C Programming Language (2)

Pointers

- Pointers are derived data types from basic ones. A variable of a pointer data type store the address of the variable of the basic data type.

- Two operators are involved for pointer: & and *

```
int  i=5,  j, * pi;
pi = &i;
j = *pi;
```

Types of expression

- Arithmetic expression : An expression that involves arithmetic operators: binary operator + - * / %
  unary operator ++ --

- Relational or Logical expression : An expression that involves relational (>, <, >=, <=, ==, !=) or logical (&&, ||, !) operators.

- C language doesn't provide a logical data type.
  Result of Relational or Logical expression is an integer: **1** (means TRUE) or **0** (means FALSE)

Comparison Expression

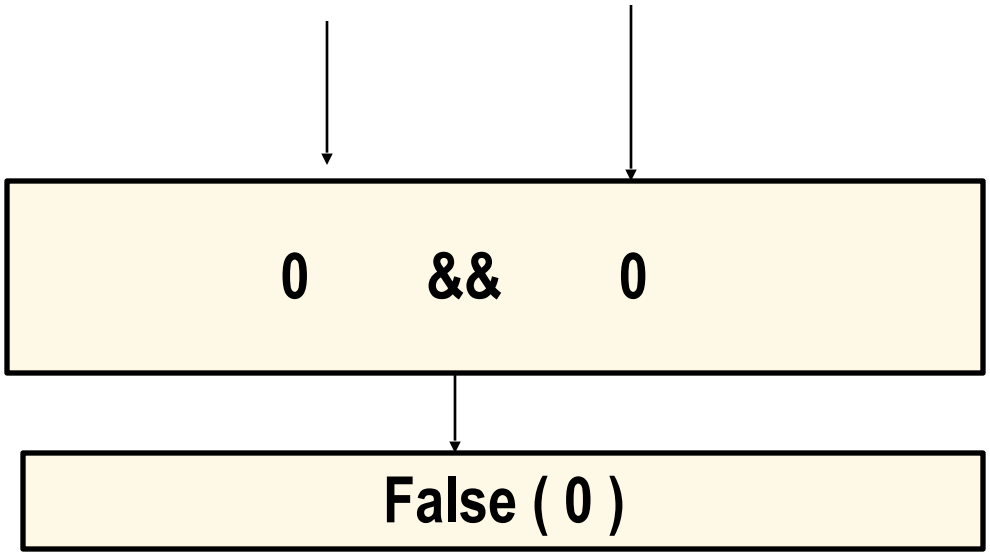
```
int i=1,   j=2,   k=3;
```

Expression	Value	Interpretation
i < j	1	true
(i + j) >= k	1	true
(j + k) > (i + 5)	0	false
K != 3	0	false
j == 2	1	true

Logic AND &&

- The result of AND operation is **TRUE** only if BOTH the operands are true.

Example: **(8 < 7) && (6 > 7)**



Logical OR ||

- The result of a logical OR operation will be true if either operand is true or if both operands are true.

```
(8 < 7)  ||  (6 > 7)

(8 > 7)  ||  (6 > 7)

(8 > 7)  ||  (6 < 7)
```

Value	Interpretation
0	false
1	true
1	true

Logical NOT !

- The Logical NOT is a unary operator. It negates the value of the operand.

- If value of X = 0 ! X = ?
  ! X = 1 -> true

- ! (5 > 3) = ??
  !(1) -> 0 -> false

- ! (5 < 6) || (7 > 7) = ???
  ! (1) || (0) = ! 1 = 0 -> false

USACO Code Format

```
/*
ID: zjuzzc1
LANG: C
TASK: ride
*/
#include <stdio.h>
int  main( )
{
        FILE *fin,  *fout;
        fin   = fopen(“ride.in", "r");
        fout = fopen(“ride.out", "w");
        while( fscanf( fin, "%c", &c ) > 0 ) {
                fprintf( fout, "%c", c);
        }
        fclose( fin );
        fclose( fout );
        return 0;
}
```

Line 1:
Line 2:
Line 3:
Line 4:
Line 5:
Line 6:
Line 7:
Line 8:
Line 9:
Line 10:
Line 12:
Line 13:
Line 14:

Comparison

- **printf/scanf**:
  int printf(fmt, …)
  int scanf(fmt, …)

- Loop **for**:

```
for( i=1; i<=9; i++ ) {
        statements;
}

i=1;
while( i<=9 ) {
        statements;
        i++;
}
```

- **fprintf/fscanf**:
  int fprintf(FILE *, fmt, …)
  int fscanf(FILE *, fmt, …)

- Loop **while**:

```
while( expression ) {
        statements;
}

do {
        statements;
} while ( expression ) ;
```

Arrays

- Basic data types and derived data types pointers:

  char c, *ptr_c = &c; int age, *ptr_i = &age;

- Another derived data types arrays:

  char string[10]; //character array with 10 elements

  int age[4] = {3, 5, 4, 6}; // with initialization

  float weight[ ] = {120.0, 90.7, 88.6, 104.4, 288.9};

- Array element index starting from 0:

  string[0]='A'; weight[4]=0.1; age[4] is not available

  int myage = age[2];

- Array name without [ ] is of pointer data type.

  int * ptr_i = age; char * ptr_c = string;

they are constant addresses, cannot be changed.

