CPS 101 Introduction to Computational Science

Wensheng Shen

Department of Computational Science
SUNY Brockport
Chapter 10: Program in C

- Why C language:
  - C is the language which can be used for both system software and application software.
  - C is human readable, and also can be used to access hardware.
The origin of C language

(1) C language was developed by Dennis M. Ritchie between 1972 to 1973 at Bell Laboratory.
(2) It was used to develop the famous Unix operating system in 1973 by D.M. Ritchie and K. Thompson.
(3) “The C Program Language” by Brian W. Kernighan and Dennis M. Ritchie was the most famous book in C language.
Characteristics of C Language

- It is concise, compact, flexible, and convenient
- There are only 32 keywords, 9 control statements
- It is an idea language for structural programming
10.1 Introduction to C language

- Editing, compiling and executing using Visual Studio;
- A simple C code
- Components in a C code
- The structure of a function in C language
The C++ Compiler

- We use Microsoft Visual Studio.
- Launch Microsoft Visual Studio.
  - Start->Programs->Microsoft Visual Studio
    ->Microsoft Visual C++ (whatever version)
- Find **File** menu on the pop-up window, select new, select project, select Win32 Console Application.
  - File->New->Project->Win32 Console Application
  - Save file: Name->Location->Finish
What is C language

- C is called a **compiled language**
- A C program, human readable, must be run through a **C compiler** to turn it into an **executable** that the computer can run (execute).
A simple C code

```c
int main(int argc, char* argv[])
{
    printf ("Hello World from C\n");
}
```
A simple C code

```c
int main(int argc, char* argv[])
{
    int a, b, sum;
    a = 123;
    b = 456;
    sum = a + b;
    printf("The summation of %d and %d is %d\n", a, b, sum);
}
```
More example

```c
int main(int argc, char* argv[]) {
    int a, b, c;
    scanf("%d, %d", &a, &b);
    c = max(a, b);
    printf("The maximum of %d and %d is %d \n", a, b, c);
    return (0);
}

int max(int x, int y) {
    int z;
    if (x > y) z=x;
    else z=y;
    return (z);
}
```
Compiling and executing using Visual Studio

- Create a new project, or open an existing project.
- Build the solution to create an exe file under the directory /debug.
- Go to the build menu to run the executable file.
Compiling, building, and executing using Visual Studio

- In Visual Studio:
  - Source code
  - Build
  - executable file
  - Sort3.c
  - Build
  - Sort3.exe
  - EXAMPLE

```
10110011001100111001011001  
10110011001100111001011001  
01010101010101010101010101  
00100111001001011100101010  
01010101010101010101010101  
00000110101010100101010101
```
int main(int argc, char*argv[]) {
    int a, b, sum;
    // a=123;
    // b=456;
    scanf("%d %d", &a, &b);
    sum = a + b;
    printf("The summation of %d and %d is %d\n", a, b, sum);
}

Component of C program

- The C program starts with `#include <stdio.h>`. This line includes the “standard I/O library” into your program. Keyboard is “standard in” and screen is “standard out”
- A **library** is simply a package of code that someone else has written for you to use to make your life easier
- `int main(int argc, char* argv[])` declares the main function. Every C code should have a main function somewhere, or, a C program consists of at least one function, the **main** function.
- At run time program execution starts at the first line of the main function.
- A function include two parts: description and body.
- Description of a function:
  - type, name, parameters, the type of parameters.

Example:
```
int max (int x, int y)
```
Body of a function:
- The contents of a function should be included in the pair of "{ }", following the description of the function.
- The declaration of variables.
- The operation of the function.

Example:

```c
int max(int x, int y) {
    int z;
    if (x > y) z=x;
    else z=y;
    return (z);
}
```
Each statement or definition must be ended with ";".

Input and output are operated by the functions "scanf" and "printf" respectively.

Comments are marked by the pair "/\*" and "*/". In other words, anything between "/\*" and "*/" are comments without and functional ability.
The `printf` statement allows you to send output to standard out (for us, the screen). The portion in quotes is called the **format string** and describes how the data is to be formatted when printed. The format string can contain string literals such as “Hello World From C!.” To change a line or return, C use “\n”.

