1. illegal variable name: 6part, ip-part.   
   ok: ip\_part, part2
2. repeat decleare: int a=5; int a=6;
3. keyword must not be used in the meaning other than defined in the language standard.
4. a comment placed anywhere inside the code is a syntactical equivalent of a space.
5. each variable has the attributes of type, name, value
6. C complier is case-sensitive
7. a file which contains a computer program translated into machine language is called an executable file.
8. int is using32 bits (4 bytes) -2147483678~2147483678:  
   int a;  
   unsigned int a; OR unassigned a;  
   unsigned long int a; OR unsigned long a;  
   unsigned long long int a;
9. data of type of float is a fractional number.  
   float a; a=18/5;  
   float a=18/5.0;  
   float a,b=4/3; a=b;
10. **3 • 10^8:** It reads: “*three times ten to the power of eight*” -> 3E8. that is "scientific notation"
11. printf()  
    %d decimal, %x hexadecimal, %o octal, %c char, %f float(4 bytes) %lf double(8 bytes), %% itself  
    good: printf("The value of seven is: %02d\n", 2); -> 02  
    good: printf("The value of seven is: %.6f\n", 7.0);  
    good: printf("The value of seven is: %f\n", 7.0);  
    good: printf("The value of seven is: %.6f\n", 09.0);  
    good: printf("The value of ten is: %f\n", 9E0);  
    wrong: printf("The value of seven is: %.6f\n", 7);  
    wrong: printf("The value of seven is: %f\n", 7);
12. scanf()  
    scanf("%d", &MaxSheep); &means the variable is currently storing
13. asterik \* is multiplication operator.   
    slash / is divisional operation.   
    plus sign + is addition operator.  
    minus sign - is substraction operator  
    unary minus namely -100  
    unary plus namely +100  
    percentage sign % is remainder operator (modulo process)
14. priority:   
    left to right  
    unary ++ -- + - >> \*/ >> =-  
    parentheses from inner to outside
15. increment operator ++, decrement operator --
16. char type: ASCII code with 0-7 digits  
    'A' 65, 'a' 97, ' ' 32  
      
    c='A' encompassed by apostrophes'' OR  
    c=65, 65 is ASCII code of of 'A'  
    c='\65' octal digits  
    c='\x27' hexadecimal digits  
      
    You can always assign a char value to an int variable;  
    You can always assign an int value to a char variable, but if the value exceeds 255 (the top-most character code in ASCII), you must expect a loss of value;  
    The value of the char type can be subject to the same operators as the data of type int.
17. \n transition to a new line  
    \r return to the beginning of the line  
    \a in alarm  
    \0 Nullus
18. data type conversion:

data of type char or short int will be converted to type int (this is called an integer promotion);

if there is any value of type float in the expression, the other data will be converted to float;

if there's any value of type double in the expression, the other data will be converted to double;

if there's any value of type long int in the expression, the other data will be converted to long int.

1. a digraph && (*ampersand ampersand*) AND

the digraph | | (*bar bar*)

~tilde, ^caret

time for an updated **priority table**. Study it carefully – it’ll come in very handy soon.

|  |  |  |
| --- | --- | --- |
| **1** | !, ~ (type), ++, --, +, -, \*, &, sizeof | **unary** |
| **2** | \*, /, % |  |
| **3** | +, - | **binary** |
| **4** | <<, >> |  |
| **5** | <, <=, >, >= |  |
| **6** | ==, != |  |
| **7** | & |  |
| **8** | | |  |
| **9** | && |  |
| **10** | || |  |
| **11** | =, +=, -=, \*=, /=, %=, &=, ^=, |=, >>=, <<= |  |

'\0' is denoted as terminating tag, known empty character or nul (it has nothing to do with NULL pointer)

2. scanf() and fgets() different

1. difference "" and ''

**Mistake no. 1: use of an uninitialized pointer**

You make this mistake when you use a pointer which hasn’t previously been assigned any useful value and doesn't point to an existing variable or valid memory space. The **compiler is unable to detect** the error, because its nature is revealed at run time only.

Consider the example provided in the editor.

strcpy will use the current value of the ptr pointer to determine the location where the string specified in the second parameter should be copied. However, the ptr variable hasn’t been assigned. strcpy brings with it **trouble**.

#include <stdio.h>

#include <string.h>

int main(void) {

char \*ptr;

strcpy(ptr, "you may get into trouble soon");

puts(ptr);

return 0;

}

#include <stdio.h>

int main(void) {

char \*ptr;

\*ptr = 'C';

printf("%c",\*ptr);

return 0;

}

**Mistake no. 2: exceeding the size of the array**

This happens when you forget what the declared size of your array is. Look at this example in the editor.

See, we told the compiler that the str array will contain ten elements and then recklessly told it to place a much longer string in there.

Your program may finish its work with a message about a memory violation error, although if you’re unlucky, the program will go further, but the results may have little in common with your intentions.

#include <stdio.h>

#include <string.h>

int main(void) {

char str[10];

strcpy(str,"Welcome to Troubleland!");

printf("%s",str);

return 0;

}

**Mistake no. 3: non-terminated strings**

We have a ten-element array of characters. Unfortunately, we haven’t put a null character anywhere inside the array, and therefore that string likely has no end.

The strcat function will persistently look for the end of the string, and it’ll probably find one eventually (there’s a good chance that there is a byte containing a zero somewhere in the memory), but more than likely not within the str array itself.

#include <stdio.h>

#include <string.h>

int main(void) {

char str[10];

int i;

strcat(str,"Bump!");

printf("%s",str);

return 0;

}

**C standard 7.1.1.1**

*A string is a contiguous sequence of characters terminated by and including the first null character.* The term multibyte string is sometimes used instead to emphasize special processing given to multibyte characters contained in the string or to avoid confusion with a wide string. A pointer to a string is a pointer to its initial (lowest addressed) character. The length of a string is the number of bytes preceding the null character and the value of a string is the sequence of the values of the contained characters, in order.

int \*ptr[10]: This is an array of 10 int\* pointers, not as you would assume, a pointer to an array of 10 ints

int (\*ptr)[10]: This is a pointer to an array with 10 int

It is I believe the same as int \*ptr; in that both can point to an array, but the given form can ONLY point to an array of 10 ints

# basic conceptions

## type cast

type cast is a conversion from one type to another.

implicit type conversion done by the compiler:

**bool -> char -> short int -> int ->**

**unsigned int -> long -> unsigned ->**

**long long -> float -> double -> long double**

|  |  |
| --- | --- |
| int x=10;  char y ='a';  x=x+y;  float z=x+1.0 | y implicitly converted to int based on ASCII  x is implicitly converted to float. |
|  |  |
|  |  |

explicit type conversion

|  |  |
| --- | --- |
| double x=1.2;  int sum = (int)x +1; | convert double to int |
|  |  |
|  |  |
|  |  |

