Revision of C

Dr. Alekha Kumar Mishra

About C

- Evolution
  - BCPL by Martin Richards
  - B by Ken Thompson
  - C in the 1970s by Dennis Ritchie

Features

- Combines the best elements of high-level languages with the control and flexibility of assembly language
- C allows the manipulation of bits, bytes, and addresses—the basic elements of an OS
- A Structured Language

C Tokens

- Keywords
- Constants
- Identifiers
- Special symbols
- Operators
- Strings

C keywords

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

- Meaning is already explained to C Compiler
- Must be written in lowercase letters

Identifiers

- Names that are given to various programming elements
- Identifiers are mostly user-defined words
- Programming elements
  - Variables
  - Functions
  - Labels
  - other user-defined objects

Identifier rules

- The name should consist of only
  - alphabets (A,B.....Z or a,b.....z)
  - Digits (0,1.....9)
  - Underscore sign (_)
- First character should be an alphabet or underscore
- The name should not be a keyword
- Case sensitive
- Some compilers recognize up to 31 characters

Constant and Variable

- The value of a constant does not change during program execution
- Variable is a name used to store values
- Values assigned to variable can be changed during execution of a program
- Variables
  - Local
  - Global

Data Types

- The way data is represented within computer memory.
- Description of the kind of data stored, passed and used.
- Classes of Data Types
  - Primary (Fundamental) Data Types
  - User-defined Data Types
  - Derived data types
  - Empty Data

Primary data types

Integer

signed type	**unsigned type**
int	unsigned int
short int	unsigned short int
long int	unsigned long int

Character

signed char
unsigned char

Floating Point

float
double
long double

Data type qualifiers

Size Qualifiers
- short
- long

Sign Qualifiers
- signed
- unsigned

Note:

- If unsigned qualifier is assigned, the number is always positive
- If signed qualifier the number may be positive or negative

Symbolic Constant

- It is the name that substitutes for a sequence of characters.
- Characters may represent numeric or Character or String Constant
- When a program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence.
- Usually defined at the beginning of the program
- Syntax
- #define name text

Storage Class Specifiers

- extern
- static
- register
- auto

Single character I/O

- Single Character Input by getchar function
- Syntax -:
  - variable = getchar()
- Single Character Output by putchar function
- Syntax -:
  - putchar(variable)

Example

```
#include <stdio.h>
int main(){
    char c;
    …
    c=getchar();
    putchar(c);
    …
    return 0;
}
```

Input by scanf() function

- Moves the data from standard input device to computer’s memory
- Can be used to enter any combination of numerical values, single characters and strings.
- This functions returns the number of data items that have been entered successfully
- Syntax :-
  - scanf(control string, argument1, argument2, …);

Input by scanf() function

- The control string consists of individual groups of characters with one character group for each argument.
- Each character group of control string must begin with %
- In simpler form, each character group contains % followed by the conversion character which indicate the type of the corresponding data item (arguments)

Conversion characters & example

Conversion Character	Meaning
c	Single char
d	Decimal integer
f	Floating point
h	Short integer
l	Long integer
s	String (without space)
[^\n]	String includes space

```
#include <stdio.h>
int main(){
    int a,b;
    char c;
    float m;

    scanf(“ %d “,&a);
    scanf(“ %d %d”,&a,&b);
    scanf(“ %d %c”,&b,&c);
    scanf(“ %c %d %f”,&c,&b,&m);
    …
    return 0;
}
```

output with printf() function

- It is similar to the input function scanf(), except that its purpose is to display data.
- Moves the data from computer's memory to the standard output device
- Syntax :-
  - printf(control string, argument1, argument2,....);
- In contrast to the scanf() function, the arguments in the printf function do not represent memory addresses an therefore not preceded by &.