The **return 0;** line causes the function to return an error code of 0 (no error) to the shell that started execution. (shell means any program that users use to type commands).
10.2 Data Types in C Language

- A program consists of two parts
  - Data description
  - Procedures
- Program = data structure + algorithms (Nikiklaus Wirth)
- Program = data structure + algorithms + program design + language environment
Basic data types

- C has the following data types:
  - `int` --- integer values;
  - `float` --- floating point values in single precision;
  - `double` --- floating point values in double precision;
  - `char` --- single character values;
Constant and Variable

- Constant: integer, real (float, double), char
  
  #define PI = 3.14159 or
  
  Using capital letters to identify constants; The values of constants cannot be changed in the program.
Character constant

- Special character constants: ‘\n’, ‘\t’, ‘\v’
- Character string: “USA”, “How are you?”
Variables

- We use **variables** to remember things or values. The value of a variable can be changed in a program.
  
  ```
  int a;
  ```

- A variable has a **name** (in this case, `a`) and a **type** (in this case, `int`, an integer).

- We can store a value to `a` by initializing:
  
  ```
  a = 10;
  ```

  We can print the value by doing:
  
  ```
  printf("%d", a);
  ```
Variables

- The name of a variable must be:
  - letters, numbers, and underscore
  - A variable must begin with letters or underscore, but not numbers;

- Examples:
  - Sum, average, _above, lotus_1_2_3
  - #33, 3d64, M.D.John, a>b
Calculation of combined data types

- $10+\text{'a'}+1.5-57.654*\text{'b'}$
Arithmetic operations and arithmetic expressions

+ --- addition (3 + 4)
- ---- subtraction (10 - 5)
* ---- multiplication (3*6)
/ ---- division (5/3)
% ---- module (5%3)

a*b/c – 2.5 + ‘b’
Conversion of data type

- (double) a
- (int) (x+y)
- (float) (7%3)
Self-addition and self-subtration

- `++i, --i`: let i increase or decrease by 1 before using i;
- `i++, i--`: let i increase or decrease by 1 after using i;

Example (i=5):
- `j = ++i; (i = 6, j = 6)`
- `j = i++; (j = 5, i = 6)`
- `i = 10;`
- `printf("%d", ++i);` what is output?
- `printf("%d", i++);` what is output?
Assign values to variables

- Variable name = expression;
- \( a = 10; \ a = b = 6; \)
- \( a = 10 + (c = 6); \)
- \( a = (b = 20)/(c = 4); \)
int main(int argc, char *argv[])
{
    int i, j, m, n;
    i = 10; j = 12;
    m = ++i;
    n = j++;
    printf("%d %d %d %d", i, j, m, n);
}

10.3 Simple C Program Design

- Control Statements in C:
  1. if() {} else{}
  2. for() {}
  3. while() {} 
  4. do{}while()
  5. continue
  6. break
  7. switch
  8. return
  9. goto
- Function call statements
  - printf();
  - max(a, b);
- Expression statements
  - c = a + b;
- Empty statements
  - ;
Basic Structure of Program Design

(1) Sequential structure
(2) Selection structure
(3) Loop structure
Data Output

- `putchar(c)`: output a character to terminal
- `char c;`

```c
#include "stdio.h"
int main(int argc, char *argv[]) {
    char a, b, c;
    a = 'U'; b='S'; c='A';
    putchar(a); putchar(b); putchar(c);
    return 0;
}
```

```c
#include "stdio.h"
int main(int argc, char *argv[]) {
    char a, b, c;
    a = 'U'; b='S'; c='A';
    putchar(a); putchar(\n); putchar(b); putchar(\n); putchar(c);
    return 0;
}
```
Printf

- The `printf` statement allows you to send output to standard out, which is generally the screen.
  ```c
  int main()
  {
    int a, b, c;
    a = 5; b = 7;
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    return 0;
  }
  ```
Use **printf** to print information on the screen asking for user input.

```c
int main(int argc, char* argv[]) {
    int a, b, c;
    printf("Enter the first value: \n");
    scanf("%d", &a);
    printf("Enter the second value: \n");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d \n", a, b, c);
    return (0);
}
```
## Printf

