiers:

Keywords/Reserved words: predefined by C

Standard identifiers: names of functions provided by C library

Programmer-created identifiers: first character must be \_ or letter; blank spaces are not allowed

Escape Character (转义字符):

\a bell \b 退格 \n 换行 \r 回车(移动光标到行首) \t 制表 一个tab \\ 输出\ \’ 输出’ \” 输出” \? 输出? \0 NULL \x 十六进制字符

Data Types:

Precision: float 6digits; double 8; long double 10;

Operator Associativity:

Variable:

Case sensitive. C and c are different

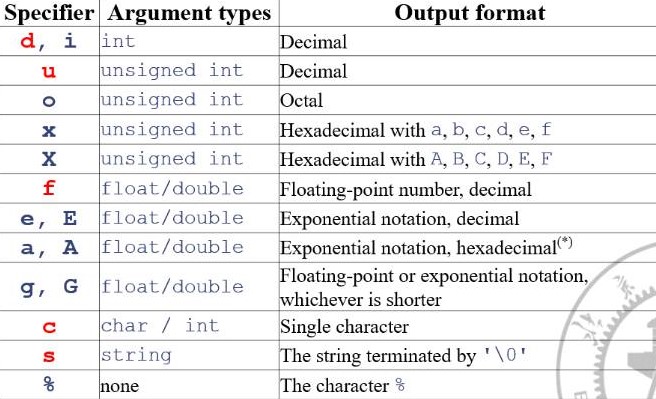
Assignment:

**Length = 25** // length is assigned the value 25

**Printf**

Conversion Specification:

Format: %[flags][field width][.precision]specifier

Specifiers: 

Exponential for 指数的

Flags: + prefixed to a positive integer

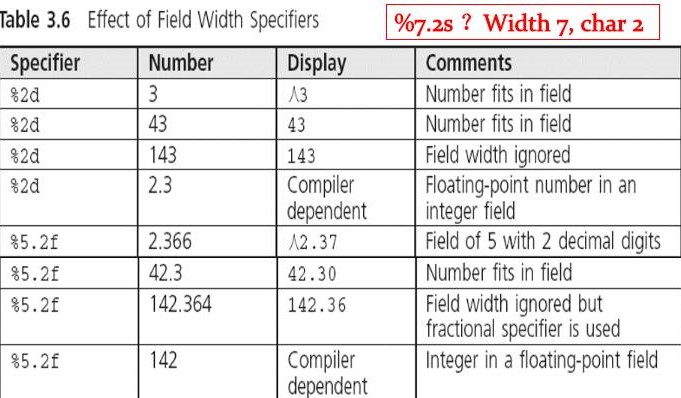
White space prefixed to a positive integer

- 左对齐

0 前导0 **%05d** ----> 00123

# specifier is A, a, E, e, G, g floating point number is formatted with a decimal point, is X, x, o, hexadecimal integers are formatted with 0X or 0x prefix, octal with 0 prefix

Width Specifiers:



%+-10d and %-+10d are the same

Scanf: returns the number of arguments it successfully reads. The end of file or read error occurs, EOF is returned.

**Relational Expression: also called simple conditions**

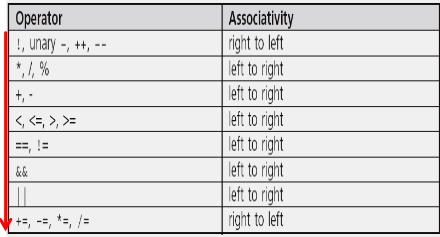
The outcome can only be 0 or 1

Char comparison are done through ASCII

Precedence: < <= > >= are higher than == !=

Lower than Arithmetic operators/Higher than Assignment and Logic operators

A relational expression returns TRUE or FALSE based on comparison of two individual values (either variables or constants). A Boolean expression returns TRUE or FALSE based on the outcome of two or more relational expressions.



**Associativity: 左结合相当于左边加(), 右结合相当于右边加()**

Logic Operators:

Short-circuit evaluation && and || make the evaluation stop as soon as it determined that an expression is true or false

when the first argument of the AND function evaluates to false, the overall value must be false; and when the first argument of the OR function evaluates to true, the overall value must be true.