# array

## integer array

|  |  |  |  |
| --- | --- | --- | --- |
|  | strlen(a) | sizeof(a) |  |
| int a[5]={1,3,5,7,9};  OR  int a[5];  a={1,3,5,7,9}; | 1 | 5\*4=20 |  |
| int a[]={1,3,5,7,9}; | 1 | 20 |  |
| int a[10]={1,3,5,7,9}; | 1 | 10\*4=40 | values of the first 5 address are 1/3/5/7/9. Other values are assigned as 0.  a[5]=0 |
| int a[10]; | 15 | 40 | declare array but doesn't initialize it. |
| int a[10]={}; | 0 | 40 | Values of all 10 address are assigned as 0. |
| int a[3]= {1,3,5,7,9}; | 1 | 3\*4=12 | Illegal: out of bounds. compile could be ok. but the other elements may be lost, and execution may cause segmentation fault  a[3] may return unpredictable value |
| int a[5]= {1,3,5,7,9};  printf("%d, %d\n", a[0], a[10]); |  |  | Illegal: out of bounds. a[10] will return unpredictable value |

Note: 1. strlen(a) is 1, "a" is equivalent to pointer that address a[0].

|  |  |
| --- | --- |
| int i=2;  printf("int=%d, size=%d\n", sizeof(int),  sizeof(i));  int a[]={0,1,2,3};  printf("size=%d, length=%d\n", sizeof(a),  sizeof(a)/sizeof(a[0]));  a[4]=4;  printf("%d\n", a[4]);  printf("%d\n", a[5]);  printf("size=%d, length=%d\n", sizeof(a),  sizeof(a)/sizeof(a[0])); | output:  int=4, size=4  size=16, length=4  4  6422284  size=16, length=4  Note: It is ok to add value which is out of bounds. The size of an array can't be changed. |
|  |  |
|  |  |

## char array

string is denoted as char array

|  |  |  |  |
| --- | --- | --- | --- |
|  | strlen(a) | sizeof(a) |  |
| char a[5]="ABCDE"; | 5 | 6 | string is denoted as char array |
| char a[]="ABCDE"; | 5 | 6 | same as the above |
| char a[]={'A','B','C','D','E','\0'}; | 5 | 6 | same as the above |
| char a[5];  strcpy(a, "ABCDE"); |  |  | **Here size could not be ignored.** |
|  |  |  |  |
| char a[]={'A','B','\0','D','\0'}; | 2 | 6 |  |
| char a[]=""; | 0 | 1 | '\0' take one character |
| char a[]="\1\n\0"; | 2 | 4 |  |
| char a[]="\0\1\2\3\4"; | 0 | 6 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| char a[3]="ABCDE"; | 3 | 3 | illegal: out of bounds. |
| char a[5];  a="ABCDE"; |  |  | syntax error. compiling failed |

Note: 1. string is encompassed by "" rather than ''.

|  |  |
| --- | --- |
|  | Export |
| char a[]="ABCDE";  printf("%s\n", a); | ABCDE |
| char a[]="ABCDE";  printf("%c\n", a[0]); | A |
| char a[]="ABCDE";  printf("%c\n", a[10]); | Illegal: out of bounds. unpredictable value or NULL |
| char a[]="ABCDE";  printf("%c\n", \*(a+3)); | C |
| char a[]="ABCDE";  printf("%c\n", \*(a+1)); | B printf("%c\n", \*(a++)); cause compiling error. |
| char a[]="AB\0CDE";  strcat(a, 'abc');  printf("%s\n", a); | ABabc |

### '\0'

2. '\0' is denoted as terminator of string. strlen() count number of characters before '\0'. sizeof() is number of character plus by one '\0'.

|  |  |
| --- | --- |
| void f(char \*s){      s[1]='\0';  }  int main(){      char p1[]="ABC", p2[]="XYZ";      f(p1);f(p2);      printf("%d\n",strlen(p1)+strlen(p2));      return 0;  } | output: 2 |
| struct Q{          char S[3];      };      struct S{          struct Q Q;      };      struct S S={'\0','\0','\0'};      S.Q.S[0]='A';      S.Q.S[2]='B';      printf("%d\n", strlen(S.Q.S)); | output: 1 |
| FILE \*f;      char s[]="abcdefgh";      char \*p=s+2;      p[4]='\0';      f=fopen("f","wb");      fputs(s,f);      fclose(f); | output: abcdef= 6 bytes |
|  |  |
|  |  |

### char pointer

|  |  |
| --- | --- |
| char \*p = (char \*) malloc(11\*sizeof(char));  for(int i=0; i<10; i++){  **\*p ='A'+i;**  **printf("%c",\*p);**  **p++;**  }  \*p='\0';  p-=10;  printf("==%s\n", p); | output: ABCDEFGHIJ==ABCDEFGHIJ |
| char \*p = (char \*) malloc(11\*sizeof(char));  for(int i=0; i<10; i++){  **\*p++ ='A'+i;**  **printf("%c",\*(p-1));**  }  \*p='\0';  p-=10;  printf("==%s\n", p); | output: ABCDEFGHIJ==ABCDEFGHIJ |
| char f(char \*n, int m){  **return \*(n+2\*m);**      }      printf("%c\n",f("aAbBcCdD",1)); | output: b  default pointer \*n address the 1st address of string. |
| char f(char \*n, int m){  **return (m+2)[n];**      }      char n[]="aAbBcCdD";      printf("%c%c\n",f(n,1), \*n); | output: Ba |
| char f(char \*n, int m){          return m+2[n];      }      printf("%c\n",f("aAbBcCdD",1)); | output:c |

## 2D array

|  |  |
| --- | --- |
| int a[2][3]={1,2,3,4,5,6};  printf("%d\n", a[1][0]); | 4 |
|  |  |
| int a[2][3]={{1,2,3},{4,5,6}};  printf("%d\n", a[1][0]); | same as the above |
| char a[7][10]={"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};  printf("%c\n", a[1][0]); |  |
| char a[4][20]={"Mary Smith", "James Johnson",  "Patricia Williams", "John Brown"};  for(int i=0;i<4;i++){  printf("%s\n", a[i]);  } | output:  Mary Smith  James Johnson  Patricia Williams  John Brown |
| char \*\*p;  int row=3, col=3;  p = (char \*\*) malloc(row\*sizeof(char \*));  for(int i=0;i<col;i++){  p[i] = (char \*) malloc(col\*sizeof(char));  } | pointer to pointer initialize 3x3 char matrix |
| int row=3, col=3;  char \*p[row];  p[0]=(char \*)malloc(col\*sizeof(char));  p[0][0]='A';  p[0][1]='B';  p[0][2]='C';  p[1]="abc";  printf("%c, %s\n", p[0][0], p[1]); |  |
|  |  |
| int a[2][3];  a={1,2,3,4,5,6}; | Illegal: wrong syntax |
| int a[][]={1,2,3,4,5,6}; | Illegal: wrong syntax |
|  | compare two matrices |

## built-in function

strlen()

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

strcat()