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Integer (decimal)</td>
</tr>
<tr>
<td>o</td>
<td>Integer (octal numbers)</td>
</tr>
<tr>
<td>x</td>
<td>Integer (hexadecimal)</td>
</tr>
<tr>
<td>c</td>
<td>a char</td>
</tr>
<tr>
<td>s</td>
<td>a string</td>
</tr>
<tr>
<td>f</td>
<td>Decimal, six digits after decimal points (single or double precision)</td>
</tr>
<tr>
<td>e</td>
<td>Exponential form, six digits after decimal points (single or double precision)</td>
</tr>
<tr>
<td>g</td>
<td>Output the short one of %f and %e</td>
</tr>
</tbody>
</table>
Printf

- Additional format:
  - “%m.nf”: output data occupies total m columns, and there are n digits after the decimal point;
  - “%-m.nf”: the same rule as “%m.nf”, but data is aligned left.

```c
int main (int argc, char* argv[]) {
    double x, y, z;
    x = 111111111111.111111111;
    y = 333333333333.333333333;
    z = x + y;
    printf("%f\n", z);
    return (0);
}

printf("%10f\n");
printf("%24f\n");
printf("%24.6f\n");
printf("%.6f\n");
printf("%-24.6f\n");
```
Printf

- Additional format:
  - “%m.ne”: m is the total number of columns a data occupies, n is the number of digits after the decimal point.
  - “%-m.ne”: the data is aligned left.

```c
printf("%e", 123.456)

x = 123.456;
printf("%e\n");
printf("%10e\n");
printf("%10.2e\n");
print("%.2e\n");
Printf("%-10.2e\n");
```
Data input

- getchar() --- input a char from terminal;

```c
#include "stdio.h"
int main(int argc, char *argv[]) {
    char c; c = getchar(); putchar(c); return 0;
}
```

```c
#include "stdio.h"
int main(int argc, char *argv[]) {
    char c; printf("%c\n", getchar()); return 0;
}
```
Scanf

- "%d": integer
- "%f": single precision real numbers
- "%lf": double precision real numbers
- "%e": the same as "%f"
- "%c": single character
- "%s": character string

```c
int main(int argc, char* argv[]) {
    int a, b, c;
    double x, y, z;
    scanf("%d %d %d", &a, &b, &c);
    scanf("%lf %lf %lf", &x, &y, &z);
    printf("a = %d, b = %d, c = %d\n", a, b, c);
    printf("x = %f, y = %e, z = %10.4f\n", x, y, z);
}
```
Issues in `scanf`

```c
Int main(int argc, char* argv[]) {
    int a, b;
    double x;
    scanf("%d %d", a, b);
    scanf("%d", &x);
    scanf("%6.4f", &x);
}
```
10.4 Branching and logic operations

- The six relational operators in C language:
  - `<` (less than)
  - `<=` (less than or equal to)
  - `>` (greater than)
  - `>=` (greater than or equal to)
  - `==` (equal to)
  - `!=` (not equal to)
Relational expressions

- $a > (b + c)$
- $(a > b) != c$
- $a == (b < c)$
- $a = (b > c)$

a=1; b = 2; c = 3
```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a;
    printf("please enter a number: ");
    scanf("%d", &a);
    if (a < 0)
        printf("The value is negative\n");
    return 0;
}
```
Logic operators and logic expressions

- There are three logic operators:
  - `&&` (and)
  - `||` (or)
  - `!` (not)

Arithmetic operator: `!`
Relational operator: `&`, `|`
Assignment operator: `=`

- \(!a \lor a>b\) → (\(!a\) \lor (a>b))
- \(a>b \land x>y\) → (\(a>b\) \land (x>y))
- \(a=a\land x=y\) → (\(a=a\) \land (x=y))
### Truth table

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>!a</td>
<td>!b</td>
<td>a&amp;&amp;b</td>
<td>a</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Boolean expressions

- \((a > b) \&\& (a > c)\)
- \((a == b) \mid\mid (c == d)\)
- \((! a) \&\& (c > d)\)
Questions

- Are these a valid boolean expression
  
  - $(c + d) \land (a + 5)$?
  
  - $(a = 5) \land (c > 1)$?
  