Example

```
#include <stdio.h>
int main(){
    int a=4,b;
    char c=‘x’;
    float m;
    b=34;
    printf(“ value of a=%d\n ”,a);
    printf(“ c is %c and b=%d\n”,c,b);
    printf(“enter value for m\n”);
    scanf(“ %f ”,&m);
    printf(“%f\n”,m);
    …
    return 0; }
```

gets() and puts() functions

- Facilitates input and output of strings respectively
- Both functions takes one argument of string type
- Provides simple alternatives to the use of scanf and printf for reading and displaying strings
- Example
  - gets(str);
  - puts(str);

Operators in C

- Arithmetic Operator
- Unary Operator
- Relational Operator
- Logical Operator
- Assignment Operator
- Conditional Operators

Operator precedence and associativity

Operator	Description	Associativity
() [] . -> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to and not equal to	left to right
&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
|	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
||	Logical OR	left to right
? :	Ternary operator	right to left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	Comma operator	left to right

Type conversion in assignment

- In an assignment, When variables are of mixed type, a type conversion will occur
- The value of the right side (expression side) of the assignment is converted to the type of the left side (target variable)
- We can force an expression to be of a specific type by using a cast.
  - (type) expression

Branching in C

- if-else statement used to carry on logical test and takes one of two possible actions depending on outcome of the test (true or false)
- else portion of if-else statement is optional
- Simplest form

```
if(expression)
        statement;
```

- General form

```
if(expression)
        statement1;
else
        statement2;
```

Branching in C

```
#include <stdio.h>
int main(){
    int num;
    printf("enter a number");
    scanf("%d",&num);
    if(num>0)
        printf("number is > 0");
    if(number<=100)
        printf("number is <=  100");
    else
        printf("number is > 100");
    return 0;
}
```

if-else nested

- If the body of if or/and else part contains more than one statement then those statements should be enclosed within curly braces { }

```
if(expression){
    statement1;
    statement2;
    ...
}
else{
    statement3;
    statement4;
    ...
}
```

- **Nested if-else statements**

```
if(expr1)
    if(expr2)
        stmt1;
    else
        stmt2;
else
    if(expr3)
        stmt3;
    else
        stmt4;
```

If-else-if ladder

- When we need to create a situation in which one of several different courses of will be selected, we can take help of if-else-if structure as follows

```
if(expr1)
    stmt1;
else if(expr2)
    stmt2;
else if(expr3)
    stmt3;
else
    stmt4;
```

```
#include <stdio.h>

int main(){
    float time;
    printf("enter current time");
    scanf("%f",&time);
    if( time >= 0.0 && time<12.0 )
        printf("Good Morning");
    else if( time >12.0 && time < 18.0 )
        printf("Good Afternoon");
    else if( time >18.0 && time < 24.0 )
        printf("Good Evening");
    else
        printf("time is not valid");
    return 0;
}
```

Looping

- A loop allows a program to repeat a group of statements, either a fixed number of times or until some exit condition is satisfied
- Convenient if the exact number of repetitions are known
- Loop Consists of
  - Body of the loop
  - Control Statement
- Steps in Loop
  - Initialization of condition variable
  - Execution of body of the loop
  - Test the control statement
  - Updating the condition variable
- Loops in C
  - While loop
  - Do-While loop
  - For loop

While loop

```
while(condition){
    statement1;
    statement2;
    statement3;
    …
    …
}
statement n;
```

```
#include <stdio.h>
int main(){
    int num,sum=0;
    printf(“enter a number”);
    scanf(“%d”,&num):
    while(num){
        sum+=(num%10);
        num/=10;
    }
    printf(“ %d”,sum);
}
```