Strings

- C language doesn't have a particular data type for string, rather, we treat character arrays as strings

```
char str1[ ] = "hello";  //ending mark '\0' automatically
                         //added. So size of array is 6
char str1[ ] = { 'h', 'e', 'l', 'l', 'o', '\0' };  //this is equivalent
```

- You can think of str1 as a character array with size of 6, or you can treat it as a string of length 5

```
printf("string str1 is %s", str1);   // output: hello
printf("size of array str1 is %d", sizeof(str1));   // output 6
printf("length of string str1 is %d",strlen(str1));   // output 5
```

ASCII codes

0	nul
1	soh
2	stx
3	etx
4	eot
5	enq
6	ack
7	bel
8	bs
9	ht
10	nl
11	vt
12	np
13	cr
14	so
15	si
16	dle
17	dc1
18	dc2
19	dc3

20	dc4
21	nak
22	syn
23	etb
24	can
25	em
26	sub
27	esc
28	fs
29	gs
30	rs
31	us
32	sp
33	!
34	“
35	#
36	$
37	%
38	&
39	‘

40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;

60	<
61	=
62	>
63	?
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O

80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_
96	`
97	a
98	b
99	c

100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w

120	x
121	y
122	z
123	{
124	|
125	}
126	~
127	del

In memory, a character variable is stored with its ASCII code, an integer less than 127, and thus can be treated as an integer.

```
char   c = ‘A’;
printf(“asci code of %c is %d”, c, c); //ascii code of A is 65
int  n = ‘Z’ - ’A’;  //n will be 25
```

Important string functions

- int strlen(char *) exp: char name[]="Jack"; int n=strlen(name);

  char * strcpy(char *, char *) exp: char name[100];

  name="Jack"; //this is wrong

  strcpy(name, "Jack");

- int strcmp(char *, char *) exp: int n=strcmp("Jake", "Jack");

  //the value of n will be 'k' - 'c' = 8

- char * gets(char *) exp: char name[100]; gets(name);

  char * puts(char *) exp: puts(name);

- char * fgets(char *, int, FILE *) exp: fgets(name, 100, fin);

  char * fputs(char *, FILE *) exp: fputs(name, fout);

Pointer NULL

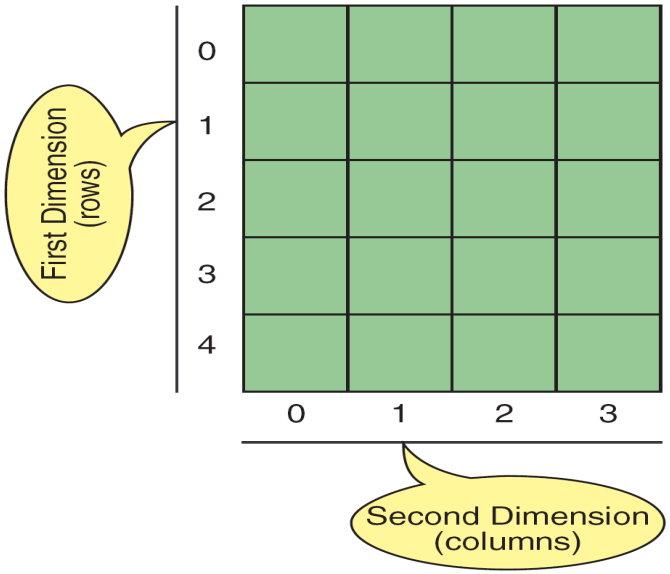
- NULL is a built-in constant for pointer:

  int * ptr_c = NULL; char * str = NULL;

- Use fgets/fputs instead of fscanf/fprintf for copying a file:

```
char str[100];
while ( fgets(str, 100, fin) != NULL ) {
        fputs(str, fout);
}
```

