Quiz02:

1. In a switch statement, the word `**default**` is optional and operates the same as the last else in an if-else chain.
2. The `**!**` operator is used to change an expression to its opposite state.
3. The logical AND operator is `**&&**`.
4. The use of `**40=age**` in a C program will result in a compiler error.
5. What will the following program print on screen? `**Happy Birthday!**`.

    int age = 0;
    if (age = 40)
        printf("Happy Birthday!");
    else
        printf("Sorry");

6. The `**while**` statement literally loops back on itself to recheck the expression until it evaluates to 0 (becomes false).
7. The logical OR operator is `**||**`.
8. A `**switch**` statement is a specialized selection statement that can be used in place of an if-else chain where exact equality to one or more integer constants is required.
9. A(n) `**sentinel-controlled**` is a condition-controlled loop where one specific value is required to terminate the loop.

    >1. counter-controlled————————计数控制型循环
    >2. condition-controlled——————条件控制型循环
    >3. sentinel-controlled———————标记控制型循环
    >4. input-validation——————————输入验证

10. Omitting the `**test**` expression in a for statement results in an infinite loop.
11. `**sum+=num**` is an accumulating statement.
12. What will the following program print on screen `**Sorry**`.

    int tenure = -5;
    if (tenure + 5)
        printf("Congratulations!");
    else
        printf("Sorry");

13. In IBM-compatible computers, the EOF mark is generated whenever the `**ctrl+z**` keys are pressed simultaneously.
14. Which of the following operators has the lowest precedence? `**||**`.

    ! && || *

15. It is a good practice to terminate the last case in a switch statement with a `**break**`.
16. Which of the following operators has right to left associativity? `**!**`.

    ! || && *

17. In Unix operating systems, the EOF mark is generated whenever the keys are pressed simultaneously.`**ctrl+d**`.

    (1) * IBM——————ctrl+z

(2) * Unix—————ctrl+d

18. In computer programming, data values used to signal either the start or end of a data series are called `**sentinel**`.

19. The second loop of a nested loop is called the `**inner**` loop.

20. A(n) `**input-validation**` loop is a condition-controlled loop that terminates when a value within a valid range is entered.

Quiz03:

1. When a function simply receives copies of the values of the arguments and must determine where to store these values before it does anything else, this is known as a `**pass by value(值传递)**`.
2. A `**stub(存根？)**` is the beginning of a final function that is used as a placeholder for the final function until the function is completed.
3. A function that is called or summoned into action by its reference in another function is a `**called function**`.
4. `**float roi(int yrs, double rate)**` is an example of a function header line.
5. `**~~float roi(int, double);~~ printf("%f", roi(3, amt));**` is an example of a calling statement.
6. The minimum requirement of a `**stub function(存根函数？)**` is that it compiles and links with its calling module.
7. `**void funcA();**` is a prototype of a function that returns no value.
8. Scaling a random number as an integer value between 1 and N is accomplished using the expression `**1 + (int)rand() % N**`.
9. The purpose of a `**function body**` is to operate on the passed data and return, at most, one value directly back to the calling function.
10. The purpose of a `**function header**` is to identify the data type of the value returned by the function, if any, provide the function with a name, and specify the number, order, and type of values expected by the function.
11. `**time(NULL)**` reads the computer’s internal clock time, in seconds.
12. The function `**double fabs(double)(注意，abs()是 int 类型的绝对值函数)**`returns the absolute value of its double-precision argument.
13. To return a value, a function must use a(n) `**return**` statement.
14. The `**rand()**` function can be used to generate a random number in C.
15. The function `**int atoi(string)**` converts an ASCII string to an integer.
16. All C compilers provide `**2 (srand & rand)**` function(s) for creating random numbers, defined in the stdlib.h header file.
17. The items enclosed within the parentheses in a function call statement are called `**~~parameters~~ arguments**` of the function.
18. The argument names in the header line of a function are known as `**parameters / formal parameters / formal arguments**`.
19. The portion of the function header that contains the function name and parameters is known as a `**~~prototype~~ function declarator**`.
20. The function `**double log10(double)**` returns the common logarithm of its argument.