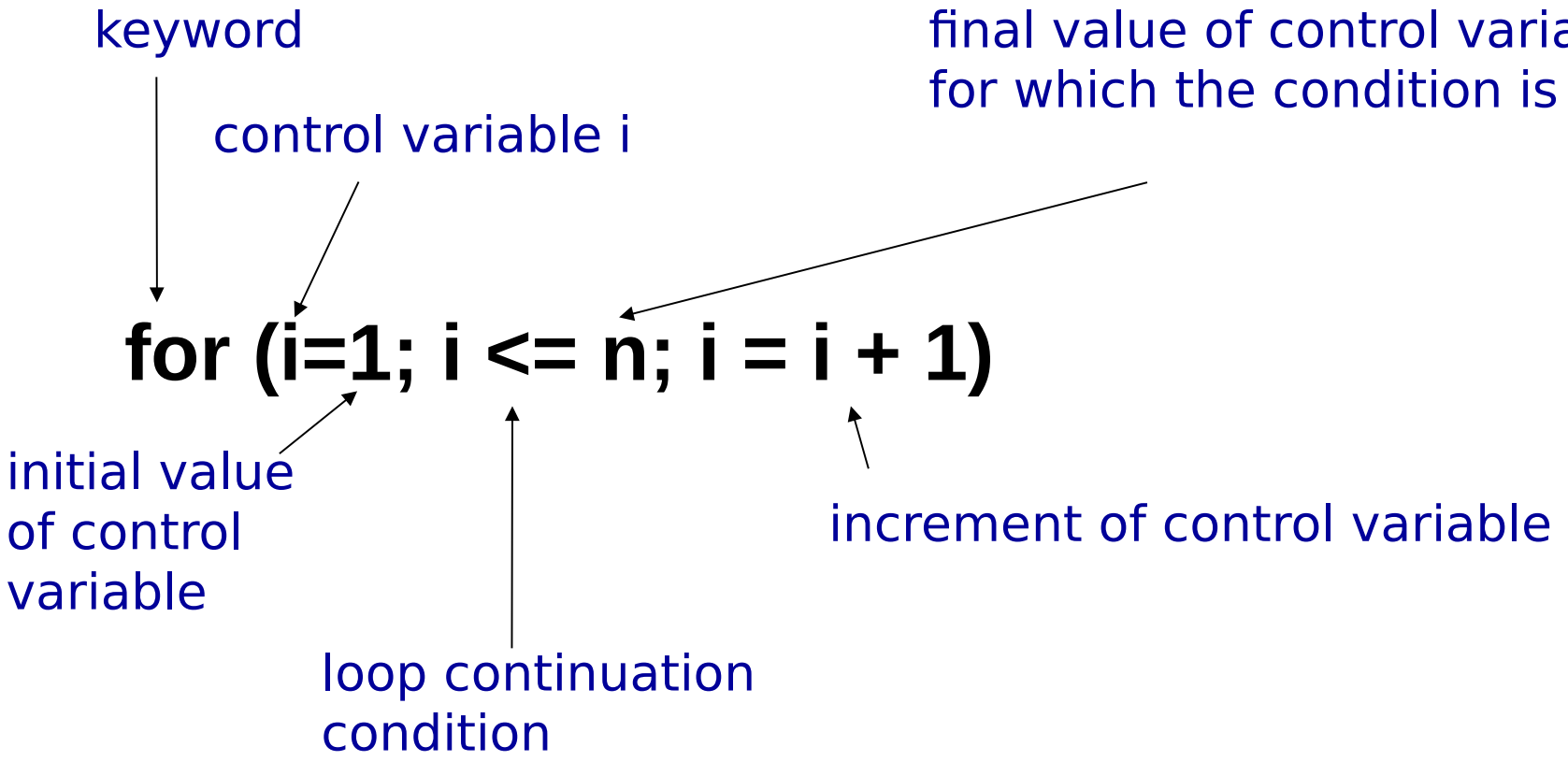
do-while loop

```
do{
    statement1;
    statement2;
    statement3;
    …
    …
}while(condition);
statement n;
```

```
#include <stdio.h>

int main(){
    int i = 1, sum = 0;
    do{
        sum = sum + i;
        i= i + 1;
    }while(i<=10);
    printf("Sum = %d\n", sum);
    return 0;
}
```

For loop


keyword
control variable i
final value of control variable
for which the condition is true
for (i=1; i <= n; i = i + 1)
initial value
of control
variable
loop continuation
condition
increment of control variable

Example

```
#include <stdio.h>
int main(){
        int i,num,p=q=0;
    printf(“enter a number”);
    scanf(“%d”,&num):
    for(i=1;i<=num;i++){
         if( i % 2 == 0)
      p+=i;
        else
      q+=i;
   }
    printf(“ %d %d\n”,p,q);
    return 0;
}
```

```
/* NESTED LOOPS */
for(i=1;i<n ;i++)
     for(j=1;j<m;j++){
        k=j;
        while(k){
           printf(“%d \n”,(i-j+k));
            k--;
       } //end of while
    } // end of inner for
```