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

strncpy()

|  |  |
| --- | --- |
| const char\* from = "john@openedg.org";  char \*t = (char \*) malloc(20);  strncpy(t, from+5, 11);  printf("%s\n",t); | output: opened.org |
|  |  |
|  |  |
|  |  |

atoi() and itoa()

|  |  |
| --- | --- |
| int i = 123;      int ii =i;      char ic[3];      itoa(ii,ic,10);      printf("int=%d, str=%s\n", i, ic); | itoa(): convert integer to string |
| char a[]="456";      int x = atoi(a);      printf("int=%d, str=%s\n", x, a); | convert string to integer  if "a123b", return 0  if "123b", return 123  if "", return 0 |
| // check if illgal positiv integer      int is\_digits(char \*ip){          //remove empty string          if(\*ip=='\0') return 0;          //remove string with 0,namely "0012"          if(\*ip=='0'&&\*(ip+1)!='\0') return 0;          while(\*ip!='\0'){              // ACSII 0-9: 48-57              if(\*ip<48 || \*ip>57){                  return 0;              }              ip++;          }          return 1;      }        char ip[] = "127.0.0.1";      char sep[] = ".";      char \*token = strtok(ip, sep);      while(token != NULL){          if (is\_digits(token)){              int ip\_part = atoi(token);              if (ip\_part<=255 && ip\_part>=0){                  printf("%s->%d\n", token, ip\_part);              }          }          token = strtok(NULL, sep);      } | check illegal IP address  Note:  1.is\_digits(): check digits  2. strtok() split string  3. atoi(): convert string to integer |
|  |  |

sizeof()

|  |  |
| --- | --- |
| struct Q{          char S[3];      };      struct S{          char S[3];          struct Q Q;      };      struct R{          int i;      };      printf("%d,", sizeof(struct Q));      printf("%d,", sizeof(struct S));      printf("%d\n", sizeof(struct R)); | output: 3,6,4 |
|  |  |
|  |  |
|  |  |

strtok()

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

# pointer

## define

|  |  |
| --- | --- |
| int i;  char i; | integer variable  sizeof(i)=4 |
| int a[10]; | integer array known as 'a' with 10 integers |
| int \*p, i=2;  p=&I; | reference variable known as 'p', which contain address of an integer variable  sizeof(p)=4 |
| int \*p;  int a[]={1,2,3,4,5,6,7,8,9,10};  p=&a[2]; | reference variable known as 'p', which contain address of a[2], of which the value is 3. |
| int \*p[10]; | an array with 10 integer pointers  sizeof(p)=4\*10 |
| int (\*p)[10]  int a[]={1,2,3,4,5,6,7,8,9,10};  p=&a; | reference variable known as 'p', which contain address of an array with 10 integers.  sizeof(p)=4 |
|  |  |
|  |  |
| int a[]; | illegal |

## get value and move

int a[5]={1,3,5,7,9};

int \*p, i, j;

p=&a[0];

precedence: right to left side

|  |  |
| --- | --- |
| **i=\*p++;** | move pointer p from a[0] to a[1]. Assign the value a[0]=1 to i  **equivalent: p=p+1, p+=1, p++, \*(p++)** |
| (\*p)++; | access the value of a[0], and then add 1 to the a[0]  a[10]={2,2,3,4,5,6,7,8,9,10}; |
| **i=\*++p;** | move pointer p from a[0] to a[1], assign value of a[1] to i  **equivalent: \*(++p)**  i=3 |
| **i=++\*p;** | access value of a[0], and add 1 to a[0], assign a[0] to i  **equivalent: ++(\*p)**  a[10]={2,2,3,4,5,6,7,8,9,10}  i=2 |
| i=\*(p+3); | Access value of a[3] and assign the value of a[3]=7 to i.Don’t move pointer |
| i=\*p+3; | Access value of a[0], add by 3 and assign to i=a[0]+3=1+3=4 |
| p=&a[1];  i=\*(p-1)+\*p+\*(P+1) | pointer is to a[1]. Access values of a[0], a[1], and a[2].  Don’t' move pointer. |
|  |  |

if \* and ++/-- precede pointer variable, the precedence are the same, and right-to-side. But if ++/-- follows the pointer variable. the precedence is the last.

## move

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## void pointer

void pointer indicate the absence of type. It can hold address of any type

void pointer can't be dereferenced

|  |  |
| --- | --- |
| int a=10;  char b='A';  **void \*p =&a**  p=&b; | Advantage: void pointer can be reusable.  void pointer holds address of integer variable 'a' and then char 'b'. |
| int \*x =malloc(sizeof(int\*\*3); | malloc() return void \* type, which allows to allocate memory of any data type  Note: c++ can't allow it. |
| int a=10;  void \*p=&a;  **printf("%d", \*(int \*)p);** | declare data type which the pointer references. |
| int a[2] = {1, 2};  void \*ptr = &a;  printf("%d", \*(int \*)ptr+1); | arithmetic of void pointer  output: 2 |
| int a[2] = {1, 2};  void \*ptr = &a;  printf("%d", sizeof(ptr)); | The size of the void pointer in C is the same as the size of the pointer of character type.  output: 4 |
| int a[2] = {1, 2};  void \*ptr = &a;  ptr = ptr + sizeof(int);  printf("%d", \*(int \*)ptr); | ??? output 2 |
| int a=10;  void \*p=&a;  printf("%d", \*p); | Compile Error: void \* can't be dereferenced like other pointers namely int pointer. |

## NULL pointer

|  |  |
| --- | --- |
|  |  |
| int \*p=NULL;  printf("%d", p); | don’t assign pointer with address. default value is 0. |
|  |  |
|  |  |
|  |  |

## pointer in array

By default, the pointer denotes the address of the first element of an array

|  |  |
| --- | --- |
| int a[]={1,2};      int \*p=a;      printf("%d\n", \*p); | output: 1 |
| char a[]="ABC";      char \*p=a;      printf("%c\n", \*p); | output: A |
| void func(char \*p){          printf("%c", \*p);      }      char a[]="ABC";      char \*p=a;      func(p);      func(p+1); | output: AB |
| void func(char \*s, int i){          \*(s+i)='\0';      }      char a[]={'a', 'b', 'c', 'd'};  **func(&a[1],1);**  **//wrong: func(a[1],1);**      printf("%d", strlen(a)); | output:2  Note: "func(a,1);" is ok. output is 1. But "func(a[1],1);" will cause compiling failure. |
| int func(char t[]){          return t[1]-t[0];      }      int i=2;      i -= func("ABDGK"+1);      printf("%d", i); | output:0 |

# flow control

## for loop

### basic

The syntax of a for loop in C programming language is −

for ( init; condition; increment ) {

statement(s);

}

Here is the flow of control in a 'for' loop −

The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.