21. `**#include <header-file-name>**` is the correct way to include a header file in your program.
22. A function that calls another function is referred to as the `**calling function**`.
23. The function `**int isalnum(int)**` returns a non-0 number if the argument is a letter or a digit; otherwise it returns a 0.
24. The method for adjusting the random numbers produced by a random-number generator to reside within a specified range is called `**Scaling(缩放)(调整随机数产生 的范围)**`.
25. `**float roi(int, double);**` is an example of a function prototype.
    (1) argument/parameter————————都是参数？

Quiz04:

1. A function can invoke a second function, which in turn invokes the first function; this type of recursion is referred to as `**mutual(互相的)**` recursion.
2. If numAddr is a pointer, `***numAddr**` means the variable whose address is stored in numAddr.
3. In `**run-time**` initialization, initialization occurs each time the declaration statement is encountered.
4. The declaration statement `**int *milesAddr;**` declares milesAddr to be a pointer variable that can store the address of (that is, will point to) an integer variable.
5. To use a stored address, C provides us with an indirection operator, `**~~&~~ ***`.
6. When a function invokes itself, the process is called `**direct**` recursion.
7. `**~~Global variables~~ Local variables**` can only be members of the auto, static, or register storage classes.
8. The four available storage classes are called auto, static, extern, and `**register**`.
9. When the function returns control to its calling function, its `**local auto**` variables "die".
10. A declaration statement that specifically contains the word `**extern**` is different from every other declaration statement in that it does not cause the creation of a new variable by reserving new storage for the variable.
11. A local variable that is declared as `**static**` causes the program to keep the variable and its latest value even when the function that declared it is through executing.
12. A variable that can store an address is known as a(n) `**pointer**` variable.
13. A variable with a `**local**` scope is simply one that has had storage locations set aside for it by a declaration statement made within a function body.
14. A `**global**` variable is one whose storage has been created for it by a declaration statement located outside any function.
15. Coding a function prototype as `**global**` makes sense when the function is used by a number of other functions in a source code file.
16. Functions that call themselves are referred to as `**recursive**` functions.
17. The purpose of the `**extern**` storage class is to extend the scope of a global variable declared in one source code file into another source code file.
18. The variable secnum is `**local to main()**` .

int main(){int secnum;. . .}

19. The `~~storage class~~ **scope**` of a variable defines the location within a program where that variable can be used.
20. Variables created inside a function are `**local**` variables.
21. Where and how long a variable's storage locations are kept before they are released can be determined by the `**storage class**` of the variable.
22. `**global**` variables allow the programmer to jump around the normal safeguards provided by functions.
23. A `**register**` is a high-speed storage area physically located in the computer's processing unit.
24. `**Scope(作用域)**` is defined as the section of the program where the variable is valid or "known".
25. `**register**` variables have the same time duration as automatic variables.

Quiz05:

1. char codes[] = "sample"; sets aside `**7**` elements in the codes array.
2. A(n) `**array**`, is used to store and process a set of values, all of the same data type, that forms a logical group.
3. A two-dimensional array is sometimes referred to as a `**table(表)**`.
4. Any individual element in an array can be accessed by giving the name of the array and the element's position; this position is called the element's `**index/subscript(下 标)**` value.
5. A(n) `**scalar 标量**` variable, is a variable whose value cannot be further subdivided or separated into a built-in data type.
6. A `**one-dimensional array**` is a list of values of the same data type that is stored using a single group name.
7. The term `**val[1][3]**` uniquely identifies the element in row 1, column 3.
8. `~~**char codes[6] = ['s', 'a', 'm', 'p', 'l', 'e'];**~~注意括号！应该是大括号！` shows a correct array initialization statement.
9. In a function prototype that has a two-dimensional argument, the `**row**` size is optional.
10. A `**for**` loop is very convenient for cycling through array elements.
11. `**grade[0]**` refers to the first grade stored in the grades array.
12. `**int grades[5] = {98, 87, 92, 79, 85};**` is a correct statement.
13. For one-dimensional arrays, the offset to the element with index i is calculated as `**Offset = i * the size of an individual element**`.
14. A `**three**`-dimensional array can be viewed as a book of data tables.
15. Any expression that evaluates a(n) `**integer**` may be used as a subscript.
16. `**char codes[] = {'s', 'a', 'm', 'p', 'l', 'e'};**` shows a correct array initialization statement.
17. Each item in an array is called a(n) `**element**` of the array.
18. In C, the array name and index of the desired element are combined by listing the index in `**square braces(方括号)**` after the array name.
19. The individual elements of all global and static arrays are, by default, set to `**0**` at