For loops

- for(; ;) printf("This loop will run forever.\n");
- for(; *str == ' '; str++) ;
- for(t=0; t<SOME_VALUE; t++) ;

break and continue

- These interrupt normal flow of control
- **break** causes an exit from the innermost enclosing loop
- **continue** causes the current iteration of a loop to stop and the next iteration to begin immediately

```
#include <stdio.h>
int main(){
    int i,num;
    scanf(“%d”,&num);
    for(i=2;i<num;i++){
        if(num % i == 0)
            break;
        else
            continue;
    }
    if(i==num)
        printf(”yes”);
    return 0;
}
```

switch statement

- The complexity of a program increases by increasing no. of if statement.
- To overcome this C has a built in multi-way decision statement known as switch
- Expression results in integer value. May also be a of char type
- When switch statement is executed, the expression is evaluated and control transferred directly to the group of statements whose case value matches the value of expression.

```
switch(expression){
    case value1:
        stmt1;
        stmt2;
        …
    case value2:
        stmt3;
        …
   default:
        stmtn;
        …
}
```

```
int main(){
   char color;
   printf("give colour option");
   scanf("%c",&color);
   switch(color)
   {
       case 'r':
       printf(" RED");
       case 'b':
       printf(" BLUE");
       case 'g':
       printf(" GREEN");
       case 'w':
       printf(" WHITE");
       default:
       printf(" Not available");
    }
    return 0;
}
```

switch statement

- The complexity of a program increases by increasing no. of if statement.
- To overcome this C has a built in multi-way decision statement known as switch
- Expression results in integer value. May also be a of char type
- When switch statement is executed, the expression is evaluated and control transferred directly to the group of statements whose case value matches the value of expression.

```
int main(){
   char color;
   printf("give colour option");
   scanf("%c",&color);
   switch(color)
   {
       case 'r':
       printf(" RED");
       break;
       case 'b':
       printf(" BLUE");
       break;
       case 'g':
       printf(" GREEN");
       break;
       case 'w':
       printf(" WHITE");
       break;
       default:
       printf(" Not available");
    }
    return 0;
}
```