After the body of the 'for' loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

|  |  |
| --- | --- |
|  |  |
| int i, j=0;  for(i=0;i<4;i++){  j++;  } | standard pattern  i=4, j=4 |
| int i, j=0;  for(i=0;i<4;i++)  j++; | the same as above. remove curly bracket {} if only one statement in for-loop |
| int i=10, j=0;  for(int i=0;i<4;i++){  j++;  } | initialize i within for()  i=10,j=4 |
| int i, j=0;  for( ; ; ){  j++;  } | infinite loop |
| int i=-1, j=3;  for(j>0;j;j--)  i\*=2;  printf("integer: %d\n", i+j); | There are 3 expression: "j>0","j","j--"  execute "j>0" at one time, then execute "j--" and then "j"  export: -8 |
| i=-3, j=0;  for(i++;i++;i++){  j--;  }  printf("%d, %d, %d\n", i,j, i-j); | export: 2 |
|  |  |

### use pointer in for loop

|  |  |
| --- | --- |
| char \*p = (char \*)malloc(11\*sizeof(char));  for(int i=0;i<10;i++){  \*p='A';  printf("%c,", \*p);  p++;  }  \*p='\0';  p-=10;  printf("==%s\n",p); | output: A,A,A,A,A,A,A,A,A,A,==AAAAAAAAAA |
| char s[] = "ABC";  char \*p=s;  **for(;\*p;p++){**  printf("%c,", \*p);  }  p-=strlen(s);  printf("==%s\n",p); | output: A,B,C,==ABC  the same as the above |
| char \*mystrcat(char \*destination, char \*source) {  char \*res;  for(res = destination; \*destination++; ) ;  for(--destination; (\*destination++ = \*source++); ) ;  return res;  }  char s1[]="abc";  char s2[]="XYZ";  char \*p = mystrcat(s1, s2);  printf("%s", p); | output: abcXYZ  verify understanding of pointer. |
|  |  |
|  |  |

### multiple initializations or conditions or steps

|  |  |
| --- | --- |
| for(int i=0,j=0; i<10,j<10; i++){  j+=i;  printf("%d:%d, ",i,j);  } | output: 0:0, 1:1, 2:3, 3:6, 4:10,  Here, "i<10,j<10" is equivalent to " i<10&&j<10" |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## switch

1. branches are scanned in the same order as they are defined.

2. case statement should be with the value rather than an expression

3. break statement should be involved for each case statement or default statement. If not, execution will continue to break statement or the end of switch loop.

4. Execute default if not case conditions. The default statement is not required but it is suggested to. For example: default:break;

|  |  |
| --- | --- |
| int i=3;  switch(i){  case 1: printf("January\n");break;  case 2: printf("February\n");break;  case 3: printf("March\n");  default: printf("wrong\n");  }  printf("%d\n", i); | output:  March wrong |
| i=2,j=0;  switch(i){  case 0: j++;break;  case 2: j++;  case 4: j++;break;  default: j--;  }  printf("i=%d,j=%d\n", i,j); | Here, some break statements are missing. Execute case 2, and then case 4.  output: i=2,j=2 |
| i=6,j=0;  switch(i){  case 0: j++;break;  default: j--;  case 2: j++;  case 4: j++;break;  case 1: j++;  }  printf("i=%d,j=%d\n", i,j); | Here, default is in the middle. Execute default, case 2, and case 4 till break  output: i=6, j=1 |
| int a=1;  switch(a){  puts("output:");  break;  case 1: puts("A");  case 2: puts("B");  } | Here, puts and breaks statements are not executed. switch loop only recognize case and default statements  output: A  B |
|  |  |
|  |  |
|  |  |

# structure

## define a structure

|  |  |
| --- | --- |
|  |  |
| struct STUDENT{  char gender;  char \*name;  int age;  float weight;  } s;  int STUDENT=4;  s.gender='M';  s.name="Cary Howard";  s.age=45;  s.weight=123.6;  printf("%d,%f, %c, %s\n", s.age,s.weight, s.gender,s.name); | define a structure known as STUDENT.  declare a variable 's', of which data type is STUDENT. |
| struct STUDENT{  int age;  float weight;  };  struct STUDENT s;  s.age=45;  s.weight=123.6; | Another approach to define structure |
| struct STUDENT{  int age;  float weight;  };  struct STUDENT s[100];  s[0].age=45;  s[0].weight=123.6;  printf("%d,%f\n", s[0].age,s[0].weight); | define a structure array with the size of 100. |
| struct STUDENT{  char \*first\_name;  char last\_name[10];  };  struct STUDENT s;  s.first\_name = "Howard";  strcpy(s.last\_name, "Hope");  printf("%s,%s\n", s.first\_name, s.last\_name); | two patterns for defining string by pointer and char array. |
| struct STUDENT{  char \*name;  int age;  };  struct STUDENT s= { "Howard Hope", 45};  printf("%s,%d\n", s.name, s.age); | Another pattern to pass value into structure variable |
|  |  |
|  |  |
|  |  |
| struct {  int f1;  } str1;  struct {  char f1;  } str2;  str1.f1 = 32;  str2.f1 = str1.f1; | Two structures can contain fields with the same names – the snippet in the editor is correct. |
| struct STR {  int field;  } Structure;  int STR;  Structure.field = 0;  STR = 1; | name of struct type and integer type are identical. Compiling is ok. but try to avoid it. |
| struct Q{          char S[3];      };      struct S{          char S[3];          struct Q Q;      };  **struct S S;**      printf("%d\n", sizeof(struct S));      printf("%d\n", sizeof(S)); | output: 6 6  Note: "struct S S;" will not cause compiling error. but that approach is not suggested. |

## pointer and structure

pattern: "<pointer>-><field>" or "\*(<pointer>.<field.)"

|  |  |
| --- | --- |
|  |  |
| struct STUDENT{  char \*name;  int age  };  **struct STUDENT \*p;**  **p= (struct STUDENT \*) malloc(sizeof(struct STUDENT));**  **struct STUDENT var ={"good", 45};**  **p=&var;**  **printf("%s,%d\n", p->name, p->age);** | pointer to structure |
| struct STUDENT{  char \*name;  int age  };  **struct STUDENT \*p;**  **p= (struct STUDENT \*) malloc(sizeof(struct STUDENT));**  **(\*p).age=10;**  **(\*p).name="good";**  printf("%s,%d\n", p->name, p->age); | Note: Either \*p.age=10 or \*(p.age)=10 is wrong.  Note: printf("%s,%d\n", \*(p.name), \*(p.age) ); is wrong. |
|  |  |

## union

|  |  |
| --- | --- |
| C code | output |
| union SIZE{  int a;  float b;  } s;  s.a=5;  s.b=3.66;  printf("%d, %f\n", s.a, s.b); | output: 1080704369, 3.660000  variable a and b share the same address. Here value of b cover that of a. |
| union SIZE{  int a;  float a;  } s;  s.a=5;  s.a=3.66;  printf("%d\n", s.a); | error: duplicate variable |
| union SIZE{  int a;  float b;  } s={3.3,6};  printf("%d, %f\n", s.a, s.b); | output: 3, 0.000000  only 3.3 is assigned to a. |
|  |  |
|  |  |
|  |  |
|  |  |

