1 C 语言数组与指针部分练习参考题

1.1 填空题

1. 已知`float b[4];`那么，b 的类型是（*float ()[4] 类型*），b[2] 的类型是（*float 类型*）。

2. C 语言存储的字符串时以（*'\0'*）结尾。

3. 已知`char str[M];`能存储长度为 12 的字符串，那么，M 的最小值为（*13*）。

4. 已知`char s[] = "I love China";`，那么，数组s中最后的 3 个元素依次为（*'n', 'a', '\0'*）。

5. 假设`sizeof(char)`的值为 1，并且`sizeof(char*)`的值为 8。已知 `char str[] = "China";`那么，`sizeof(str)`的值为（*6*）。

6. 已知数组 `arr` 是一个 `Type` 型数组，`sizeof(arr)`的值为 48，并且`sizeof(Type)`的值为 4，则它的元素个数为（*12*）。

7. 已知`long a[12];`为了使语句 `a[i] = 5;`对数组 `a` 的访问不越界，那么整数 `i` 的取值范围是（*不小于 0 不大于 11*）。

8. 已知`int a[4];`且`sizeof(int)`的值为 4。如果元素 `a[0]`的地址为 $(0028FF10)_{16}$，则元素 `a[3]`的地址为（$(0028FF1C)_{16}$）。

9. 已知`int a[10];`,则代码块`for(int i=0;i<100;i++) a[i]=0;`所犯的错误为（*数组越界*）。

10. 已知二维数组`int a[2][4];`中，`sizeof(int)`的值为 4，并且元素 `a[0][0]`的地址为 $(0028FF00)_{16}$，则元素 `a[1][2]`的地址为（$(0028FF18)_{16}$）。

11. 已知数组`double a[2][3][5];`，则它的元素个数为（*30*）。

12. 已知 `Type a[M][N]`,`sizeof(a)`的值为 `112`,`sizeof(Type)`的值为 `8`，`M`的值为 `2`，则 `N`的值为（*7*）。

13. 代码段`int *p; *p = 12;`所犯的错误为（*写入不确定区域*）。

1.2 选择题（每题恰好有一个选项符合要求）

1. 以下关于`double balance[100];`的说法，不恰当的是（*B*）。

   A 上述数组有 100 个元素

   B 上述数组的下标从 1 开始

   C 上述数组的每个元素均为`doube`型

   D 上述数组占用连续的存储空间

2. 设`sizeof(int)`的值为4,`sizeof(float)`的值为4,`sizeof(char)`的值为1，则以下说法不恰当的是（*D*）。

   A 如果声明`int a[3];`，那么`sizeof(a)`的值为 `12`

   B 如果声明`float b[4];`，那么`sizeof(b)`的值为 `16`

   C 如果声明`char c[5];`，那么`sizeof(c)`的值为 `5`

   D 如果声明`float d[3];`，那么`sizeof(d)`的值为 `14`

3. 已知`int a[3]; int b[5];`，以下语句合法的是（*B*）。

   A `a = b;`

   B `(*a) = 5;`

   C `a += 2;`

   D `a = 3;`

4. 以下代码输出的结果为（*A*）。

```c
int n = 5;
int *p = &n;
int *q = p;
printf("%d,", *p);
*p = 3;
printf("%d", *q);
```

   A `5,3`

   B `5,5`

   C `3,5`

   D `3,3`

5. 已知`char s1[] = "I love China."; char* s2 = "I love China.";` 并且`sizeof(char)` 的值为 1，以下说法不恰当的是 (*C*) 。

   A `sizeof(s1)` 的值为 14

   B `s2++;` 不会使编译器报错

   C `s1 = "Motherland";` 不会使编译器报错

   D `s2 = "Motherland";` 不会使编译器报错

6. 已知`char s1[] = "China"; char* s2 = "China"; char* s3 = "love China";` 并且`sizeof(char)` 的值为 1，以下说法恰当的是 (*D*) 。