**i=1 j=2 k=0; i||j&&k==1**

Compound Statement:

Also called a block. Does not need a semicolon

Ternary Operator:

The second and the third operands are the same type

Associativity: right to left **a ? b : c ? d : e** is **a ? b : (c ? d : e)**

Loop:

Basic structure: Initialization Condition Action Updating

Pretest: while for

Posttest: do-while

for loop: **for (int i=0; i<10; i++);** is a for loop with no body

do-while: body statements is executed at least once

**Functions:**

Define: A type must be listed explicitly for each parameter

Local variables and formal parameters are private for the function

Nested functions are not permitted

Can be distributed among different source files

The prototype: **int max (int, int, int,);** is permitted. **Do not forget the semicolon**

Declare and Define: declare prototype end with semicolon and provide all the information necessary to call the function successfully. Define should not end with a semicolon and followed with function body

Formal parameter: is a variable declared in the header of a function

Actual argument: is an expression appears in the parentheses of a function

Functions can directly return at most a single value

Storage Classes in C:

Local variables can be auto, register or static

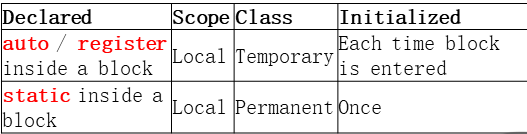
Global variables can be static or extern

Local:

Auto: the default storage class for local variables. Before return, alive; after return, die **Auto int a** equals **int a**

Register: the same life duration with auto. Switched to auto if the compiler does not support or when exceeded computer’s register capacity

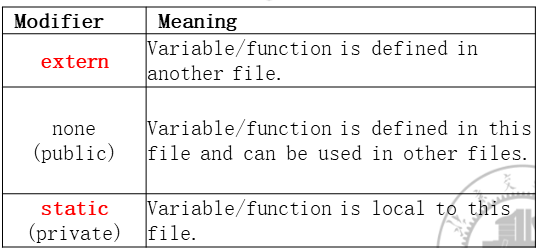
Static: once created, remain for the life of the program. Initialized only once, value is kept even when the function returns



Global:

Extern: extend the scope of global variable to current source file, do not create new storage area

Static: used to prevent the extension of a global variable into a second file



Function prototype: extern(default), can be accessed in other files(default), or static, within the file

Recursion:

Initial case (base case): is required

Address and Pointer

Pointer:

Variable that stores address

All pointers have the same size. But cannot be assigned to one another

Indirection operator: \*

Pointer variable cannot be assigned by constant

Avoid wild pointer: **int \*p; \*p=123;**

**{**

**int \*p;**

**p++; //指向下一个连续地址**

**\*p++; // 由于是后加加，因此先使用p的值，结束后指向下一个连续地址**

**\*(p++); // 与上相同**

**(\*p)++; // 先取值，再使值+1**

**}**

Array

Atomic variable, also called scalar variable

Stores the data all of the same data type

Each item in an array is an element

Array[pos] pos is the element’s index or subscript, starting by 0. An element is called an indexed variable or a subscripted variable

Array name is not a variable

Subscript can be any integer expression ----- array[i++]

Elements of an array are stores in contiguous locations

Initialization: char a[]={ ‘a’, ‘b’, ‘c’, 80 }; length is 4 automatically; int a[10]={1} first is 1

For all global and static arrays, elements are set to 0 by default. Auto arrays are undefined

Num[SIZE] is allowed when SIZE is macro (宏) #define

The NULL character ‘\0’ is appended to the end of strings

**char str[]=”hello"; str=”Hello”;** is invalid

Array as a formal parameter, its size is not required

Const qualifier in front of an array parameter, its value cannot be modified in the called function

2D array initialization is done in row order

String

gets() until enter is typed in

scanf() until a white space is typed in

puts() print the string and a ‘\n’

formatted print: printf(“|%25.12s|”,”Have a happy day”); outputs: | Have a happy|

strcpy(s1,s2) copy s2 to s1 strcmp(s1,s2) returns 1 if s1>s2; 0 if s1==s2; -1 if s1<s2

char str[]=”ABC ; char \*p=”ABC”; or char \*p; p=”ABC”; str is a constant; p is a variable

Files

Input file stream: used to receive data from a file into a program

Output file stream: used to send data from program to a file

Three streams are automatically opened when program execution begins:

Stdin: receives input from keyboard

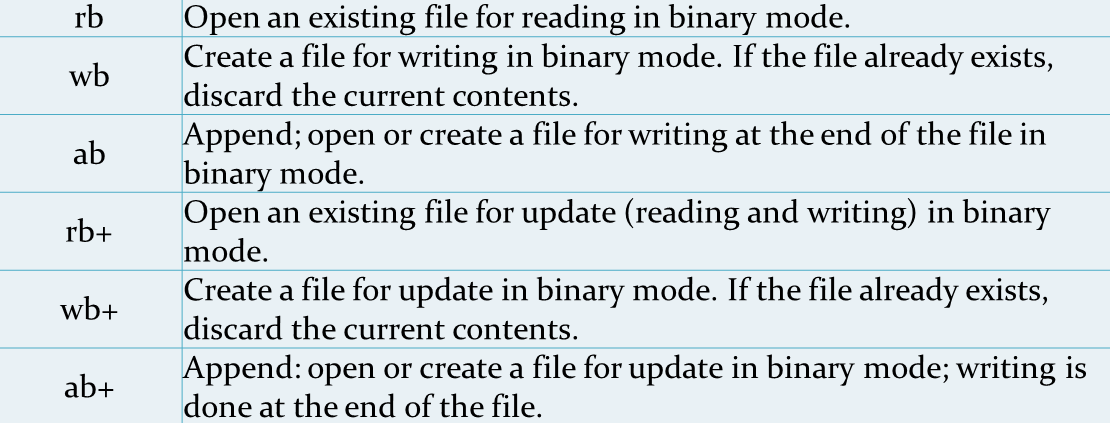
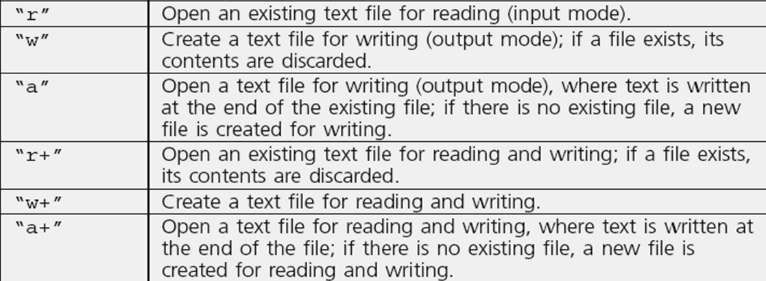
Stdout: displays output on the screen

Stderr: displays error messages on the screen

Declare a file stream: **FILE\* fp;**

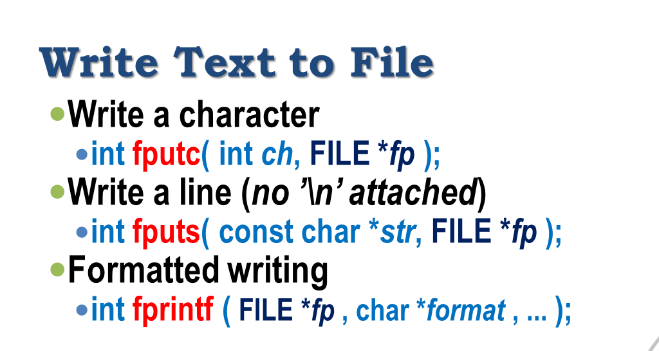
Open a file: **fp = fopen(“file.txt”, “r”);**  if error occurs, return NULL

Open modes:

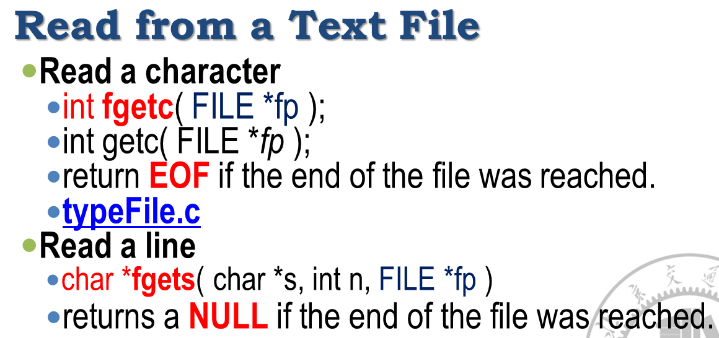


EOF: appended at the end of file; 0xFFFF; not a part of file

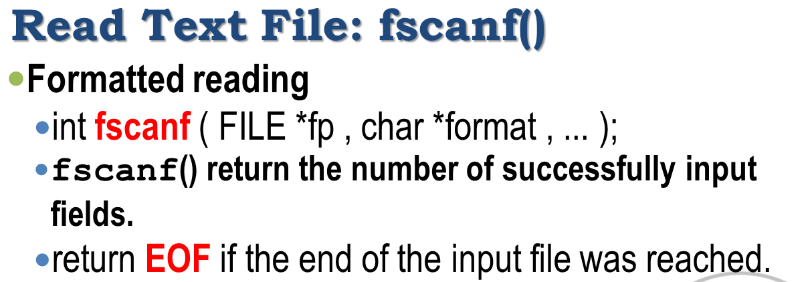
Write text to files:



Read from a text file:

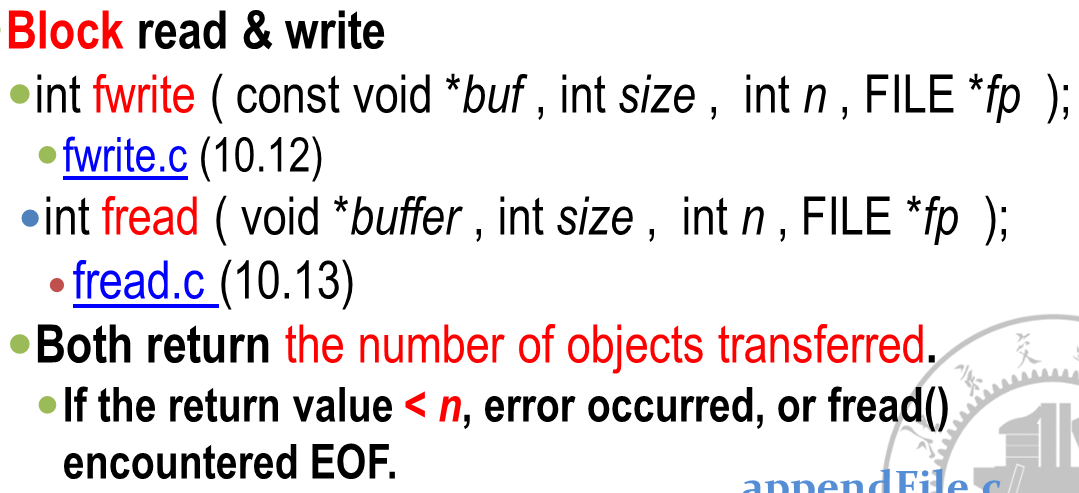


fscanf:

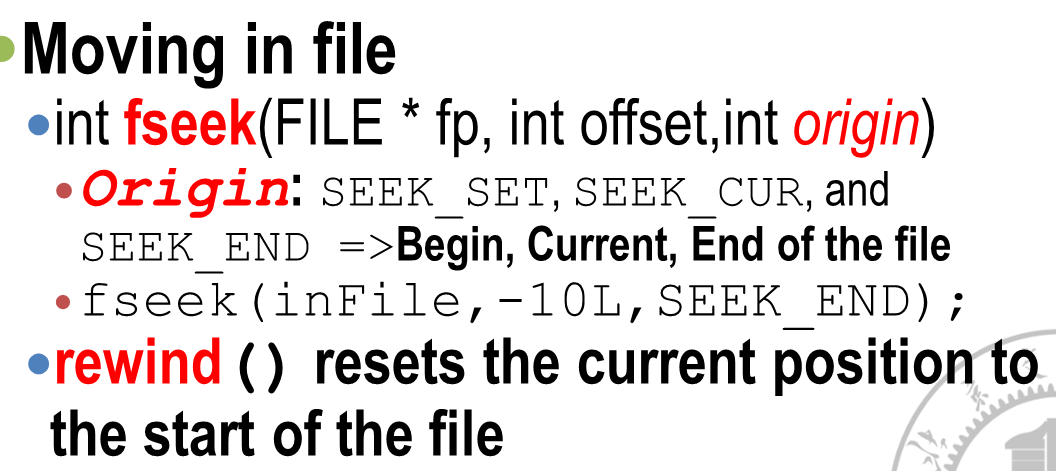


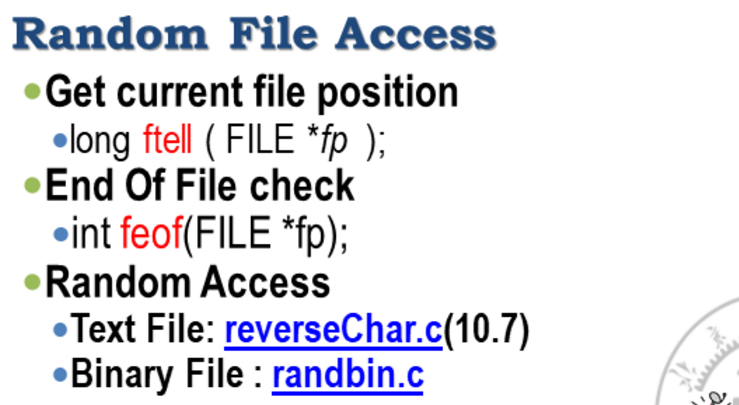
Read and write from binary files:

fgetc read a byte; fputc write a byte

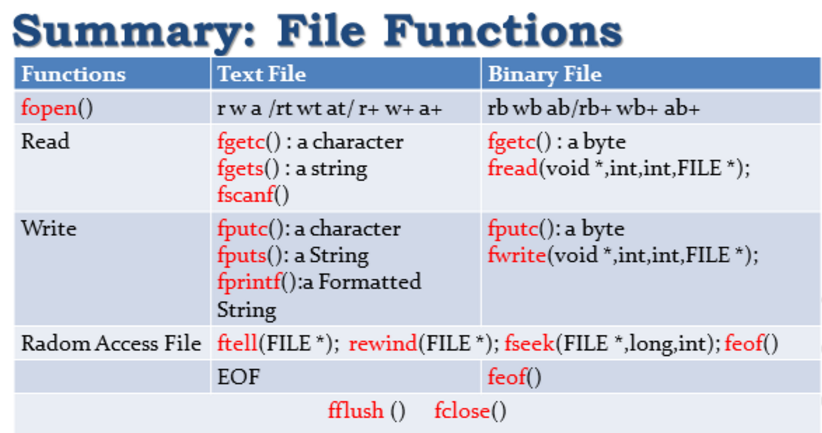


Random access file:

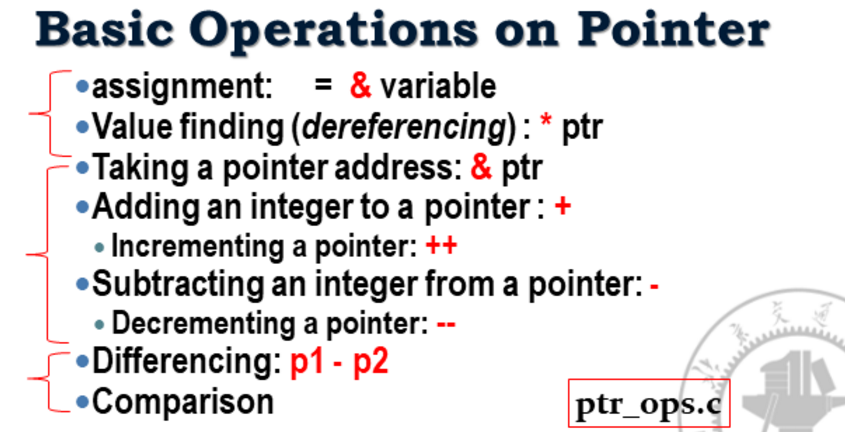




如果是文件末尾返回1，否则返回0



Pointer:



A pointer’s value: an address; Can be initialized assigned passed if have same type

Can subtract: p1-p2

Can be compared using relational operators

Value finding (dereferencing): \*p

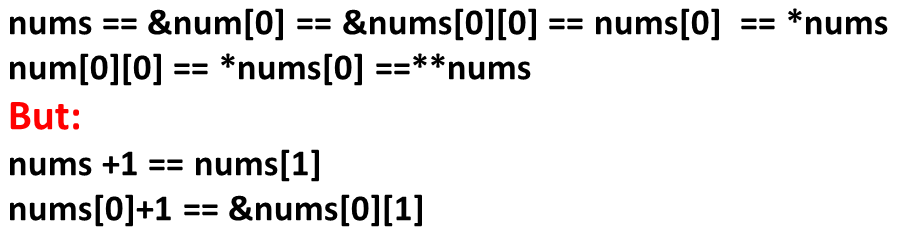
Generic pointer: void\* ; can represent any pointer type; cannot be dereferenced

Array name as pointer: \*grade grade[0]; \*(grade+1) grade[1];

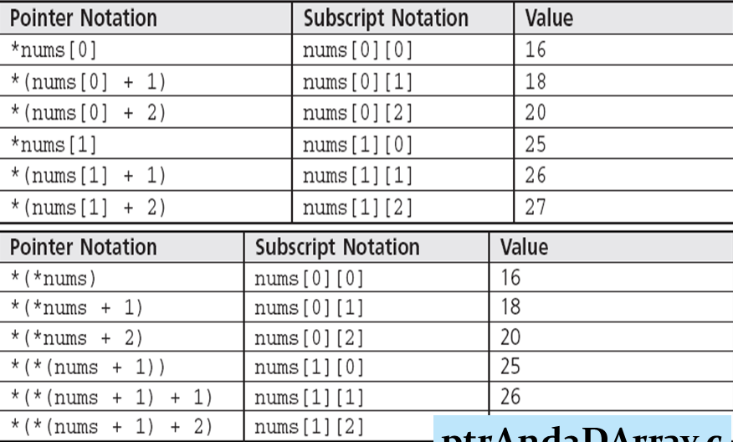
An array name is an **address constant**

Uninitialized pointers: do not dereference a wild pointer // error

2D array and pointer:



Advanced pointer notation:



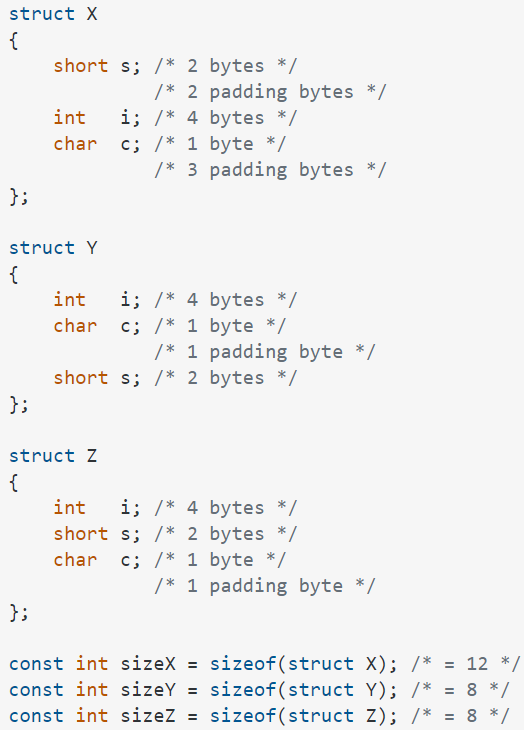
Pointer of arrays: **int (\*p) [2];** // Array of pointers: **int \*p[2];**