## nested struct

|  |  |
| --- | --- |
| struct BIRTH{  int year;  int month;  int day;  };  struct STUDENT{  char name[20];  int age;  struct BIRTH birth;  };  struct STUDENT s = {"good", 45, {2000,12,2}};  printf("%s: %d\n", s.name, s.age);  s.birth.year=2010;  printf("%d-%d-%d\n", s.birth.year, s.birth.month, s.birth.day); | output:  good: 45  2010-12-2  nested structure |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| struct house{  int number;  struct house \*previous;  struct house \*next;  };  struct house h[4];  int n[]={1,3,5,7};  for(int i=0; i<4; i++){  h[i].number = n[i];  if(i==0){  h[i].previous = NULL;  }else{  h[i].previous = &h[i-1];  }  if(i==3){  h[i].next = NULL;  }else{  h[i].next = &h[i+1];  }  }  printf("###current=%d, next=%d\n",  h[0].number, h[0].next->number);  printf("###current=%d, previous=%d, next=%d\n",  h[1].number,h[1].previous->number, h[1].next->number);  printf("###current=%d, previous=%d\n",  h[3].number,h[3].previous->number); | output:  current=1, next=3  current=3, previous=1, next=5  current=7, previous=5 |
| struct element{  int value;  struct element \*next;  };    int values[10] = { 2, 4, 5, 6, 7, 8, 9, 1, 3, 0};  struct element e[10];  for(int i=9; i>=0;i--){  e[i].value = values[i];  if(i==9){  e[i].next = NULL;  }else{  e[i].next = &e[i+1];  }    }    void func(struct element \*p, int t){  printf("%d\n", p->value);  if(p->value!=t){  func(p->next, t);  }  }  puts("First 5 values");  func(&e[0],7);  puts("First 9 values");  func(&e[0],9);  free(e); | output:  First 5 values  2  4  5  6  7  First 9 values  2  4  5  6  7  8  9 |
|  |  |
|  |  |
|  |  |

# function

## pass arguments into function

|  |  |
| --- | --- |
| void func1(void){  printf("no arugments\n");  } | no arguments |
| void func2(int a, char b, char c[]){  printf("a=%d, b=%c, c=%s\n",a,b, c);  } | pass multiple arguments |
| int main(void){  func()  return 0;  }  void func(void){  } | position of function doesn't matter |
|  |  |

## variable scope

|  |  |
| --- | --- |
| // global variables  int a=1;  void func(int data){  // local variables  int c=3;  printf("func 1(): a=%d\n", a);  a=10;  }  func(a);  printf("%d\n",a); | output:  a=10  a=20  Variable a is not defined in func(). Here a is global variable |
| int a=10;  void func(int data){  int a=2;  printf("func4(): a=%d\n", a);  a=20;  }  func(a);  printf("a=%d\n\n",a); | output:  a=2  a=10  int a in func() is local variable. int a=10 Is global variable. |
| void func(int a){  a++;  }  a=10;  func3(a);  printf("func3, a=%d\n",a); | output: a=10  a is defined as local variable. So a++ can't add global a; |
| void function(int param) {  printf("I've received value %d\n", param);  param++;  }  int main(void) {  int param = 111;  function(param);  printf("variable %d\n", param);  return 0;  } | output:  I've received value 111  variable 111 |
| void func(){  printf("func=%d\n", b);  }  void main(void){  int b=10;  func();  } | compiling error: variable b is defined in main() rather than func() |
|  |  |

## integer in function

|  |  |
| --- | --- |
| int f(void){  }  int i;  i=f();  printf("integer: %d\n", i); | **No return statement: undefined behavior**  return unpredictable value in different system of integer overflow. |
| int f(void){  return 'c';  }  int i;  i=f();  printf("integer: %d\n", i); | output: 99 |
| int f(void){  return "ABC";  }  int i;  i=f();  printf("integer: %d\n", i); | No compiling error but return unpredictable value |
|  |  |
| void f(void){  }  int i;  i=f(); | compiling error |
|  |  |

|  |  |
| --- | --- |
| #include <stdio.h>  void functionx(int \*ptr) {  \*ptr = \*ptr + 100;  }  int main(void) {  int i = 100;  int \*p = &i;  printf("i = %d\n", i);  functionx(p);  printf("i = %d\n", i);  return 0;  } | output:  i=100  i=200  add value stored in \*p; |
| void incr(int \*value) {  **(\*value)++;**  }  int main(void) {  int var = 100;  incr(&var);  printf("var = %d\n", var);  return 0;  } | output: var=101  add value stored in pointer \*p  Note: same as ++\*value or ++(\*value) |
| void incr(int \*value) {  \*value++;  printf("value = %d\n", \*value);  }  int main(void) {  int var = 100;  incr(&var);  printf("var = %d\n", var);  return 0;  } | output:  value = 381708544  var = 100  value is unpredictable value because \*value++ will move \*value. |
|  |  |
|  |  |
|  |  |
|  |  |

## struct in function

|  |  |
| --- | --- |
| struct STR{  int age;  char gender;  char \*name;  };  void func(struct STR s){  printf("%d,%c,%s\n", s.age,s.gender, s.name);  s.age=34;  s.gender ='M';  s.name = "good";  printf("%d,%c,%s\n", s.age,s.gender, s.name);  }  struct STR s={30, 'F', "yes"};  func(s); | pass struct into function |
| struct STR{  int age;  char gender;  char \*name;  };  void func(struct STR \*p){  printf("%d,%c,%s\n", p->age,p->gender, p->name);  p->age=34;  p->gender ='M';  p->name = "good";  printf("%d,%c,%s\n", p->age,p->gender, p->name);  }  struct STR s={30, 'F', "yes"};  func(&s); | pass struct pointer into function |
| struct element {  int value;  };  struct element func1(struct element x){  struct element y = x;  y.value += 10;  return y;  }  struct element z = {10};  struct element a = func1(z);  printf("%d\n", a.value); | output: 20  pass struct as argument into function. The function returns another struct. |
| struct element {  int value;  };  void func2(struct element \*p){  p->value += 10;  }  struct element z = {10};  struct element \*p = &z;  func2(p);  printf("%d\n", p->value); | pass pointer of struct into function. update the same struct due to the address shared. |
| struct element {          int value;      };      struct element \* func(struct element \*p){          p->value = 20;          return p;      }      struct element z = {10};      struct element \*pp = func(&z);      printf("%d, %d\n", pp->value, z.value); | output: 20,20  pass struct pointer into function, and then return this pointer |
| struct element {          int value;      };      struct element \* func(struct element \*p){          struct element  \*pp = (struct element \*)  malloc(sizeof(struct element));          pp->value = 20;          return pp;      }      struct element z = {10};      struct element \*pp = func(&z);      printf("%d, %d\n", pp->value, z.value); | pass struct to function, define a new struct and then return the pointer. |