  - $!(a) \land (c + d)$?
  
  - $!(a + b)$
Attention
--- the result of a logic calculation
    --- 1 for true
    --- 0 for false
--- for a value in logic expression
    --- 0 for false
    --- non-zero for true
Example

a = 4 what is $\rightarrow \neg a$

a = 4; b = 5 what is $\rightarrow a \& \& b$?

a = 4; b = 5 what is $\rightarrow \neg a \| b$?

What is $4 \& \& 0 \| 2$?

What is $5 > 3 \& \& 2 \| 8 < 4 - \neg 0$?
## Truth table

| a    | b     | !a | !b | a&&b | a||b |
|------|-------|----|----|------|-----|
| not 0| not 0 | 0  | 0  | 1    | 1   |
| not 0| 0     | 0  | 1  | 0    | 1   |
| 0    | not 0 | 1  | 0  | 0    | 1   |
| 0    | 0     | 1  | 1  | 0    | 0   |
Values of boolean operations

- Let $a = 3$, $b = 4$, $c = 5$;
- (a) $(a + b > c) \land \land (b == c)$
- (b) $a \lor b + c$
- (c) $(a + b) > c \land \land (a + c < b)$

In C language, 0 indicates false, non zero indicates true.
What is leap year: (1) dividable by 4 but not dividable by 100; (2) dividable by 400.

year % 4 == 0; year % 100 != 0; year % 400 == 0.
If statements

(1) If (expression)
    statement

(2) If (expression)
    statement
Else
    statement

(3) If (expression)
    statement
else if (expression)
    statement
else if (expression)
    statement
else
if( (a+b)>c && (a+c)>b && (b+c)>a )
{
    s = 0.5*(a + b + c);
    area = sqrt(s*(s - a)*(s - b)*(s - c));
    printf("area of the triangle = %f\n", area);
}
else
{
    printf("it is not a triangle!!!\n");
}
int main(int argc, char *argv[]) {

    double a, b, t;
    printf("Input any two real numbers:\n");
    scanf("%lf %lf", &a, &b);
    if(a>b) {
        t = a;
        a = b;
        b = t;
    }
    printf("the two numbers are: %f and %f\n", a, b);

}
```c
#include "stdio.h"
int main(int argc, char* argv[]) {
    double a, b, c, t;
    scanf("%lf %lf %lf", &a, &b, &c);
    if(a > b) {
        t = a;
        a = b;
        b = t;
    }
    if(a > c) {
        t = a;
        a = c;
        c = t;
    }
    if(b > c) {
        t = b;
        b = c;
        c = t;
    }
    printf("%f, %f, %f\n", a, b, c);
    return 0;
}
```
```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int b;
    printf("Enter a value:");
    scanf("%d", &b);
    if (b < 0)
        printf("The value is negative\n");
    else if (b == 0)
        printf("The value is zero\n");
    else
        printf("The value is positive\n");
    return 0;
}
```
```c
#include "stdio.h"
#include "math.h"

int main(int argc, char* argv[]) {
    double a, b, c, check, x1, x2, p, q;
    printf("please input the coefficients of the quadratic equation\n");
    scanf("a = %lf, b = %lf, c = %lf", &a, &b, &c);
    check = b*b - 4.0*a*c;
    p = -b/(2.0*a);
    q = sqrt(check)/(2.0*a);
    x1 = p + q;
    x2 = p - q;
    printf("The roots of the quadratic equation %f x^2 + %f x + %f is:\n", a, b, c);
    printf(" x1 = %10.4f, x2 = %10.4f\n", x1, x2);
}
```
= vs. == in Boolean expressions

- The == sign is a problem in C because very often you may forget and type just = in a Boolean expression.
- This is an easy mistake to make, but to the compiler there is a very important difference.
- C will accept either = and == in a Boolean expression, but the behavior of the program changes remarkably between the two, however.
- Boolean expressions evaluate to integers in C, and integers can be used inside of Boolean expressions. The integer value 0 in C is False, while any other integer value is True.