Structure

Structure member operator: . highest precedence

Populate the structure: assigning values to the members of a structure

Structure alignment:



Structure variable can be assigned by another struct; **cannot be compared in any relational expressions**

Initialization: **struct Date birth = {5, 15, 2002} ;** if fewer initializers, remains are initialized to 0 or NULL

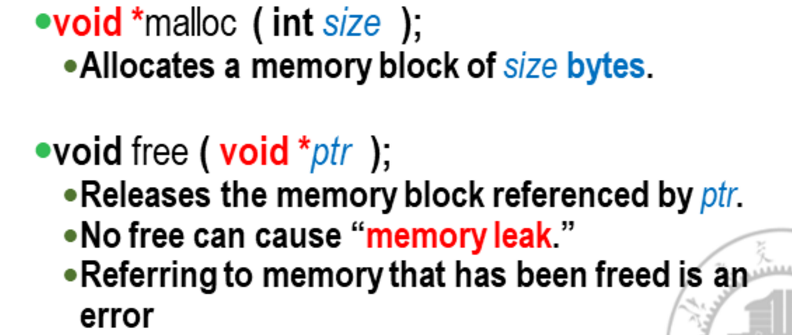
Pointer of structure: struct student \*p; **(\*p).num** or **p->num**

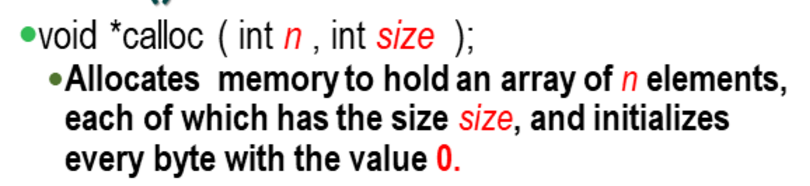
() . [] -> highest precedence

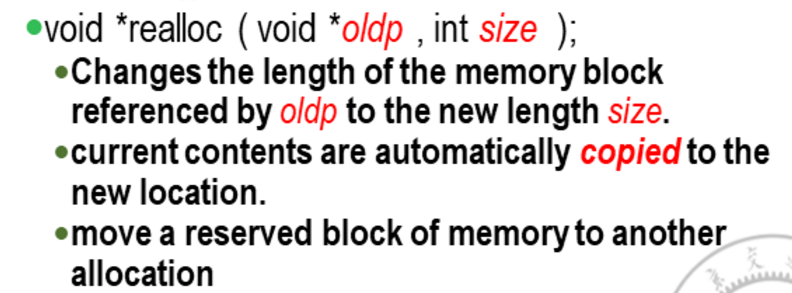
Dynamic data structures;

Linked list: self-reference: struct xxx {struct xxx \*next;}

Memory allocation: **void \*malloc(int size); void \*calloc(int n, int size);//initialized to 0 void \*realloc(void \*p, int n);** returns the beginning address of a memory block, general pointer, can be casted to any pointer; return NULL if allocation failed







LIFO structure stack: push and pop

FIFO structure queue: enqueue and dequeue

Words: suffix 后缀; entity 独立实体; attribute 属性，归因于; retrieve 找回

Code

#include <stdio.h>

void swap(int \*a, int \*b)

{

int temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

void BubbleSort(int \*a, int n)

{

for(int i=0;i<n-1;i++)

for(int j=0;j<n-1-i;j++)

if(a[j]>a[j+1])swap(&a[j],&a[j+1]);

}

void SelectionSort(int \*a,int n)

{

for(int i=0;i<n-1;i++)

for(int j=i+1;j<n;j++)

if(a[i]>a[j])swap(&a[i],&a[j]);

}

int main()

{

int a[10]={2,3,7,5,6,8,8,4,1,10};

int b[10]={24,32,76,54,54,97,81,43,14,10};

BubbleSort(a,10);

SelectionSort(b,10);

return 0;

}

//stack

struct NameRec \* push(struct NameRec \*top, char \*name) {

struct NameRec \*newNode;

newNode = (struct NameRec \*)

malloc(sizeof(struct NameRec));

if (newNode == NULL ) {

printf("\n内存分配失败！\n");

exit(1);

}

strcpy(newNode->name, name); /\* 保存数据内容 \*/

newNode->next = top; /\* 插入到栈顶 \*/

top = newNode; /\* 更新栈顶指针为新元素地址 \*/

return top;