## array in function

|  |  |
| --- | --- |
| void func(int \*arr, int len){  int i;  for(i=0; i<len; i++){  arr[i] ++;  printf("%d-%d, ", \*arr, \*(arr+i));  }  }  int arr[]={1,2,3};  func(arr, sizeof(arr)/sizeof(arr[0])); | output: 2-2, 2-3, 2-4,  pass array pointer into function |
| void func(int arr[], int len){  int i;  for(i=0; i<len; i++){  arr[i] ++;  printf("%d, ", arr[i]);  }  }  int arr[]={1,2,3};  func(arr, sizeof(arr)/sizeof(arr[0])); | output: 2, 3, 4,  pass array into function |
| int compare\_matrics(int \*m1, int \*m2, int m, int n){          int i, j, equal=2, greater=0, smaller=0;          for(i=0; i<m;i++) {              for(j=0; j<n;j++) {  **int x= \*((m1+i\*n) + j);**  **int y= \*((m2+i\*n) + j);**                  if(x==y){equal=0;}                  else if (x<y){smaller=-1;}                  else if (x>y){greater=1;}                  // printf("%d,%d\n", x, y);                }          }          if(equal==0&&greater==0&&smaller==0){              printf("Both matrices are equal\n");              return 0;          }          else if(equal==2&&greater==1&&smaller==0){              printf("Matrix A are larger\n");              return 1;          }          else if(equal==2&&greater==0&&smaller==-1){              printf("Matrix B are larger\n");              return -1;          }else{              printf("Unknown");          }          return 2;      }      int row=2, col=3;      int a[2][3]={{1,2,3},{4,5,6}};      int b[2][3]={{1,2,3},{4,5,6}};      int c[2][3]={4,5,6,7,8,9};      int d[2][3]={0,0,0,1,2,3};      compare\_matrics(a, b, row, col);      compare\_matrics(a, c, row, col);      compare\_matrics(a, d, row, col); | compare two matrices |
|  |  |

## string in function

|  |  |
| --- | --- |
| void func(char \*str){  for(int i=0; i<strlen(str); i++){  printf("%c", str[i]);  }  for(int j=strlen(str)-2; j>=0; j--){  printf("%c", str[j]);  }  }  char str[]="abcde";  func(str); | output:abcdedcba  pass string pointer into function |
| void func(char \*p){  int i=1;  while(\*p){  printf("%d=%c, ", i, \*p);  i++;  p++;  }  }  char s[20] = "good";  func(s); | output: 1=g, 2=o, 3=o, 4=d, |
| char \* combine\_str(char \*s1, char \*s2){  char \*s = (char \*) malloc(20\*sizeof(char));  int i=0;  while(\*s1&&\*s2){  \*s++=\*s1;  \*s++=\*s2;  i+=2;  printf("%c%c,", \*s1++,\*s2++);  }  \*s='\0';  s-=i;  printf("==%s\n", s);  return s;  }  void func(char \*p){  while(\*p){  printf("%c", \*p++);  }  }  char s1[]="abc";  char s2[]="XYZ";  char \*p = combine\_str(s1,s2);  printf("%s\n", p);  func(p);  //free memory  free(s1);  free(s2);  free(p); | output: 20 |
| int f(char t[]){          return t[0]-t[-1];      }      int i=2;      i-=f("ABDGK"+1);      printf("%d",i); | output:1 |
|  |  |

## main function

|  |  |
| --- | --- |
| #include <stdio.h>  int main(int argc, char \*argv[]) {  int i;  for(i = 0; i < argc; i++)  printf("%s\n", argv[i]);  return 0;  } | F:\C> ./function\_main.exe a b c  F:\C\function\_main.exe  a  b  c |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  int main(int argc, char \*argv[]){    if(argc==2){  float x = atof(argv[1]);  float square = x\*x;  printf("square of %.2f is %.2f", x, square);  }else{  puts("Usage: ./cal\_square.exe <length>");  }  return 0;  } | the function atof(), defined in stdlib.h convert string type of arguments into float type |
|  |  |
|  |  |
|  |  |

## incursive function

|  |  |  |
| --- | --- | --- |
|  |  |  |
| int factorial(int n){          int res = 1;          while(n){              res \*= n--;          }          return res;      } | int factorial(int n){          if(n>1){              return n\*factorial(--n);          }else{              return 1;          }      } | calculate n! |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# File IO

ead below table for the difference between open modes r, r+, w, w+, a and a+ in open() function.

|  | **r** | **r+** | **w** | **w+** | **a** | **a+** |
| --- | --- | --- | --- | --- | --- | --- |
| read | \* | \* |  | \* |  | \* |
| write |  | \* | \* | \* | \* | \* |
| create |  |  | \* | \* | \* | \* |
| truncate |  |  | \* | \* |  |  |
| position at start | \* | \* | \* | \* |  |  |
| position at end |  |  |  |  | \* | \* |

## stream

#include <stdio.h>

because that's where the declaration of the three streams is placed.

The declaration looks as follows:

FILE \*stdin, \*stdout, \*stderr;

The stdin stream: the stdin stream is normally associated with the keyboard, pre-opened for reading and regarded as the primary data source of running programs;

the scanf() function reads the data from stdin by default.

The stdout stream: the stdout stream is normally associated with the screen, pre-opened for writing, regarded as the primary target for outputting data by the running program;

the printf() function outputs the data to the stdout stream.

The stderr stream : the stderr stream is normally associated with the screen, pre-opened for writing, regarded as the primary place where the running program should send information on the errors encountered during its work;

|  |  |
| --- | --- |
| fprintf(stderr,"error"); | output: error |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### fopen() and fclose()

|  |  |
| --- | --- |
| FILE \*f;      f = fopen("file", "w");      fclose(f); | 1. file handle should be pointer  2. path is different between UNIX and Windows  3. fopen() and fclose() should always be used together. |
|  |  |
|  |  |

### fseek(), ftell() and rewind()

**fseek(<stream>, <offset>, <position>)**: move pointer to the beginning or end of the stream or keep current position

**ftell(<stream>)**: return current position relative to the beginning of the stream. Both fseek() and ftell() are used for calculating size of file.

**void rewind(<stream>)**: sets the file position to the beginning of the file of the given **stream**.

|  |  |
| --- | --- |
| FILE \*f;      int i;      f=fopen("f.txt", "wb");      printf("<><%s>\n",f);      int x= fwrite(f,2,1, f);  **i = ftell(f);**      fclose(f);      printf("%d\n", i);      f=fopen("f.txt", "rb");  **fseek(f, 2, SEEK\_END);**  **i = ftell(f);**      fclose(f);      printf("%d\n", i); | output:  2  4  size\_t fwrite(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream) writes data from the array pointed to, by ptr to the given stream. |
| FILE \*f;      int i;      f=fopen("f", "wb");      fclose(f);      f=fopen("f", "rb");      // move pointer to the end      fseek(f, 0, SEEK\_END);      //current value of the position indicator      i = ftell(f);      fclose(f);      printf("%d\n", i); | output: 0  int fseek(FILE \*stream, long int offset, int whence)  offset is number of bytes  SEEK\_EST:beginning  SEEK\_END: end  SEEK\_CUR: current position  long int ftell(FILE \*stream) |
| FILE \*f = fopen("f", "w+b");      fputs("123abc", f);  **rewind(f);**      fputs("3",f);      fclose(f);      int i;      f=fopen("f", "rt");      fscanf(f, "%d", &i);      fclose(f);      printf("%s\n",i); | output: 323  rewind() move pointer the beginning of the file stream |
| FILE \*f = fopen("f", "w+b");      fputs("123abc", f);      rewind(f);      fputs("3",f);      fclose(f);      char i[10];      f=fopen("f", "rt");      fscanf(f, "%s", &i);      fclose(f);      printf("%s\n",i); | output:323abc  compared with the outcome of the previous snippet code. |
|  |  |