- In C, a statement like `if (a=b)` means, "Assign b to a, and then test a for its Boolean value." So if a becomes 0, the if statement is False; otherwise, it is True. The value of a changes in the process. This is not the intended behavior if you meant to type `==`, so be careful with your `=` and `==` usage.
```c
#include <stdio.h>
int main(int argc, char* argv[]) {
    int a;
    printf("Enter a number: ");
    scanf("%d", &a);
    if (a) {
        printf("The value is True\n");
    }
    return 0;
}
```

Try this using: 0, 2, 4, -1
```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a, b;
    printf("Enter a number:");
    scanf("%d", &b);
    if (a = b) {
        printf("The value is True\n");
    }
    return 0;
}
```

Try this using: 0, 2, 4, -1
Switch statement

- There are only two branches in "if" "else" statement
- Multi-branching using switch

```java
switch(expression)
{
    case constant 1: statement 1;
    case constant 2: statement 2;
    ......
    case constant n: statement n;
    default : statement n+1;
}
```
Switch(grade)
{
    case 'A': printf("90 ~ 100\n");
        break;
    case 'B': printf("80 ~ 89\n");
        break;
    case 'C': printf("70 ~ 79\n");
        break;
    case 'D': printf("60 ~ 69\n");
        break;
    default : printf("error\n");
}
Program examples

- Write a program to decide if a year is a leap year

Program design:
- Input any year from the keyboard;
- Output the user if the year is a leap year or not;
- The algorithm to determine if a year is a leap year;
- Variables, if{} else{} statements;
```c
#include "stdio.h"

int main(int argc, char* argv[]) {
    int year, leap;
    printf("This program decide if a year is a leap year!!!\n");
    printf("Enter a year:\n");
    scanf("%d", &year);
    if( (year%4 == 0) && (year%100 != 0) || (year%400 == 0) )
    {
        leap = 1;
    }
    else
    {
        leap = 0;
    }
    if(leap)
    {
        printf("Year %d is a leap year!\n");
    }
    else
    {
        printf("Year %d is NOT a leap year!\n");
    }
    return 0;
}
```
10.5 Loop Control

- `while(){}`
- `do{}while()`
- `for(){}`
The “for” loop

```java
for (expression1; expression2; expression3)
    statement
```

- Single statement
- Multiple statements
Execution procedure

- Step 1: find the value of expression 1;
- Step 2: find the value of expression 2. If it is true, execute the code inside “for” loop; if it is false, goto step (5);
- Step 3: find the value of expression 3;
- Step 4: goto step 2 to continue execution
- Step 5: continue execution following the “for” statement
Expression 1

Expression 2

false

true

Statements inside for loop

Expression 3

Statements after for loop
for (; (c=getchar()) != '\n'; )
{
    printf("%c", c);
}
```c
int i;
for(i=0; i<10; i++) {
    printf(" I is %d\n", i);
}
```
Loop examples

- The factorial
  \[ n! = 1 \times 2 \times \cdots \times (n-1) \times n \]
  ```c
  int fac = 1;
  for(i=1; i<=n; i++) {
    fac = fac * i;
    printf(" THE FACTORIAL OF %d is %d\n", i, fac);
  }
  ```
Loop examples

- To calculate the summation of n integers
  \[ \text{sum} = 1 + 2 + 3 + \ldots + n; \]
The Fibonacci sequence

\[
f_1 = 1 \\
f_2 = 1 \\
f(n) = f(n-1) + f(n-2)
\]

Fibonacci: 1, 1, 2, 3, 5, 8, 13, ...,
int fibo, fibo_1, fibo_2;
int i, n;
printf("Enter an integer:\n")
scanf("%d", &n);
for(i=1; i<=n; i++)
{
    if(i == 1)
    {
        fibo_1 = 1;
        printf("The %dst number in Fibonacci sequence is: %d\n", i,fibo_1);
    }
    else if(i == 2)
    {
        fibo_2 = 1;
        printf("The %dnd number in Fibonacci sequence is: %d\n", i,fibo_2);
    }
    else
    {
        fibo = fibo_1 + fibo_2;
        fibo_2 = fibo_1;
        fibo_1 = fibo;
        printf("The %dth number in Fibonacci sequence is: %d\n", i,fibo);
    }
}
Break and continue statements

