

Advanced pointer

Inst. Nguyễn Minh Huy

Contents



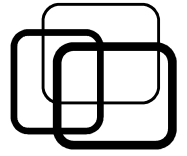
- String manipulations.
- Pointer of pointer.
- Other types of pointers.

Contents



- **String manipulations.**
- Pointer of pointer.
- Other types of pointers.

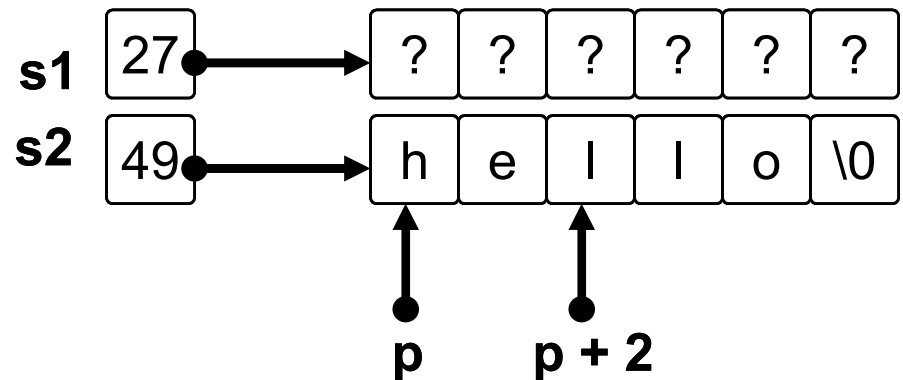
String manipulations



■ Dynamic C string:

- Array of chars + `'\0'`.
- Dynamic string = pointer + memory management.

```
char *s1 = new char[6];  
char *s2 = new char[ ] { "hello" };  
char *p = s2;  
int len1 = strlen( p );    // len1 = 5  
int len2 = strlen( p + 2 ); // len2 = 3  
//...  
delete [ ]s1;  
delete [ ]s2;
```



String manipulations



■ Iterate C string:

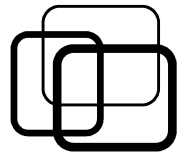
■ Use index:

```
for ( int i = 0; s[ i ] != '\0'; ++i )  
{  
    // Process character s[ i ].  
}
```

■ Use pointer:

```
for ( char *p = s; *p != '\0'; ++p )  
{  
    // Process character *p.  
}
```

String manipulations

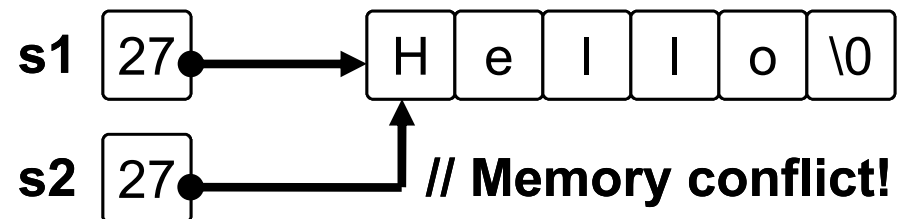


■ Copy string:

- Do not use “=”.

```
char s1[] { “Hello” };
```

```
char *s2 = s1;
```

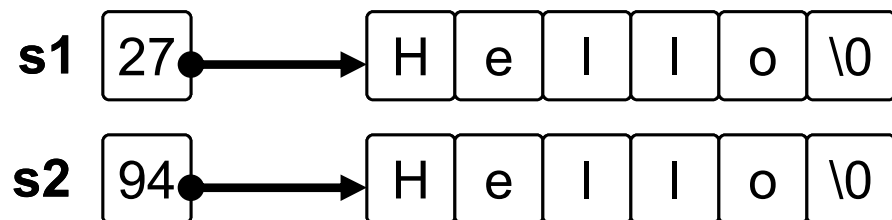


- Two-step copy: **strcpy**(<dest>, <source>).