## read stream

example:

int c = fgetc(f); or char c= fgetc(f);

char p[20]; p=fgets(p, 20, f); #conain 19 character plust '\0'

char p[20];int i; fscanf(f, "%s %d", p,i);

char p[10];fread(p,5,2,f); #read bytes. so no "\0" is included

### fgetc()

fgetc(<stream>): This function returns the character read as an unsigned char cast to an int or EOF on end of file or error.

|  |  |
| --- | --- |
| char s[20]="ABC";      FILE \*f = fopen("file", "w");      fputs(s, f);      fclose(f);      f=fopen("file", "r");  **int i=fgetc(f);**      printf("%c",i); | output:A  The statement of "int i=fgetc(f);" and " char i=fgetc(f);" are equal |
| char s[20]="ABC";      FILE \*f = fopen("file", "w");      fputs(s, f);      fclose(f);      f=fopen("file", "r");      for(int i=0; i<5;i++){          int a=fgetc(f);          printf("%c,",a);      } | output: A,B,C, ,  return the character or EOF(-1) |
| **char s[20]="AB\0CDE";**      FILE \*f = fopen("file", "w");      fputs(s, f);      fclose(f);      f=fopen("file", "r");      for(int i=0; i<4;i++){          int a=fgetc(f);          printf("%c,",a);      } | output: A,B, , ,  only export A and B because '\0' is met |
| FILE \*f = fopen("file", "w");      fputs("ABC\n", f);      fputs("123", f);      fclose(f);      f=fopen("file", "r");      for(int i=0; i<8;i++){          int a=fgetc(f);          printf("%c",a);      }      fclose(f); | output:  ABC  123  Note: fgetc() captures '\n' |
|  |  |
|  |  |
|  |  |
|  |  |

### fgets()

**char \*p = fgets(<string pointer>, <length in bytes>, <stream>)**: On success, the function returns the same str parameter. If the End-of-File is encountered and no characters have been read, the contents of str remain unchanged and a null pointer is returned.

If an error occurs, a null pointer is returned.

|  |  |
| --- | --- |
| char s[20];      FILE \*f = fopen("file", "w");      int i = fputs("12ABCDE", f);      fclose(f);      f=fopen("file", "r");  **fgets(s,2,f);**      puts(s); | output: 1  return a string with one character |
| char s[20];      FILE \*f = fopen("file", "w");      int i = fputs("12ABCDE", f);      fclose(f);    char \*p=fgets(s,4,f);      printf("%s", p);      fclose(f); | output: 12A  return a string with 3 character |
| char s[20];      FILE \*f = fopen("file", "w");      fputs("123", f);      fclose(f);      f=fopen("file", "r");      for(int i=0; i<4;i++){          fgets(s,2,f);          printf("%s", s);      } | output: 1233  If the End-of-File is encountered and no characters have been read, the contents of str remain unchanged and a null pointer is returned. |
| char s[20]="ABC";      FILE \*f = fopen("file", "w");      fclose(f);      f=fopen("file", "r");      fgets(s,2, f);      printf("%s",s); | output=ABC  the content of the file stream is empty. So the string pointer keeps the original position. |
| f=fopen("filed", "r");      char \*p=fgets(s,2, f);      printf("%s",p); | output: (null)  the file doesn’t exist. return NULL pointer |

### fscanf()

fscanf(): format reading from stream. **int fscanf(FILE \*stream, const char \*format, ...)** reads formatted input from a stream.

fscanf() read string with space

|  |  |
| --- | --- |
| char s[]="Cary 2013 M";      FILE \*f = fopen("file", "w");      fputs(s, f);      fclose(f);      char p[100];      int i;      char n;      f=fopen("file", "r");      fscanf(f, "%s %d %c", p, &i, &n);      printf("<%s> <%d> <%c>\n", p, i, n); | output: <Cary> <2013> <M>  fscanf() can string, integer, float, character from text file. |
| char s[]="Cary 2013M";      FILE \*f = fopen("file", "w");      fputs(s, f);      fclose(f);      char p[100], p2[10];      f=fopen("file", "r");  **fscanf(f, "%s%s", p, p2);**      printf("<%s><%s>\n", p, p2);      fclose(f); | output: <Cary><2013M>  for fscanf(), the default separator is whitespace or \n |
| char s[]="08/5/2010";      FILE \*f = fopen("file", "w");      fputs(s, f);      fclose(f);      int y,m,d;      f=fopen("file", "r");  **fscanf(f, "%d/%d/%d", &m,&d,&y);**      printf("%d-%d-%d\n", y,m,d);      fclose(f); | output: 2010-8-5  read date string |
|  |  |

### fread()

fread(): read bytes from stream. **size\_t fread(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream)** reads data from the given **stream** into the array pointed to, by **ptr**. The total number of elements successfully read are returned as a size\_t object, which is an integral data type. If this number differs from the nmemb parameter, then either an error had occurred or the End Of File was reached.

|  |  |
| --- | --- |
| char s[]="abc";      FILE \*f = fopen("file", "w");      fputs(s, f);      fclose(f);      char p[100];      f=fopen("file", "r");  **fread(p, 3, 1, f);**  **\*(p+3)='\0';**      printf("%s\n", p);      fclose(f); | output: abc  **Note: p is only char pointer. '\0' should be added in order to print it as string type.**  **compare with "fgets(p, 4,1, f);"** |
| char s[]="abcde12345";      FILE \*f = fopen("file", "w");      fputs(s, f);      fclose(f);      char p[10];      f=fopen("file", "r");      fread(p, 4, 1, f);      for(int i=0; i<10; i++){          printf("%c,", \*(p+i));      }      fclose(f);      free(p); | output: a,b,c,d,,P, ,a,,k,  except the first "abcd", the other characters are unpredictable. |
|  |  |
|  |  |

## write to the stream

fputc('a', f);

fprintf() writes data as a string.

fwrite() writes data in bytes.

### fputc()

fputc(): writing one character to the stream

|  |  |
| --- | --- |
| char s[]="abcde12345";      FILE \*f = fopen("file", "w");  **fputc(s[0], f);**      fclose(f);      free(s);        char p[10];      f=fopen("file", "r");      fgets(p, 10, f);      printf("%s", p);      fclose(f);      free(p); | output:a |
| FILE \*f = fopen("file", "w");      fputc(65, f);      fclose(f);      char p[10];      f=fopen("file", "r");      fgets(p, 10, f);      printf("%s", p);      fclose(f);      free(p); | output: A |
| FILE \*f = fopen("file", "w");      fputc("123", f);      fclose(f); | output: unpredictable character |
| FILE \*f = fopen("file", "w");      int a= fputc('a', f);      printf("input = %c\n", a);      fclose(f); | output: input = a |

### fputs()

fputs("<string>", <stream>): writing a string to the stream.  returns a non-negative value, or else on error it returns EOF.

|  |  |
| --- | --- |
| int a= fputs("abc", stdout);      printf("%d", a); | output:abc0  successful return 0 |
| int a= fputs("abc", stdin);      printf("%d", a); | output:-1  failure return -1 |