   A `sizeof(s1)` 和`sizeof(s2)` 的值相等

   B 引入头文件 `string.h` 后，执行 `strcpy(s1, s3);` 安全

   C `s1++;` 和 `s2++;` 都合法

   D `sizeof(s2)` 和`sizeof(s3)` 的值相等

7. 考虑如下代码段（假设已经引入头文件 `stdio.h`)。

   I. `char s[1024]; scanf("%s", s);`

   II. `char s[1024]; gets(s);`

   设用户在运行过程中输入`"love China"`，那么，下列说法正确的是 (*D*) 。

   A 代码段（I）能完整接收字符串，代码段（II）则不能

   B 代码段（I）和（II）都能完整接收字符串

   C 代码段（I）和（II）都能接收字符串中的空格符

   D 代码段（I）不能完整接收字符串，代码段（II）能

8. 假设 `sizeof(int)`、`sizeof(short)` 和 `sizeof(char)` 的值分别为 4,2,1。已知`int a[4];short b[2];char c[2];`,并且 `c[0]` 的地址为 `0028FF0A`，那么，以下说法可能成立的是 (*B*) 。

   A `b[0]` 的地址为 `0028FF0B`，`a[2]` 的地址为 `0028FF18`

   B `b[1]` 的地址为 `0028FF0E`，`a[2]` 的地址为 `0028FF18`

C b[1] 的地址为 0028FF0E，a[1] 的地址为 0028FF10

D b[1] 的地址为 0028FF0E，a[3] 的地址为 0028FF18

9. 假设 sizeof(int)、sizeof(short) 和 sizeof(char) 的值分别为4, 2, 1。已知int a[4];short b[2];char c[2];，那么，数组 a、b 和 c 所占内存大小的比为（*D*）。

   A 1 : 2 : 8

   B 1 : 2 : 4

   C 4 : 2 : 1

   D 8 : 2 : 1

10. 已知float a; float b[8];，那么，以下说法不恰当的是（*B*）。

    A a 与 b[0] 同类型

    B b 与 b[0] 同类型

    C b[0] 与 b[1] 同类型

    D b[1] 与 (*b) 同类型

11. 已知float *a; float b[8]; 那么，以下说法不恰当的是（*A*）。

    A a 与 b 同类型

    B (*a) 与 b[1] 同类型

    C a 与 &b[2] 同类型

    D &b[0] 与 b 不同类型

12. 已知int a[10];，考虑以下代码

    I. `int *i_address = &a[0];`

    II. `float *f_address = &a[0];`

    那么，以下说法恰当的是（*C*）。

    A 代码段（I）和代码段（II）都安全

    B 代码段（I）和代码段（II）都不安全

    C 代码段（I）安全，代码段（II）不安全

D 代码段（I）不安全，代码段（II）安全

13. 已知`int a[] = {0, 1, 2};`，考虑以下代码

I. `int *address = &a[0]; printf("%d", (*address));`

II. `int *address = a; printf("%d", (*address));`

那么，以下说法恰当的是（*A*）。

A 代码段（I）和代码段（II）都输出 0

B 代码段（I）和代码段（II）都输出 1

C 代码段（I）输出 0，代码段（II）输出 1

D 代码段（I）输出 1，代码段（II）输出 0

14. 已知`int a[] = {0, 1, 2};`，考虑以下代码

I. `int *address = &a[1]; printf("%d", (*address));`

II. `int *address = a + 1; printf("%d", (*address));`

那么，以下说法恰当的是（*A*）。

A 代码段（I）和代码段（II）都输出 1

B 代码段（I）和代码段（II）都输出 2

C 代码段（I）输出 1，代码段（II）输出 2

D 代码段（I）输出 2，代码段（II）输出 1

15. 已知`sizeof(int)`的值为 4，`int a[4];` 并且 `&a[0]` 的值为 `0028FF10`，那么，`&a[3]` 的值为（*D*）。