compilation time.

20. The character `**\0**` is automatically appended to all strings by the C compiler.
21. A(n) `**data structure**` is a data type with two main characteristics:
    (1). Its values can be decomposed into individual data elements.
    (2). It provides an access scheme for locating individual data elements.
22. The initialization of a two-dimensional array is done in `**row**` order.
23. `**int val[3][4];**` declares an array of three rows and four columns.
24. In a one-dimensional array in C, the first element has an index of `**0**`.
25. All `**auto**` arrays are created and destroyed each time the function they are local to is called and completes its execution.

Quiz06:

1. The maximum allowable filename in the DOS operating system is `**8 characters plus an optional period and 3-character extension**`.
2. Line `**5**` in the following section of code checks for the end-of-string character.

```
(1) void strcopy (char string1[], char string2[])
(2) {
(3)         int i = 0;
(4)
(5)         while (string2[i] != '\0')
(6)           {
(7)               string1[i] = string2[i];
(8)               i++;
(9)           }
(10)              string1[i] = '\0';
(11) }
```

3. `**fprintf(stdout,"Hello World!");**` causes the same display as the statement printf("Hello World!");.
4. To write to a binary file you use the `**fwrite()**` function.
5. The actual declaration of the FILE structure is contained in the `**stdio.h**` standard header file.
6. A `**file stream**` is a one-way transmission path that is used to connect a file stored on a physical device, such as a disk or CD-ROM, to a program.
7. The statement `**~~printf("%-25s","Have a Happy Day");~~** 是右对齐，不需要加负号` displays the message Have a Happy Day, right-justified, in a field of 25 characters.
8. The array char message[81]; can be used to store a string of up to `**80**` characters.
9. The string "Good Morning!" is stored in memory using a character array of size `**14**`.
10. Programs that use the gets() routine must include the `**stdio.h**` header file.
11. Typically, the `**~~strcat()~~ sprintf()**` function is used to “assemble” a string from smaller pieces until a complete line of characters is ready to be written, either to the standard output device or to a file.
12. Programs that use the atoi()(参数 str 所指向的字符串转换为一个整数（类型为 int 型）)

routine must include the `**stdlib.h**` header file.

13. The value assigned to the NULL constant is `**'\0'**`.
14. Data that is stored together under a common name on a storage medium other than the computer's main memory is called a `**data file**`.
15. Notice that each file stream name, when it is declared, is preceded by a(n) `**asterisk(星号)**`.
16. `**fscanf()**` reads values for the listed arguments from the file, according to the format.
17. When using #include, the characters `**~~<>~~ “”(找寻的是主程序文件所在目录下的其他.h 文件)**`, tell the compiler to start looking in the default directory where the program file is located.
18. `**Text**` files store each individual character, such as a letter, digit, dollar sign, decimal point, and so on, using an individual character code.
19. A file stream is closed using the `**fclose()**` function.
20. fputc() is the general form of `**putchar()**`.