}

出栈操作

将栈顶元素的数据保存到指定参数，从栈中删除该栈顶元素，并设置新的栈顶元素。

返回栈顶指针。

struct NameRec \*pop(struct NameRec \*top, char \*name) {

struct NameRec \*temp

if (top != NULL ) {

strcpy(name, top->name); //获取栈顶数据

temp = top->next; //保存新的栈顶地址

free(top); //释放栈顶结点

top = temp; //移动栈顶到下一个结点

}

return top;

}

// 连续获取人名，调用入栈操作插入栈顶

// 输入 x/X 时结束

struct NameRec \*readPush(struct NameRec \*top) {

char name[MAXCHARS]

printf("\n输入名字（直接回车结束）\n");

while (1) {

printf("输入名字: ");

gets(name);

if (strlen(name) == 0)

break;

top = push(top, name);

}

return top;

}

//调用出栈操作，取出栈中所有元素，输出元素数据

void popShow(struct NameRec \*top) {

char name[MAXCHARS];

printf("\n栈中的姓名有:\n");

while (top != NULL ) /\* 输出栈中所有数据 \*/

{

top = pop(top, name);

printf("%s\n", name);

}

printf("\n栈中没有数据了。\n");

}

Queue

/\* 入队和出队指针 \*/

struct NameRec \*tailer, \*header;

入队操作

动态申请新结点，将指定的数据保存在结点中，并插入队尾。

void enque(char \*name) {

struct NameRec \*newNode;

newNode = (struct NameRec \*)

malloc(sizeof(struct NameRec));

if (newNode == NULL ) {

printf("\n内存分配失败！\n");

exit(1);

}

strcpy(newNode->name, name); //保存数据

newNode->next = NULL;

if (header == NULL ){

header = newNode; //空队列，队列的第一个结点

//tail = newNode;

} else

tailer->next = newNode; //非空队列，添加到队尾

tailer = newNode; //更新队尾指针

}

出队操作

将队首元素的数据保存到指定参数，删除队首元素，并设置新的队首指针。

void serve(char \*name) {

struct NameRec \*temp;

strcpy(name, header->name); //获取队首结点数据

temp = header->next; //删除队首结点

free(header);

header = temp; //队首移到下一个结点

}

连续获取人名，调用入队操作插入队尾。输入x时结束

void readEnque() {

char name[MAXCHARS];

void enque(char \*);

printf("\n输入名字（输入字母 x 结束）\n");

while (1) {

printf("输入名字: ");

gets(name);

if (strcmp(name, "x") == 0)

break;

enque(name);

}

}

调用出队操作，取出队列中所有元素，输出元素数据

void serveShow() {

char name[MAXCHARS];

printf("\n队中的姓名有:\n");

while (header != NULL ) {

serve(name);

printf("%s\n", name);

}

}

Dynamic linked list

/\*

定义创建链表函数createLinkedList

创建无序链表，以输入顺序创建链表，新结点插入在链表末尾

返回链表的头指针

输入 0 0 结束

\*/

struct student \*creatLinkedList() {

//头指针

struct student \*head = NULL;

//临时指针变量

//pCur为当前结点地址，pEnd保留链表末尾结点地址

struct student \*pCur, \*pEnd;

//结点数初始值

//numberOfNodes = 0;

//申请第一个结点的内存

pCur = (struct student\*) malloc(LEN);

//获取第一个结点的数据

//提示

puts("请输入学号（整数）和成绩（可以有小数点）：");

//获取输入

scanf("%d%f", &pCur->num, &pCur->score);

//循环，当学号>0时，为有效结点，链入链表中最后位置

while (pCur->num > 0) {

if (head == NULL ){ //pCur为第一个节点

head = pCur;

} else { //不是第一个节点，新结点成为末尾结点

pEnd->next = pCur;

}

//保留当前链表末尾结点

pEnd = pCur;

//当前新结点

pCur = (struct student\*) malloc(LEN);

//获取新结点的数据

puts("请输入学号（整数）和成绩（可以有小数点）：");

scanf("%d%f", &pCur->num, &pCur->score);

}

pEnd->next = NULL;

//返回头指针

return head;

}

/\*

输出链表函数 outputLinkedList()

首先确定链表第一个结点，也就是head的值。

然后设一个指针变量p,先指向第一个结点，输出ｐ所指的结点

然后使ｐ后移一个结点，再输出，直到链表的尾结点。

\*/

void outputLinkedList(struct student \*head) {

struct student \*p;

int count = 0;

//指向头指针

p = head;

while (p != NULL ) {

printf("学号：%d 成绩：%5.1f\n", p->num, p->score);

//移动到下一个结点

p = p->next;

count++;

}

printf("共有 %d 条记录\n\n", count);

}

/\*

查找链表结点

在一个动态链表中，依据指定的某项数据域值或指定的值范围，

查找符合条件的结点。

查找链表结点的算法和输出链表类似，

都是遍历链表的所有结点。

函数的参数为链表头指针和指定的学号。

返回查找到结点的地址，如果没有找到，返回NULL。

\*/

struct student \*findNode(struct student \*head, int num) {

struct student \*p = head;

if (p == NULL ) //空链表

{

printf("空链表!\n");

return NULL ;