A `0028FF16`

B `0028FF18`

C `0028FF1A`

D `0028FF1C`

16. 已知`sizeof(short)` 的值为 2，`int b[4];` 并且 `&b[3]` 的值为 `0028FF1E`，那么，`&b[1]` 的值为（*C*）。

A `0028FF16`

B 0028FF18

C 0028FF1A

D 0028FF1C

17. 已知 Type c[4]; &c[1] 的值为 0028FF14 并且 &c[3] 的值为 0028FF1C，那么，sizeof(Type) 的值为 (*D*) 。

A 1

B 2

C 3

D 4

18. 已知sizeof(int) 的值为 4，则下面代码段输出的结果是 (*A*) 。

```
int array[12];
int* addr1 = array; int* addr2 = addr1 + 5;
printf("addr1 - addr2 = \%d,", addr1 - addr2);
addr2+=3;
printf("addr2 - addr1 = \%d", addr2 - addr1);
```

A -5, 8

B 5, -8

C -20, 32

D 20, -32

19. 已知char str[] = "I-love-China."; 考虑以下三个代码段

(I). printf(str);

(II). printf("%s", str);

(III). char* addr = str; printf("%s", addr);

(IV). char* addr = str + 3; printf("%s", addr);

则 (*D*) 。

A 只有代码段（I）和代码段（II）输出 I-love-China.

B 只有代码段（II）和代码段（III）输出 I-love-China.

C 只有代码段（I）和代码段（III）输出 `I-love-China.`

D 代码段（IV）输出 `ove-China.`

20. 已知`char str[] = "I-love-China.";`考虑以下三个代码段

(I). `printf("%c", *str);`

(II). `printf("%s", str);`

(III). `char* addr = str; printf("%s", addr);`

(IV). `char* addr = str + 3; printf("%c", *addr);`

则下列说法不恰当的是（*D*）。

A 只有代码段（I）输出 `I`，代码段（II）输出 `I-love-China.`

B 只有代码段（III）输出 `I-love-China.`，代码段（IV）输出 `o`

C 只有代码段（II）和代码段（III）输出同样的内容

D 只有代码段（I）输出 `I`，代码段（IV）输出 `ove-China.`

21. 已知`int m[2], n; int* addr;`则以下说法不恰当的是（*A*）。

A `m` 与 `addr` 同类型

B `n` 与 `(*addr)` 同类型

C `m[1]` 与 `(*addr)` 同类型

D `&m[1]` 与 `addr` 同类型

22. 已知`int a[2]; int *p, *q; p = &a[1]; q = p;`则以下说法不恰当的是（*C*）。

A 接下来执行 `a[1] = 3;` 与执行 `(*p) = 3;` 等效

B 接下来执行 `a[1] = 3;` 与执行 `(*q) = 3;` 等效

C 接下来执行`int **r = &p;` 与执行`int **r = &q;` 等效

D 接下来执行 `(*p)--;` 与执行 `(*q)--;` 等效

23. 假设考虑以下代码段

```
int a[2]; // 假设sizeof(int)=4
short b[4]; // 假设sizeof(short)=2
char c[4]; // 假设sizeof(char)=1
int* pa = a; short* pb = b; char* pc = c; // 假设三种指针都占8个字节
```

设数组 a 的地址为 0061FE18，则以下说法不可能发生的是（*B*）。

A a、b 和 c 的地址分别为 0061FE18、0061FE10 和 0061FE0C

B pa、pb 和 pc 的地址分别为 0061FE08、0061FDF8 和 0061FDF0

C c、pa 和 pb 的地址分别为 0061FE0C、0061FE00 和 0061FDF8

D b、c 和 pa 的地址分别为 0061FE10、0061FE0C 和 0061FE00

24. 已知`short b[] = {2, 3, 5, 7, 11, 13, 17, 19}; short* addr1 = b + 2; short *addr2 = addr1--;`，则 addr1[3] 和 addr2[4] 的值分别为（*C*）。