### fprintf()

fprintf(<stream>, "<string>"):formatted writing to the stream. if successful, return length of input or return negative value

|  |  |
| --- | --- |
| char s[]="ABC";      char \*p="DEF";      int i=123;      char c='\'';      FILE \*f;      f=fopen("f", "wb");  **fprintf(f, "%d %s %s %c", i,s,p,c);**      int a = ftell(f);      printf("%d\n",a);      fclose(f); | output: 13 |
| FILE \*f;      f=fopen("f", "wb");  **int a=123;**  **fprintf(f, "%d", a);**      int b = ftell(f);      printf("%d\n",b);      fclose(f); | output: 3 |
| FILE \*f;      f=fopen("f", "wb");  **char a=123;**  **fprintf(f, "%c", a);**      int b = ftell(f);      printf("%d\n",b);      fclose(f); | output: 1 |
| FILE \*f;      f=fopen("f", "wb");  **char a[]="123";**  **fprintf(f, "%s", a);**      int b = ftell(f);      printf("%d\n",b);      fclose(f); | output=3 |
| int i;      i=fprintf(stdout,"hello!");      printf("%d", i); | output: hello!6 |
| int i;      i=fprintf(stdin,"hello!");      printf("%d", i); | output: -1  can't write stdin stream |
| int i;      i=fprintf(stderr,"123");      printf("%d", i); | output: 1233 |
| fprintf(stdout, '4');  fprintf(stdout, 123); | The two statements are wrong |

### fwrite()

fwrite(): writing bytes to the stream. fwrite() is for binary data.

|  |  |
| --- | --- |
| FILE \*f;      f=fopen("file\_write.txt", "wt");      char s[]="abcdef";      printf("%d\n", sizeof(s));  **fwrite(s, 2,2, f);**      fclose(f); | the string size is 7 bytes, but only 4 bytes, that is "abcd", is exported into the file. |
| FILE \*f;      f=fopen("file\_write.txt", "wt");      char s[]="abcdef";  **fwrite(s, 10,2, f);**      fclose(f); | In the file, fwrite() exports 20 bytes, but the string is shorter than 20. Therefore, "abcdef" followed by unpredictable characters in the file. |

## text stream

FILE \*file=fopen(), file is pointer to file or NULL if opening failed.

fprintf() write text data

|  |  |
| --- | --- |
| FILE \*file;      char line[10], lines[2][10];      // read mode, text mode      file = fopen("f:\\C\\file.txt", "rt");      // exist if file not found      if(file == NULL){          exit(0);      }      // read line by line      int i=0;      while(fgets(line, sizeof(line), file) != NULL){          // trim trailing newline character          strcpy(lines[i], strtok(line, "\n"));          printf("%s\n", lines[i]);          i++;      }      fclose(file); |  |
| FILE \*file;      file = fopen("f:\\C\\file\_out.txt", "wt");      char lines[7][10]={"Monday", "Tuesday",  "Wednesday", "Thursday",          "Friday", "Saturday", "Sunday"};      for(int i=0; i<7; i++) {          printf("%s\n", lines[i]);          // write string line by line          fputs(lines[i], file);          fputs("\n", file);      }      fclose(file); |  |
| FILE \*fp1, \*fp2;  fp1 = fopen("file", "r");  fp2 = fopen("file", "r"); | It is ok to open the same file in read mode. but that pattern is not recommended in write mode. |
|  |  |
| FILE \*f;      int i;      f=fopen("f.txt", "wb");      printf("<><%s>\n",f);      int x= fwrite(f,2,1, f);      i = ftell(f);      fclose(f);      printf("%d\n", i);      f=fopen("f.txt", "rb");      fseek(f, 2, SEEK\_END);      i = ftell(f);      fclose(f);      printf("%d\n", i); | output:  2  4  size\_t fwrite(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream) writes data from the array pointed to, by ptr to the given stream. |

## EOF: end of file

[getc()](http://www.cplusplus.com/reference/clibrary/cstdio/getc/)returns EOF when end of file is reached.

[feof()](http://en.wikipedia.org/wiki/Feof) which returns non-zero value only if end of file has reached, otherwise it returns 0. int feof(FILE \*stream); check end of a file

|  |  |
| --- | --- |
| FILE \*f = fopen("f", "wt");      char s[]="abc";      fprintf(f, "%s\n", s);      fprintf(f, "%s", s);      fclose(f);      f=fopen("f", "r");      int ch = getc(f);      while(ch != EOF){          putchar(ch);          ch=getc(f);      }      if(feof(f)){          printf("<End\n");      }      fclose(f); | output:  abc  abc>End |
| printf("%d", EOF); | output: -1 |
|  |  |
|  |  |
|  |  |

## handle errors

|  |  |
| --- | --- |
| FILE \*file;      errno = 0;      file = fopen("f:\\C\\dfile.txt", "rt");      // exist if file not found      if(file == NULL){          printf("errno=%d\n", errno);          printf("%s\n", strerror(errno));          exit(0);      } | output:  errno=2  No such file or directory  errno is none-zero integer type, and represent error  strerror(): contain readable error information. The errno and stderror() are usually used together.  Note: perror() could be equal here. |
| FILE \*file;  file=fopen("c:\\a.txt", "rt");  if(file==NULL){  perror("Error"); }  fclose(file); | output: Error: No such file or directory  **errno=2**  cause: the file doesn't exist  void perror(const char \*str) prints a descriptive error message to stderr. |
| FILE \*file;      file=fopen("f:\\C\\file.txt", "r");      if(file==NULL){          perror("Error");      }else{          fputs("abc", file);          if(ferror(file)){              printf("Error %d: can't write.", errno);          }      }      fclose(file); | output: Error 9: can't write.  **errno=9**  Cause: Can't write to the file which is opened in read-only mode.  ferror() |
| FILE \*file;      file=fopen("f:\\C\\file\_readonly.txt", "wt");      if(file==NULL){          perror("Error");          printf("errno=%d",errno);      }      fclose(file); | output:  Error: Permission denied  **errno=13**  try to open readonly file in written mode. |
|  |  |
|  |  |
|  |  |

## syntax errors

|  |  |
| --- | --- |
| FILE \*file;      file=fopen("f:\\C\file.txt", "r");      if(file==NULL){          perror("Error");      }      fclose(file); | output: Error: Invalid argument  file path in windows should be file=fopen("f:\\C\\file.txt", "r"); |
| FILE \*f;      f=fopen("file\_write.txt", "wt");      float i=65;      fwrite(&i, 1, 1, f);      fclose(f); | unpredictable exports in the file.  fwrite() write binary data in bytes into file. for normal text, fprintf() should be used. |
| FILE \*f;      f=fopen("file\_write.txt", "wt");      int i[]={1,2,3};      printf("%d\n", sizeof(i));      fwrite(i, 4,3, f);      fclose(f); | unpredictable exports in the file. |
| FILE f;  f = fopen("file", "w");  fclose(f); | file handler should be pointer  correct: FILE \*f; |
| FILE \*f = fopen("file", "w");      fputs(f, "ABC");      fclose(f); | wrong arguments for fputs()  correct: fputs("ABC", f); |
|  |  |