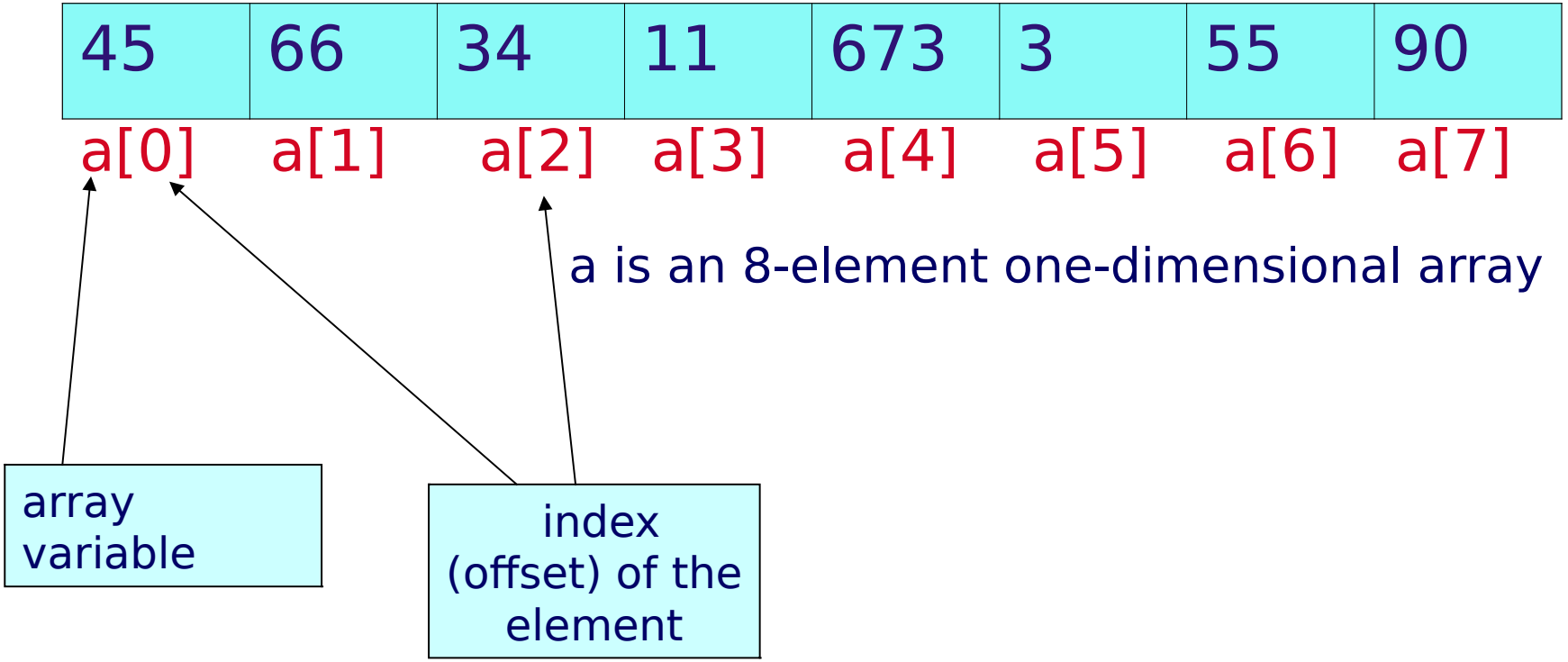
Array

- In many cases, we need to process multiple data items have common characteristics or type.(say x_1, x_2,....., x_n)
- Using individual variables to handle each data items would be tedious task
- It is often convenient to place all data item a single structure defined with same variable name
- Array is the solution to above in C

Array in C

- Data items placed in array share the same name
- They all must be of same type and same storage class
- So we can have
  - Integer array
  - Character array
  - Floating-point array
- Each individual element is referred to by specifying the array name followed by one or more subscript, with each subscript enclosed in square bracket [ ].
- Subscript value (index of element) for an n-element array varies from o to n-1
- Number of subscripts determines dimension of array.
  - X[i] – one dimensional
  - X[i][j] – two dimensional

Example


45
66
34
11
673
3
55
90
a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a is an 8-element one-dimensional array
array variable
index (offset) of the element

Defining array in C

- Syntax
  - *data-type arrayname[size];*
- Example
  - int x[100];
  - char message[25];
  - float n[12]
- The elements in an array are stored adjacent to one another in memory.
- An array element is accessed directly, simply by specifying the desired *index*

Declaration cum Initialization

- int arr[5]={ 4, 6, 8, 11, 7 };
- int arr[ ]= { 4, 6, 8, 11, 7 };
- char color[4] = { ‘B’, ‘L’, ‘U’, ‘E’ };
- char color[ ] = { ‘B’, ‘L’, ‘U’, ‘E’ };
- int num[7] = {3, 4, 2}
- float frac[5] = {-5.9, 0, 3.7 }

frac[0] = -5.9

frac[1] = 0

frac[2] = 3.7

frac[3] = 0

frac[4] = 0

Character Array vs String

- A special symbol called null character (‘\0’) is placed at the end of each string. Whereas, this is not present in a simple array of characters

- char color[3] = “RED”
- char color[ ] = “RED”
- char color[4] = “RED”

  - color[0]=‘R’
  - color[1]=‘E’
  - color[2]=‘D’
  - color[3]=‘\0’

Processing an array

- Single operations which involve entire arrays are not permitted in C
- For example, to check if two array are similar, the comparison operation must be carried out on an element-by-element basis.
- Usually access to all elements is accomplished with the help of loops
- Accessing an array with index out of its range will generate a runtime error

Example

```
#include <stdio.h>

int main()
{
int num[10], m, i;
for(i=0;i<10;i++){
   printf(“enter the elements ”);
   scanf(“ %d”,&num[i]);
}
m = num[0];
for (i=1; i<10; i++) {
   if (num[i] < m)
        m = num[i];
}
printf(“ %d”, m);
return 0;
}
```