A 11，11

B 17，11

C 11，17

D 17，17

25. 下面代码的输出为（*B*）。

```
int n = 3;
int*addr1 = &n;
int*addr2 = addr1;
(*addr2) = 5;
printf("%d,%d,%d" *addr1, *addr2, n);
```

A 3，5，5

B 5，5，5

C 5，5，3

D 3，5，3

26. 下面的代码段执行后，数组 a 的各个元素依次为（*D*）。

```
int a[] = {2, 3, 5, 7};
int* addr1 = &a[1];
int* addr2 = addr1;
(*addr2) = 8;
--addr1;
(*addr2) = 6;
(*(addr1 + 2)) = 9;
```

A 2，6，5，9

B 6，3，5，9

C 6，3，9，7

D 2，6，9，7

27. 已知代码段`int n; int *p = &n; int **q = &p;`，则以下说法不恰当的是（*A*）。

A `*p` 与 `&n` 同类型，`q` 与 `&p` 同类型

B `q` 与 `&p` 同类型，`p` 与 `&n` 同类型

C `*q` 与 `p` 同类型，`*p` 与 `n` 同类型

D `**q` 与 `n` 同类型，`*q` 与 `&n` 同类型

28. 已知代码段`int n = 7; int *p = &n; int **q = &p;` 则以下说法不恰当的是（*D*）。

A `*p` 的值为 `7`

B `**q` 的值为 `7`

C `*q` 的值与 `&n` 的值相等

D `&p` 的值与 `&q` 的值相等

29. 已知代码段`int a[3];` 则 `&a`、`(*a)` 和 `a[0]` 的类型分别是（*D*）。

A `int`型、`int*` 型和`int**` 型

B `int()[3]` 型、`int*` 型和`int`型

C `int(*)[3]` 型、`int`型和`int*` 型

D `int(*)[3]` 型、`int`型和`int`型

30. 已知`float b[5];`，则以下代码段符合语法规定的是（*D*）。

A `float (addr)[5] = &b;`

B `float (addr)[] = &b;`

C `float (*addr)[] = &b;`

D `float (*addr)[5] = &b;`

31. 已知`float b[5]; float (*addr)[5] = &b;`，则以下说法恰当的是（*C*）。

    A `addr` 与 `b` 同类型

    B `addr` 与 `*b` 同类型

    C `*addr` 与 `b` 同类型

    D `*addr` 与 `b[0]` 同类型

32. 关于下面的代码段，说法不恰当的是（*D*）。

```
int array[6] = {2, 3, 5, 7, 11, 13};
int (*addr1)[6] = &array;
int *addr2 = array;
```

    A `addr1` 与 `addr2` 数值上相等

    B `*addr2` 的值为 `2`

    C `addr1++;` 导致的地址值增量是 `addr2++;` 导致的地址值增量的 6 倍

    D `addr1[0]` 与 `*addr2` 的类型不同

33. 以下代码段的输出为（*A*）。

```
int a[] = {2, 3, 5, 7, 11, 13, 17, 19};
int(*addr)[8] = &a;
int n = *((*addr) + 3);
printf("\%d", n);
```