- Step 1: allocate new string.
- Step 2: copy string content.

```
char s1[] = “Hello”;
```

```
char *s2 = new char[ strlen(s1) + 1 ];  
strcpy( s2, s1 );
```



String manipulations



■ Copy string:

■ One-step copy:

- **strdup**(<source string>).
- Return: new string copied from source string.
- New string must be de-allocated by **free**().

```
char s1[ ] { "Hello" };  
char *s2 = strdup( s1 );
```

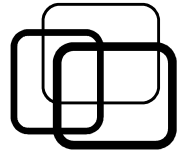
// Same as....

```
// char *s2 = new char[ strlen( s1 ) + 1 ];
```

```
// strcpy( s2, s1 );
```

```
free( s2 );
```

String manipulations



■ Compare string:

- **strcmp**(<string 1>, <string 2>).
- Return: 0 (equal), 1 (greater), -1 (less).
- Compare based on dictionary order.

```
char s1[] { "abc" };  
char s2[] { "abaab" };  
char s3[] { "Abc" };
```

```
int r1 = strcmp( s1, s2 );    // r1 = 1.  
int r2 = strcmp( s3, s1 );    // r2 = -1.  
int r3 = strcasecmp( s3, s1 ); // r3 = 0, GCC compiler.  
int r4 = stricmp( s3, s1 );   // r4 = 0, MSVC compiler.
```


String manipulations

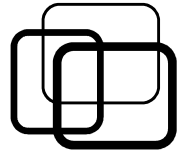


■ Join string:

- **strcat**(<dest>, <source>).
- Copy source to the end of dest.
- Dest string must have enough memory!!

```
char s1[ ] { "Hello" };  
char s2[ ] { "World" };  
int len1 = strlen( s1 );  
int len2 = strlen( s2 );  
char *s3 = new char[ len1 + len2 + 1 ];  
  
strcat( s3, s1 );           // Join s1 to s3.  
strcat( s3, s2 );           // Then, join s2 to s3.  
// More efficient way...  
strcpy( s3, s1 );  
strcpy( s3 + len1, s2 );
```

String manipulations



■ Find sub char/string:

- **strchr**(<source>, <sub>).
- **strstr**(<source>, <sub>).
- Return: pointer to sub in source or NULL (not found).

// Find and skip to find all t in s.

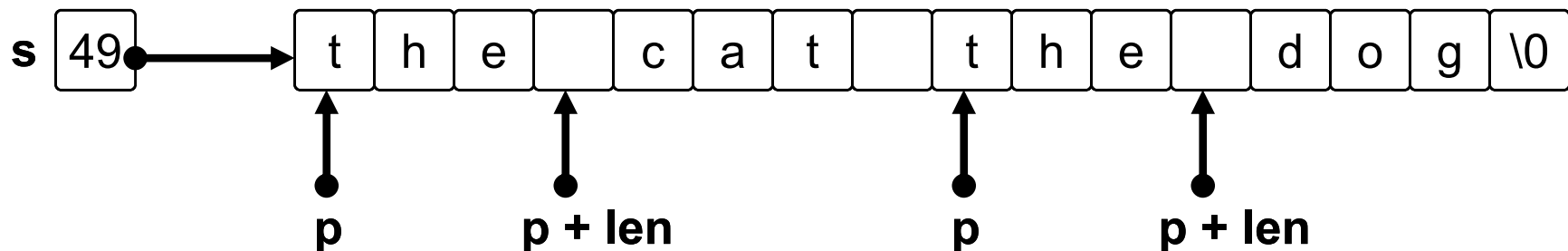
```
char s[] { "the cat the dog" };
```

```
char t[] { "the" };
```

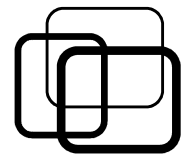
```
int len = strlen( t );
```

```
for ( char *p = s; (p = strstr( p, t )); p += len )
```

```
    printf( "Found at %d\n", p - s );
```



String manipulations



■ Find any of char in/not in set:

■ **strpbrk**(<source>, <set>).

- Return: pointer to first source char in set or NULL (not found).