- continue --- stop current loop: skip the rest statements in the loop, do the judgment of executing the next loop.
- break --- terminate the loop
```c
#include "stdio.h"

int main(int argc, char *argv[]) {
    int n;
    for(n=1; n<=200; n++) {
        if(n%3 == 0)
        {
            continue;
        }
        printf("%d", n);
    }
    return 0;
}
```
Nested loops

```c
int i, j;
for(i=0; i<10; i=i+1) {
    printf(" ******\n");
    for(j=0; j<10; j=j+1) {
        printf(" $$$$$$$\n");
    }
}
```

How many times “******” should be printed?
How many times “$$$$$$$$” should be printed?
What are i and j values after the loop?
Nested loops

int i, j;
for(i=0; i<10; i=i+1) {
    printf("******\n");
    for(j=0; j<10; j=j+2) {
        printf("$$$$$$\n");
    }
}

How many times "******" should be printed? How many times "$$$$$$" should be printed? What are i and j values after the loop?
Loop example

- Print the follow:

  * if n = 1

  *

  * if n = 2

  *

  ***

  if n = 3

  *

  *

  ***

  ****** if n = 3

  ***

  *
Errors to Avoid

(1) Putting “=” when you mean “==” in an if or while statement

(2) Forgetting to increment the counter inside a loop
   - If you forget to increment the counter, you get an infinite loop (the loop never ends).

(3) Accidentally putting a “;” at the end of a for loop or if statement, so that the statement has no effect
   - For example:
     ```c
     for (x=1; x<10; x++);
     printf("%d\n",x);
     ```

     only prints out one value because the semicolon after the for statement acts as the one line the for loop executes.
10.6 Array

- What is an array:
  - A collection of a sequence of numbers;
  - All elements in the array belong to the same data type;
  - The elements in an array can be determined by the name of the array and its array index.
Array dimensions

- One dimensional array
  ```
  int a[10];
  double b[20];
  ```
- Two dimensional array
  ```
  int a[10][10];
  float a[5][5];
  ```
- Three dimensional array
  ```
  char a[5][5][5];
  ```
10.6.1 One dimensional array

- How to declare a one dimensional array:
  - data type name of array [constant]
  - The type of array is integer;
  - The name of array is a;
  - The size of array is 10: there are 10 elements in the array

```c
int a[10];
```
Attention:

- The name of array must be letters, numbers, and under score, and **cannot** begin with numbers;
- The name of array is followed by a pair of square brackets NOT parenthesis;
- The array index starts with 0.
Initialize one-dimensional array

- Initialize at declaration
  - static int a[0] = {0,1,2,3,4,5,6,7,8,9};

- Initialize partial elements
  - static int a[10] = {0,1,2,3,4};

- Initialize 0 for all elements
  - static int a[10] = {0,0,0,0,0,0,0,0,0,0}

- Initialize all elements without specifying the size
  - static int a[5] = {1,2,3,4,5}
  - static int a[] = {1,2,3,4,5}
Examples

```c
#include "stdio.h"
#include "math.h"

int main(int argc, char *argv[]) {
    int i, a[10];
    for(i=0; i<10; i++) {
        a[i] = i;
    }
    for(i=9; i>=0; i--)
    {
        printf("%d", a[i]);
    }
}
```
```c
#include "stdio.h"
#include "math.h"

int main(int argc, char* argv[]) {
  int i; f[0] = 1; f[1] = 1;
  for(i=2; i<20; i++) {
    f[i] = f[i-1] + f[i-2];
  }
  for(i=0; i<20; i++) {
    if(i%5 == 0) {
      printf("\n");
      printf("%12d", f[i]);
    }
  }
}
```
# Bubble sorting

Compare two neighboring numbers, move the smaller one ahead of the larger one.