    A `7`

    B `11`

    C `13`

    D `19`

34. 已知`int *a[4];`则以下说法不恰当的是（*D*）。

    A `a` 的类型为`int* ()[4]` 型

    B `a[0]` 的类型为`int*` 型

C `*(a+2)` 的类型为`int*` 型

D `&a[1]` 的类型为`int*` 型

35. 已知`float *b[4];` 则以下说法不恰当的是 (*D*) 。

A `&b` 的类型为`float* (*)[4]` 型

B `*b` 的类型为`float*` 型

C `**(b+3)` 的类型为`float`型

D `&(*(b+1))` 的类型为`float*` 型

36. 假设不同类型的指针占用同样大小的存储空间，关于`char* a[4];` 的说法，下列说法不恰当的是 (*D*) 。

A `&a` 的类型为`char* (*)[4]` 型

B 如果`int* b[4];`，则`sizeof(a)` 与`sizeof(b)` 的值总是相等

C 代码`char ch; a[2] = &ch;` 的操作是安全的

D `**(a+1)` 的类型为`char*` 型

37. 已知 `sizeof(char)` 的值为 1，并且 `strlen()` 函数在 C 语言的标准库 `string.h` 中定义，关于下面代码段的说法，不恰当的是 (*D*) 。

```
1 char *a[4];
2 a[0] = "run";
3 a[1] = "this";
4 a[2] = "file";
5 a[3] = "now";
```

A `strlen(a[2] + 1)` 的值为 3

B `printf("%c", *a[1]);` 输出 `t`

C `sizeof(*a[0])` 的值和`sizeof(*a[1])` 的值都等于 1

D `sizeof(a)` 的值为 14 或 18

38. 考虑以下代码段，

(I). `char *e[]={"Rd","Wrt"}; printf("%s", e[1]);`

(II). `char *e[]={"Rd","Wrt"}; printf("%c",(*e[0])+1);`

则 (*D*) 。

A 代码段（I）输出 `Wrt`，代码段（II）输出 `e`

B 代码段（I）输出 `Rd`，代码段（II）输出 `S`

C 代码段（I）输出 `W`，代码段（II）输出 `Rd`

D 代码段（I）输出 `Wrt`，代码段（II）输出 `S`

39. 考虑如下代码段

```
int m, n;
int *p, *q, *r;
p = r = &n; q = &m;
*p = 5; *q = 3;
p = &m; q = &n;
(*p)++; (*q)--; (*r)++;
printf("%d, %d, %d", *p, *q, *r);
```

输出为 (*A*) 。

A `4, 5, 5`

B `4, 3, 3`

C `5, 5, 4`

D `5, 4, 4`

40. 考虑如下代码段

```
int a[] = {5, 6, 7, 8};
int *p = &a[0];
int *q = p;
*q = 2;
p += 3;
*(++q) = 3;
```

则以下说法恰当的是 (*C*) 。

A *p 的值为 7，*q 的值为 8

B *p 的值为 8，*q 的值为 7

C 数组 a 的元素依次为 2，3，7，8

D p - q 的值为 1

41. 考虑如下代码段的执行结果

```
int a[] = {5, 6, 7, 8};
int *p = &a[0];
int *q = p + 2;
*q = 2;
p += 3;
*q-- = 3;
```

则以下说法不恰当的是 (*B*) 。

A p - q 的值为 2

B *p 的值为 8，*q 的值为 3

C 数组 a 的元素依次为 5，6，3，8

D (*p) - (*q) 的值为 2

42. 考虑以下的代码段（I）和代码段（II），以下说法最恰当的是 (*A*)

(I)

```
#include <stdio.h>
int main(){
    int a = 1;
    int *const ptr;
    ptr = &a;
    return 0;
}
```

(II)

```c
#include <stdio.h>
int main() {
    int a = 100;
    const int* ptr;
    ptr = &a;
    (*ptr) = 5;
    return 0;
}
```

A 代码段（I）和（II）都会编译报错；

B 代码段（I）和（II）都不会编译报错

C 代码段（I）会编译报错，代码段（II）不会

D 代码段（I）不会编译报错，代码段（II）会

43. 考虑以下的代码段（I）和代码段（II），以下说法最恰当的是 (*B*) 。

(I)

```c
#include <stdio.h>
int main(){
    int a = 1;
    int *const ptr;
    *ptr = 5;
    return 0;
}
```

(II)

```c
#include <stdio.h>
int main() {
    int a = 100, b;
    const int* ptr = &a;
    ptr = &b;
    return 0;
}
```