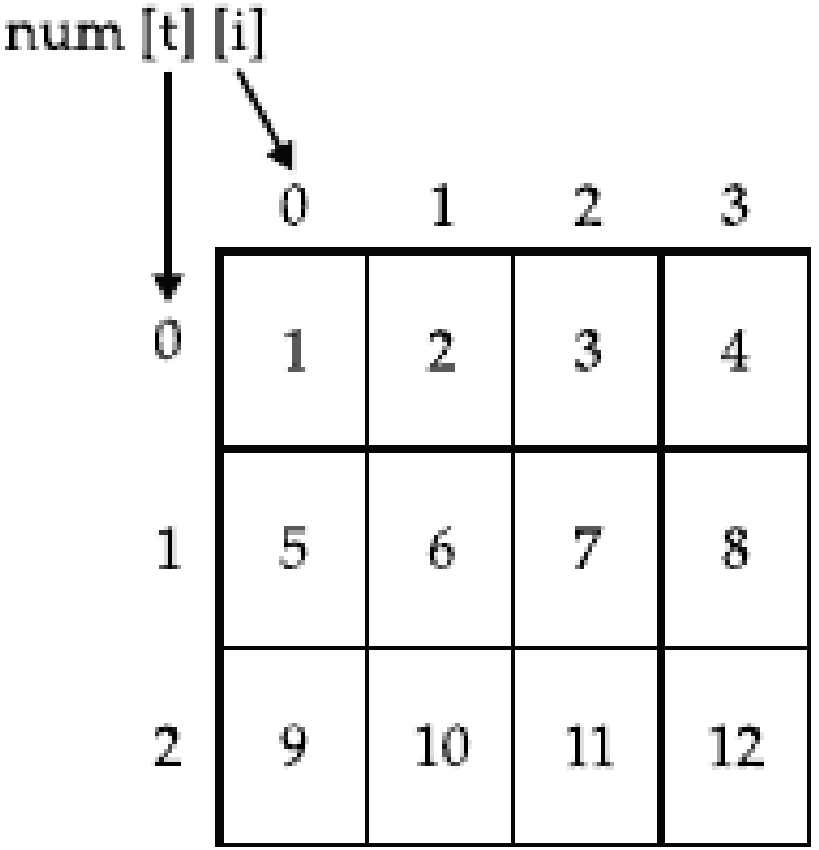
Symbolic constant for size of the array

- Some time it is convenient to define an array size in terms of a symbolic constant rather than a fixed integer quantity

```
#include <stdio.h>
#define SIZE 10
int main()
{
int num[SIZE], m, i;
for(i=0;i<SIZE;i++){
   printf(“enter the elements ”);
   scanf(“ %d”,&num[i]);
}
m = num[0];
for (i=1; i<SIZE; i++) {
   if (num[i] < m)
        m = num[i];
}
printf(“ %d”, m);
return 0;
}
```

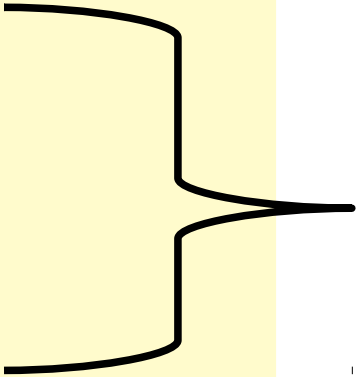
Multi-dimensional array

```
int main(void)
{
    int t, i, num[3][4];
    for(t=0; t<3; ++t)
      for(i=0; i<4; ++i)
          num[t][i] = (t*4)+i+1;
      /* now print them out */
    for(t=0; t<3; ++t) {
      for(i=0; i<4; ++i)
          printf("%3d ", num[t][i]);
      printf("\n");
    }
    return 0;
}
```



Declaration cum initialization

- int arr[3][4]={ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
- int arr[ ][4]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
-
- int arr[ ][ ]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
- char arr[3][4] = {
  { 1, 2, 3, 4 },
  { 5, 6, 7, 8 },
  { 9, 10, 11, 12 }
  };
- char arr[3][4] = {
  { 1, 2, 3 },
  { 4, 5, 6 },
  { 7, 8, 9 }
  };
- int arr[3][4]={ 1, 2, 3, 4, 5, 6, 7, 8, 9 };



arr[0][0] = 1
arr[0][1] = 2
arr[0][2] = 3
arr[0][3] = 4
arr[1][0] = 5
arr[1][1] = 6
arr[1][2] = 7
arr[1][3] = 8
arr[2][0] = 9
arr[2][1] = 0
arr[2][2] = 0
arr[2][3] = 0

Passing 2D array as arguments

- Only a pointer to the first element is actually passed.
- The parameter receiving a 2D array must define at least the size of the rightmost dimension
  - The rightmost dimension is needed because the compiler must know the length of each row

End of part 1