<table>
<thead>
<tr>
<th>10</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```c
#include "stdio.h"
#include "math.h"

int main(int argc, char *argv[]) {
    int a[10];
    int i, j, t;
    printf("Enter 10 integers:\n");
    for(i=0; i<10; i++) {
        scanf("%d", &a[i]);
    }
    printf("\n");
    for(j=0; j<9; j++) {
        for(i=0; i<9-j; i++) {
            if(a[i]>a[i+1]) {
                t = a[i];
                a[i] = a[i+1];
                a[i+1] = t;
            }
        }
    }
    printf("The sorted numbers are:\n");
    for(i=0; i<10; i++) {
        printf("%d ", a[i]);
    }
    return 0;
}
```
Two-dimensional array

float a[3][4];

A two-dimensional array is stored row wise in memory in C language.


A two-dimensional array can be regarded as a kind of special one-dimensional array

   a[0]  ---  a[0][0]  a[0][1]  a[0][2]  a[0][3]
   a[1]  ---  a[1][0]  a[1][1]  a[1][2]  a[1][3]
Multi-dimensional array

float a[2][3][4];

A multi-dimensional array is stored in memory in the order such that the left most index changes the slowest, while the right most index changes the fastest.

```
   a[0][0][0] -> a[0][0][1] -> a[0][0][2] -> a[0][0][3] 
   a[0][1][0] -> a[0][1][1] -> a[0][1][2] -> a[0][1][3] 
   a[0][2][0] -> a[0][2][1] -> a[0][2][2] -> a[0][2][3] 
   a[1][0][0] -> a[1][0][1] -> a[1][0][2] -> a[1][0][3] 
   a[1][1][0] -> a[1][1][1] -> a[1][1][2] -> a[1][1][3] 
   a[1][2][0] -> a[1][2][1] -> a[1][2][2] -> a[1][2][3]
```
Initialization of two-dimensional array

Initialize by lines
  static int a[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};

Initialize all data in one pair of curly brackets
  static int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};

Initialize partial elements
  static int a[3][4] = {{1}, {5}, {9}};
  static int a[3][4] = {{1}, {0,6}, {0,0,11}};
  static int a[3][4] = {{1}, {5,6}};
  static int a[3][4] = {{1}, { }, {9}};

Initialize without specifying the first dimension
  static int a[][][4] = {{1,2,3,4,5,6,7,8,9,10,11,12}};
  static int a[][][4] = {{0,0,3},{ }, {1,10}};
Example program

Compute the transpose of a two-dimensional array

\[ a = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad b = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \]
int main(int argc, char *argv[]) {
    static int a[2][3] = {{1,2,3}, {4,5,6}};
    static int b[2][3], i, j,;
    printf("array a: \
");
    for(i=0; i<2; i++)
    {
        for(j=0; j<3; j++)
        {
            printf("%5d", a[i][j]);
            b[j][i] = a[i][j];
        }
        printf("\n");
    }
    printf(" array b: \
");
    for(i=0; i<3; i++)
    {
        for(j=0; j<2; j++)
        {
            printf("%5d", b[i][j]);
        }
        printf("\n");
    }
    return 0;
}
Character array

Character arrays are used to store multiple characters.

char c[10];
Initialize character array


If the number of characters in the curly brackets are greater than the array size, syntax error will occur;
If the number of characters in the curly brackets are smaller than the array size, the rest elements will be filled with empty character ‘\0’.

The size of a character array can be automatically determined by system.
int main(int argc, char *argv[]) {
    static char c[10] = {'I', ' ', 'a', 'm', ' ', 'a', 'b', 'o', 'y'};
    int i;
    for(i=0; i<10; i++)
    {
        printf("%c", c[i]);
    }
    printf("\n");
}
String in C

- In C language, a string can be handled by a one-dimensional character array.
- The character ‘\0’ is used to identify the end of a string.
- The length of a string is determined by ‘\0’.
- To use a character array to store a string, the size of the array must be greater than the length of the string.
- Attention: to store a string of length N, the actual memory cost is N+1 characters.

“C Program” in memory

| C | P | r | o | g | r | a | m | ‘\0’ |
Initialize a character array using a string

static char c[]={"I am happy");
static char c[]="I am happy";
which is equivalent to
static char c[] = {'I', '', 'a', 'm', '', 'h', 'a',
 'p', 'p', 'y', '\0'};
Input and output of character string

- Input/output each individual character using “%c”
- Input/output the whole string using “%s”.
  ```c
  static char c[] = {“America”};
  printf(“%s”, c);
  ```
- Attention:
  - The output does not include ‘\0’
  - Using “%s” as output format, the output object is the name of the array, not actual elements
    ```c
    printf(“%s”, c[0]); //this is illegal.
    ```
  - If the array size is bigger than the string length, output stops when encounter ‘\0’
  - If there are multiple ‘\0’, output stops when encounter the first ‘\0’
- `scanf` can be used to input a string.
- `scanf` can also be used to input multiple strings, which are separated by an empty space, `\ `.
- **Example 1**
  ```c
  static char str1[5], str2[5], str3[5];
  scanf("%s%s%s", str1,str2,str3);
  Input: How are you?
  ```
- **Example 2**
  ```c
  static char str[13];
  scanf("%s", str);
  Input: How are you?
  ```
String operation

- `puts(character array);`
  - static char str[]="New York\nUSA";
  - puts(str);

- `gets(character array);`
  - gets(str);

Note: `puts(str)` and `gets(str)` can output and input a single string at a time.
- `strcat(character array1, character array2);`
  - Concatenate strings in the two character array, put string2 after string1, and store the concatenated string in character array1).
  - Static char str1[30]="United States of ";
  - Static char str2[]="America";
  - `Printf("%s", strcat(str1, str2));`
  - Output: United States of America

Note: the size of character array1 must be large enough to store the new string.
```c
strcpy(character array1, string2);
- copy a string to a character array.
  static char str1[10], str2[]="America";
  strcpy(str1, str2);