}

//查找指定学号的结点

while (p != NULL && num != p->num) {

p = p->next;

}

return p;

}

/\*

删除结点； deleteNode()

返回删除指定结点后的链表

从p指向的第一个结点开始，查找学号为num值的结点。

如果找到就将该结点删除；

如未找到，就将p后移一个结点；

如此进行下去，直到遇到表尾为止。

还需要考虑链表是空表（无结点）和

链表中找不到要删除的结点的情况

\*/

struct student \*deleteNode(struct student \*head, int num) {

//pCur为当前结点地址，pPre为前驱结点地址

struct student \*pCur, \*pPre;

//指向第一个结点

pCur = head;

if (pCur == NULL ) //空链表

{

printf("空链表!\n");

return NULL ;

}

//查找指定学号的结点：可以调用findNode() ?

while (pCur != NULL && num != pCur->num) {

pPre = pCur;

pCur = pCur->next;

}

if (pCur != NULL ) //找到结点

{

printf("删除:%d\n", num);

if (pCur == head) { //删除第一个结点

head = pCur->next;

} else { //删除当前结点

pPre->next = pCur->next;

}

free(pCur); //释放当前结点内存

} else {

printf("学号 %d 不存在!\n", num);

}

return head;

}

/\*

向升序链表插入指定的结点 insertNode()

参数head为链表头指针；

参数pNode指针指向将要插入的结点；

返回修改后的链表头指针。

本例按学号升序排序

\*/

struct student \*insert(struct student \*head, struct student \*pNode) {

// pPrev为前驱结点位置，pNext为后继结点位置

struct student \*pPrev, \*pNext;

pNext = head;

if (head == NULL ) //空链表

{

head = pNode; //插入到第一个位置

pNode->next = NULL;

} else //非空链表

{

/\*

查找插入位置。

将pNode->num与pNext->num相比较，

如果pNode->num ＞ pNext-> num ，

则待插入的结点应插在pNext所指的结点之后的某个位置，

此时将pNext后移，并使pPrev指向刚才pNext所指的结点.

\*/

while ((pNext != NULL )&&(pNode->num > pNext->num)){

pPrev = pNext;

pNext = pNext->next;

}

if (pNext == NULL) { //未找到插入位置

pPrev->next = pNode; //插入到链表最后位置

pNode->next = NULL;

}else{

if(head == pNext) //第一个位置

head = pNode;

else//其他位置

pPrev->next = pNode;

pNode->next = pNext;

}

}

return head;

}

/\*

定义创建有序链表函数createSortedLinkedList()

创建升序链表，返回链表的头指针

本例按学号升序排序

输入 0 0 结束

\*/

struct student \*creatSortedLinkedList() {

struct student \*sHead = NULL;

//分配结点空间

struct student \*p = (struct student\*) malloc(LEN);

//获取输入

puts("请输入学号（整数）和成绩（可以有小数点）：");

scanf("%d%f", &p->num, &p->score);

//循环，当学号>0时，为有效结点，插入链表中适当位置

while (p->num > 0) {

//插入到链表

printf("\n插入结点 %d！\n", p->num);

sHead = insert(sHead, p);

//分配新结点空间

p = (struct student\*) malloc(LEN);

//输入结点的数据

puts("请输入学号（整数）和成绩（可以有小数点）：");

scanf("%d%f", &p->num, &p->score);

}

return sHead;

}

/\*

对无序链表排序

依次将无序链表中的节点插入到新创建的升序链表

本例按学号升序排序

返回链表的头指针

\*/

struct student \*sortLinkedList(struct student \*head) {

struct student \*sHead = NULL;

struct student \*p = head;

struct student \*pNext;

while (p != NULL ) {

//暂存下一个节点的地址

pNext = p->next;

//将p指向的当前结点插入到新链表

printf("\n插入结点 %d！\n", p->num);

sHead = insert(sHead, p);

//新的当前节点

p = pNext;

}

//返回排序后新链表的头指针

return sHead;

}

判断闰年:

If( ( year%4==0 && year%100!=0) || year%400==0 )

Switch语句:

switch(choice){

case 1: xxx; break;

case 2: xxx; break;

default: break;}

计算星期几:

if(month==1 || month==2){month+=12;year--;}

weekday= (day+2\*month+3\*(month+1)/5+year+year/4-year/100+year/400) % 7;

随机数:

int seed = time(NULL);

srand((unsigned int)seed);

for(int i=1;i<=10;i++)printf("%d\n",rand()%10);

二分查找：

int l=1,r=10,mid;

while(l<=r)

{

mid=(l+r)/2;

if(a[mid]==k){ans=mid;break;}

if(a[mid]>k){r=mid-1;}

else l=mid+1;

}