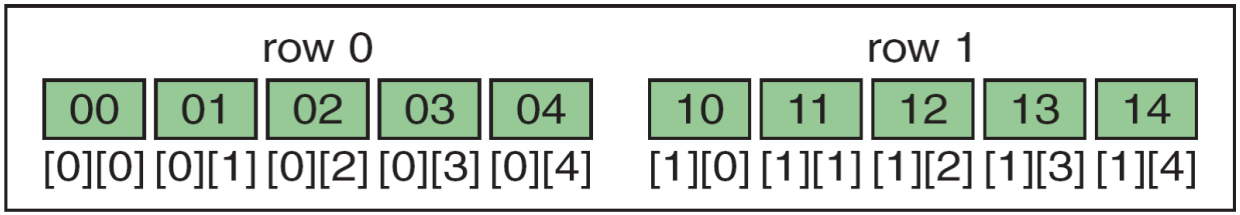
Two-dimensional Array




```
int  table[2][5] =
{
    {00,01,02,03,04},
    {10,11,12,13,14}
};
```

00	01	02	03	04
10	11	12	13	14

User's View



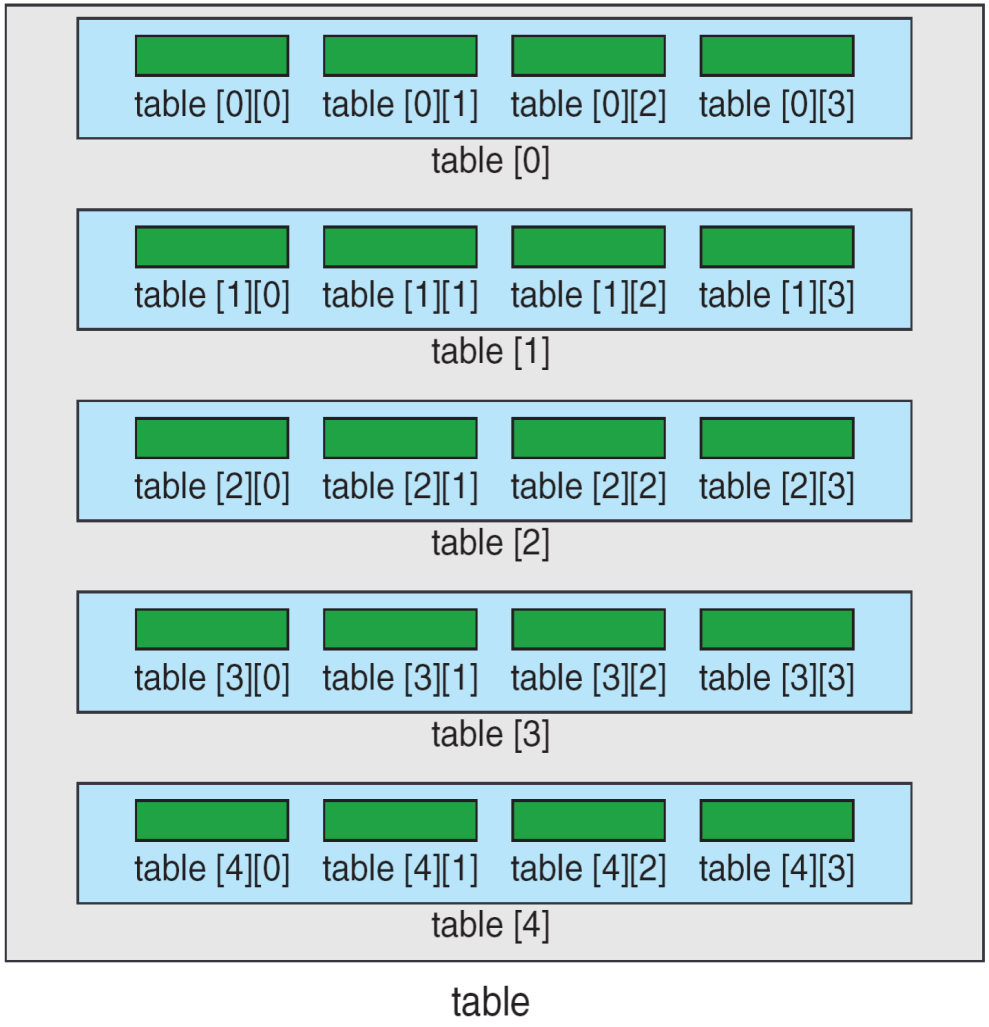
Memory View

Two-dimensional Array

```
int   table[5][4];
int  ** pptr_i = table;
```

table[0][0],…table[4][3]
are integer variables

table[0], … table[4]
are of type int *



Two-dimensional Character Array

```
char  names[ ][10] =
{
     “chris”,
     “Sophia”,
     “Lucas”,
     “Ethan”
};

for ( int i=0; i<4; i++ )
     printf(“ name of No. %d person is:%s\n”, i+1, name[i]);
```

Pointer NULL

- NULL is a built-in constant for pointer:

  int * ptr_c = NULL; char * str = NULL;

- Use fgets/fputs instead of fscanf/fprintf for copying a file:

```
char str[100];
while ( fgets(str, 100, fin) != NULL ) {
        fputs(str, fout);
}
```

Thank you!