Quiz07:

1. The expression `***gPtr + 3**` adds 3 to "the variable pointed to by gPtr."
2. The `**1**` in the expression *(gPtr + 1) is an offset.
3. Assuming grade is an array of ten integers, the statement `**grade = &grade[2];**` is invalid.
4. The indirection operator in C is `*****`.
5. After creating two variables as follows: char message1[81] = "this is a string"; char *message2 = "this is a string"; The statement `**message1 = "A newmessage";**` is not valid in C.
6. The header line `**int (*calc)()**` declares calc to be a pointer to a function that returns an integer.
7. A suitable equivalent to the function header calc(int pt[2][3]) is `**calc(int (*pt)[3])**`.
8. When working with pointers, the `**offset(偏移量)**` tells the number of variables that are to be skipped over.
9. You can replace lines 5 and 6 in the following function with `**while (*string1++ = *string2++) ;**`.
   (1) */* copy string2 to string1 */*
   (2) *void strcopy(char string1[], char string2[])*
   (3) *{*
   (4) *int i = 0;*
   (5) *while (string1[i] = string2[i])*
   (6) *i++;*
   (7) *}*
10. When an array is created, the compiler automatically creates an internal `**pointer constant**` for it and stores the base address of the array in it.
11. Consider the declarations int nums[100]; int *nPtr; The statement `**nPtr = &nums[0];**` produces the same result as nPtr = nums;.
12. **(可能是错题)**&grade[3] is equivalent to `**&grade[0] + 3 应该是对的，但题目答案却是**

&grade[0]+3*4`; assume that grade is an array of integers, and each integer requires 4 bytes of storage.

13. The address operator in C is `**&**`.
14. If nums is a two-dimensional integer array, `***(*nums)**` refers to element nums[0][0].
15. `***ptNum++**` uses the pointer and then increments it.
16. If numPtr is declared as a pointer variable, the expression `***(numPtr + i)**` can also be written as numPtr[i].
17. Consider the following declarations of a function that receives an array of integers and finds the element with the maximum value:

    (i) *findMax(int *vals, int numEls)*

    (ii) *findMax(int vals[], int numEls)*

    The address in vals may be modified `**if either (i) or (ii) is used**`.
18. Of the following expressions, `***ptNum++**` is the most commonly used.

    This is because such an expression allows each element in an array to be accessed as the address is "marched along" from the starting address of the array to the address of the last array element.
19. If nums is a two-dimensional integer array, `***nums[1]**` refers to element nums[1][0].
20. int *ptNum = &miles; is `**only valid if miles is declared as an integer variable before ptNum is declared correct**`.
21. Pointers `**can**` be initialized when they are declared.
22. If gPtr is a pointer that points to the first element of an integer array (and each integer requires four bytes of storage), `***(gPtr + 3)**` references the variable that is three integers beyond the variable pointed to by gPtr.
23. If we store the address of grade[0] in a pointer named gPtr (using the assignment statement gPtr = &grade[0];), then, the expression `***gPtr**` references grade[0].
24. Adding `**1**` to a pointer causes the pointer to point to the next element of the original data type being pointed to.
25. In performing `~~**subscript operations(下标操作)**~~ **arithmetic(算术)**` on pointers, we must be careful to produce addresses that point to something meaningful.

Quiz08:

1. In C, a record is referred to as a(n) `**structure**`.
2. Each member of a structure is accessed by giving both the structure name and individual data item name, separated by a `**.**`.
3. `**Dynamic**` memory allocation makes it unnecessary to reserve a fixed amount of memory for a scalar, array, or structure variable in advance.
4. If pt is declared as a pointer to a structure of type Employee, `**(也许是错题)**~~**pt->idNum**~~ ***pt.idNum(这个用法有误啊？！编译都过不去？？)**` refers to the variable whose address is in the pt.idNum variable.
5. `**pointer->member**` is equivalent to (*pointer).member.
6. Structures that are inked together by including the address of the next structure in the structure immediately preceding it are known as `**self-referencing(自指)**` structures.