A 代码段（I）和（II）都会编译报错；

B 代码段（I）和（II）都不会编译报错

C 代码段（I）会编译报错，代码段不会

D 代码段（I）不会编译报错，代码段（II）会

44. 已知`int x = 4; float y = 5.5; void* p; int* q; float *r;` 考虑以下代码段

I. `p = &x;`

II. `p = &y;`

III. `q = &x; r = &y;`

IV. `q = &y; r = &x;`

则（*B*）。

A 只有代码段（I）和（II）符合语法

B 只有代码段（I）、（II）和（III）符合语法

C 只有代码段（III）和（IV）符合语法

D 四段代码全都符合语法

45. 以下不可以作为假的是（*C*）。

A 指针值 `NULL`

B `0`

C `'0'`

D `'\0'`

46. 考虑以下代码段

I. `int *p = NULL; (*p) = 3;`

II. `int *p; (*p) = 3;`

则（*A*）。

A 代码段（I）和（II）都有导致系统崩溃的风险

B 代码段（I）和（II）都没有导致系统崩溃的风险

C 代码段（I）有导致系统崩溃的风险，代码段（II）没有

D 代码段（I）没有导致系统崩溃的风险，代码段（II）有

47. 已知`int a[4][5];`考虑以下代码段

I. `a[0][0] = 3;`

II. `a[0][5] = 3;`

III. `a[3][6] = 3;`

IV. `a[4][0] = 3;`

则下列说法不恰当的是（*B*）。

A 代码段（I）和（II）都不会越界访问 `a`

B 代码段（I）和（III）都不会越界访问 `a`

C 代码段（I）、（II）和（IV）至少有一个越界访问 `a`

D 代码段（III）和（IV）都会越界访问 `a`

48. 考虑以下代码段，

```
int i, j, a[3][4];
for(i = 0; i < 3; ++i)
   for(j = 0; j < 4; ++j)
      a[i][j] = i * 3 + j * 2 + 1;
```

则下列说法不恰当的是（*C*）。

A `a[0][0]` 的值为 `1`

B `a[1][2]` 的值为 `8`

C `a[1][4]` 的值为 `7`

D `a[2][4]` 的值为 `15`

49. 以下关于二维数组`int a[3][4]` 的存储结构的说法，不恰当的是（*C*）。

A 整个二维数组所占空间必连续

B `a[0]` 和 `a[1]` 各自所占空间必连续

C `a[0][1]` 和 `a[1][1]` 所占空间必连续

D `a[0][2]`、`a[1][2]` 和 `a[2][2]` 的地址服从等差数列

50. 假设sieof(int)的值为4，int a[3][4]，并且 a[0][0]的地址为 0028FEF0，则以下关于它的存储结构的说法，不恰当的是（*D*）。

A sizeof(a[1])的值为16

B sizeof(a[0])的值等于sizeof(a[2])的值

C a[2][0]的地址为 0028FF10

D a[0][1]的地址值比 a[1][1]的地址值小4

51. 以下关于二维数组int a[3][4]的存储结构的说法，恰当的是（*C*）。

A a[1]的最后一个元素为 a[1][4]

B a[0]的类型为int* ()[4]类型

C a[0]、a[1]和 a[2]的地址服从等差数列

D &a[1]的类型为int* (*)[4]类型

52. 以下关于二维数组int a[3][4];的存储结构的说法，不恰当的是（*C*）。

A &a[0]与 &a[0][0]的数值相同，类型不同

B &a与 &a[0]的数值相同，类型不同

C &a与 &a[0][0]的数值不同，类型不同

D &a与 &a[1][0]的数值不同，类型不同

53. 已知int a[3][4];考虑以下代码段，

I. `int (*addr)[4] = &a[0];`

II. `int (*addr) = a;`

III. `int (*addr)[4] = a + 1;`

IV. `int (*addr)[4] = a + 2;`

以下说法恰当的是（*A*）。

A 代码段（III）和（IV）都符合语法且安全

B 代码段（I）和（II）都符合语法且安全

C 代码段（II）和（III）都符合语法且安全

D 至少存在两段代码不符合语法或者不安全