Note:
- character array1 must be large enough;
- Character array1 must be the name of an array, string2 may be the name of an array or a string constant.
  strcpy(str1, "America");
- Assign statements cannot be used to assign a string or a character array to a character array.
  str1={"America"}; str1 = str2;
- strcpy can be used to copy a part of string2 to character array1.
  strcpy(str1, str2, 2);
```
- `strcmp(string1, string2);`
  - `strcmp(str1, str2);`
  - `strcmp(str1, "New York");`
  - `strcmp("California", "Florida");`

  If `string1 = string2`, return 0;
  If `string1 > string2`, return a positive integer;
  If `string1 < string2`, return a negative integer;
- `strlen(character array);`
  To find the string length --- actual string length, not including `\0`;
  ```c
  static char str[10] = {"America"};
  printf("%d", strlen(str));
  ```
- `strlwr(string);`
  - Convert uppercase letters into lowercase letters.
- `strupr(string);`
  - Convert lowercase letters into uppercase letters.
Example program

```c
int main(int argc, char *argv[]) {
    char string[100];
    int i, num=0, word=0;
    char c;
    gets(string);
    for(i=0; (c=string[i])!='\0'; i++) {
        if (c== ' ') word = 0;
        else if(word == 0) {
            word = 1;
            num ++;
        }
        printf("There are %d words in the line \n", num);
    }
    return 0;
}
```
10.7 File Operations

- File declaration:
  ```c
  FILE *fp;
  ```

- File open and close:
  ```c
  fp = fopen(fileName, fileOperator);
  fp = fopen("myFile", "r");
  fp = fopen("myFile", "w");
  ```

- File close:
  ```c
  fclose(fp);
  ```
fprintf and fscanf

- Read from a file with certain format:
  fscanf(file pointer, format, input list);  
  fscanf(fp, "%d %f ", &i, &t);

- Write to a file with certain format:
  fprintf(file pointer, format, output list);  
  fprintf(fp, "%d, %f ", i, t);
#include "stdlib.h"

if( (fp = fopen("fileName", "r")) == NULL )
{
    printf("CANNOT open this file\n");
    exit(0);
}
#include “stdio.h”
#include “math.h”
#include “stdlib.h”

int main(int argc, char *argv[]) {
    int i, studentNumber[10];
    float grade[10];
    FILE *fp;
    printf(“Enter information for 10 student: student number followed by grade:
”);
    for(i=0; i<10; i++) {
        scanf(“%d %f”, &studentNumber[i], &grade[i]);
    }
    if( (fp = fopen(“student.txt”, “w”)) == NULL ) {
        printf(“CANNOT open this file
”);
        exit (0);
    }
    for(i=0; i<10; i++) {
        fprintf(fp, “%d %f \n”, studentNumber[i], grade[i]);
    }
    return 0;
}