7. The expression t1.nextaddr->name can be replaced by the equivalent expression `**(*t1.nextaddr).name**`.
8. The operation of removing a structure from a stack is called a `**POP(弹出)**`.
9. The operation of removing a structure from a dynamically linked list is called a(n) `**DELETE**`.
10. If you have declared a structure named Date, you can then make the name DATE a synonym for the terms struct Date, by using the statement `**typedef struct Date DATE;**`.
11. A union reserves sufficient memory locations to accommodate `**its largest member's data type**`.
12. `~~struct {int month; int day; int year;} birth, current;~~ **struct {int month, int day, int year} birth;(注意这个错误例子中每个变量的分隔号)**` is not a valid C statement.
13. `**malloc()**` reserves the number of bytes requested by the argument passed to the function.
14. `**calloc()**` reserves space for an array of n elements of the specified size.
15. The following function cycles through a linked list and displays its contents. Line 3 can be replaced with `**while (contents)**`.

```
1. void display(struct myStruct *contents)
2. {
3.        while (contents != NULL)
4.        {
5.              printf("%-30s\n",contents->name, contents->phoneNum);
6.              contents = contents->nextaddr;
7.        }
8. }
```

16. The function call `**calcNet(emp);**` passes a copy of the complete emp structure to calcNet().
17. A(n) `**linked list**` is a set of structures in which each structure contains at least one member whose value array is the address of the next logically in the list.
18. A `**stack(栈)**` is a special type of linked list in which objects can only be added to and removed from the top of the list.
19. `**Parallel(并行)**` arrays are two or more arrays, where each array has the same number of elements and the elements in each array are directly related by their position in the arrays.
20. Stacks and queues are two special forms of a more general data object called a(n) `**deque(双向队列)**`.

Quiz09:

1. The conditional preprocessor directive `**#ifndef**` means "if not defined".
2. The operator `**?:**` is a ternary operator.
3. ARRAY first, second; is equivalent to the two definitions int first[100]; and int second[100]; if `**the statement typedef int ARRAY[100]; is used before**`.
4. For unsigned integers, each left shift (using the << operator) corresponds to `**multiplication by 2**`.

5. Enumerated lists are identified by the reserved word `**enum**` followed by an optional, user-selected name and a required list of one or more constants.
6. A conditional expression uses the conditional operator, `**?:**`, and provides an alternate way of expressing a simple if-else statement.
7. The equivalence produced by a typedef statement can frequently be produced equally well by a `**#define**` statement.
8. `**&**` bit operations are extremely useful in masking, or eliminating, selected bits from an operand.
9. Explicit values can be assigned to each enumerated constant, with unspecified values automatically continuing the integer sequence from the last specified value. For example, `**enum {Mon = 1, Tue, Wed, Thr, Fri, Sat, Sun};**`.
10. `~~**#define**~~ **#ifndef**` is the most frequently used conditional preprocessor directive.
11. In an arithmetic right shift (using the >> operator), each right shift corresponds to `**division by 2**`.
12. 1 0 1 1 0 0 1 1 `**&**` 1 1 0 1 0 1 0 1 results in 1 0 0 1 0 0 0 1.
13. 1 0 1 1 0 0 1 1 `**|**` 1 1 0 1 0 1 0 1 results in 1 1 1 1 0 1 1 1.
14. The definition REAL val; is `**equivalent to double val; if it comes after typedef double REAL;**`.
15. The `**goto**` statement provides an unconditional transfer of control to some other statement in a program.
16. #define SQUARE(x) x * x val = SQUARE(num1 + num2); results in the equivalent statement `**val = num1 + num2 * num1 + num2;**`.
17. `**^**` is the exclusive OR(异或) operator.
18. The conditional preprocessor directive `**#ifdef**` means "if defined".
19. Upon encountering the command line pgm14.3 three blind mice, the operating system stores it as a sequence of `**four**` strings.
20. The `**&**` operator causes a bit-by-bit AND comparison between its two operands.
21. 1 0 1 1 0 0 1 1 `**^**` 1 1 0 1 0 1 0 1 results in 0 1 1 0 0 1 1 0.
22. typedef can be used to create `**aliases**`.
23. Using even one `**goto**` statement in a program is almost always a sign of bad programming structure.
24. The statement `**typedef double REAL;**` makes the name REAL a synonym for double.
25. In the equivalence statement #define SQUARE(x) x * x, x is `**an argument**`.