■ **strspn**(<source>, <set>).

- Return: distance to first source char not in set.

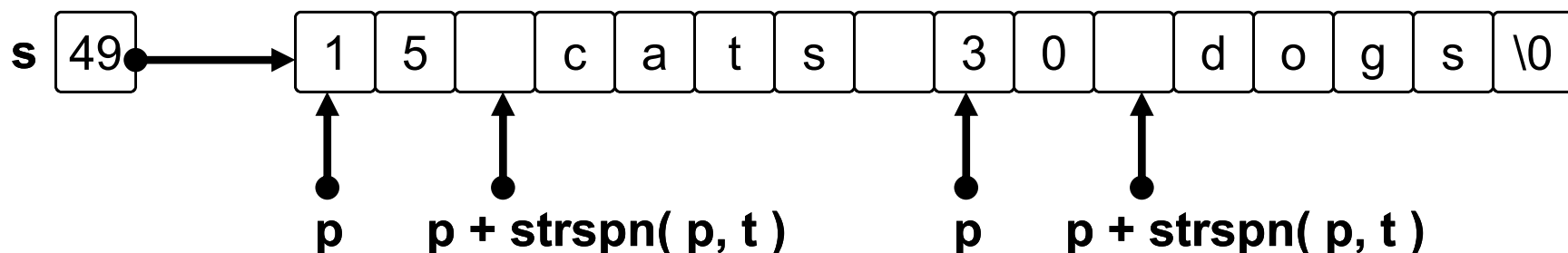
// Find and skip to find all any of t in s.

```
char s[] { "15 cats 30 dogs" };
```

```
char t[] { "0123456789" };
```

```
for ( char *p = s; (p = strpbrk( p, t )) ; p += strspn( p, t ) )
```

```
    printf( "Found at %d\n", p - s );
```



String manipulations



■ Convert string to number:

■ To integer:

- **atoi**(<string>)
- **strtoll**(<string>, <end pointer>, <base>).

■ To float:

- **atof**(<string>).
- **strtod**(<string>, <end pointer>).

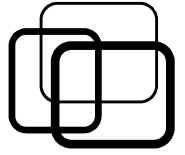
```
char s1[ ] { "123abc" };
char s2[ ] { "xyz" };
int x1 = atoi( s1 );           // x1 = 123
int x2 = atoi( s2 );           // x2 = 0
char *p;
int y1 = strtoll( s1, &p, 10 ); // y1 = 123, p = "abc"
int y2 = strtoll( s2, &p, 10 ); // y2 = 0,   p = "xyz"
```

String manipulations



- Input dynamic string:
 - Declare struct Student:
 - Id: fix-sized 8 chars.
 - Name: variable-sized 50 chars.
 - GPA: float.
 - Write function to input a student from keyboard.

Contents



- String manipulations.
- **Pointer of pointer.**
- Other types of pointers.

Pointer of pointer



■ Address of pointer:

■ Variable has an address.

- int has address int *.

■ Pointer also has an address.

- int * has address type?

■ Pointer of pointer:

- A variable stores address of another pointer.
- Declaration: **<pointer type> * <pointer name>;**

Pointer of pointer



■ Pointer of pointer in C:

■ Declaration:

- Method 1: use `*`.
- Method 2: use **`typedef`**.

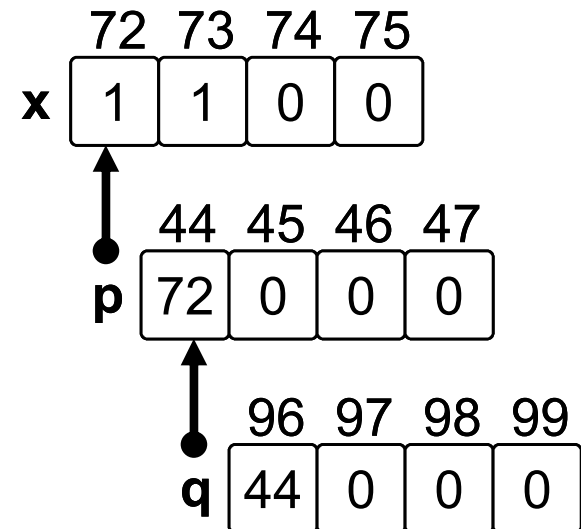
■ Initialization:

- Use **`NULL`**.
- Use **`&`** operator.

```
int x = 257;  
int *p = NULL;  
int **q = NULL;
```

```
p = &x;
```

```
q = &p;
```



Pointer of pointer



■ Pointer of pointer in C:

■ Access memory content:

- 1-level access: operator `*`.
- 2-level access: operator `**`.

■ Passing argument:

- Pass-by-value.
- Pass-by-reference.

➔ Which values are changed in `foo()` ?

```
void foo(int **g, int **&h)
{
    (**g)++; (*g)++; g++;
    (**h)++; (*h)++; h++;
}
```

```
int main()
{
    int a[10];
    int *p = a;
    int **q = &p;
    int **r = &p;

    foo(q, r);
}
```

Pointer of pointer



■ Dynamic matrix:

■ Array of pointers:

- Level-1 pointer is 1-dimensional dynamic array.
- Level-2 pointer is 2-dimensional dynamic array.

```
void inputMatrix(int **&m, int &rows, int &cols) {
```

```
    printf( "Enter rows and cols = ");
```

```
    scanf("%d %d", &rows, &cols);
```

```
    m = new int * [ rows ];
```

```
    for (int i = 0; i < rows; i++) {
```

```
        m[ i ] = new int [ cols ];
```

```
        for (int j = 0; j < cols; j++)
```

```
            scanf("%d", &m[ i ][ j ]);
```

```
    }
```

```
}
```

```
int main()
```

```
{
```

```
    int **m;
```

```
    int rows, cols;
```

```
    inputMatrix(m, rows, cols);
```

```
    delete [ ]m;
```

```
    // Error!! How to fix?!
```

```
}
```

Contents



- String manipulations.
- Pointer of pointer.
- **Other types of pointers.**

Other types of pointers



■ Constant pointer:

- Pointer points to only 1 address “for life”.
- Declaration: `<type> * const <pointer name>;`

```
int x = 5, y = 6;  
int * const p = &x;  
p = &y;      // Wrong.
```
- All static arrays in C are constant pointers.

■ Pointer to constant:

- Memory content pointer points to cannot be changed.
- Declaration: `const <type> * <pointer name>;`

```
int x = 5;  
const int *p = &x;  
*p = 6;      // Wrong.
```

Other types of pointers



■ void pointer:

- Pointer can store address of any types.
- Declaration: **void * <pointer name>**.
- Cast to specific type when accessing content.

```
void printBytes(void *p, int size)
{
    char *q = ( unsigned char * ) p;
    for ( int i = 0; i < size; i++ )
        printf( "%d ", q[ i ] );
}
```

```
int main()
{
    int    x = 1057;
    double y = 1.25;

    printBytes(&x, 4);
    printBytes(&y, 8);
}
```

Other types of pointers



■ Function pointer:

■ Function address:

- Functions are also stored in memory.
- Each function has an address.

■ Function pointer stores address of function.

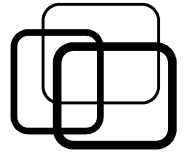
■ Declaration:

```
<return type> (* <pointer name>) (<arguments>);  
typedef <return type> (* <alias>) (<arguments>);  
<alias> <pointer name>;
```

■ Functions have same address type if:

- Same return type.
- Same arguments.

Other types of pointers



■ Function pointer:

```
typedef int (*Operator)(int a, int b);
```

```
int add(int u, int v)
{
    return u + v;
}
```

```
int mul(int u, int v)
{
    return u * v;
}
```

```
int calculate(int u, int v, Operator p)
{
    //  $u^3$  operator  $v^2$ .
    return p(u*u*u, v*v);
}
```

```
int main()
{
```

```
    int x = 5;
    int y = 6;
```

```
    Operator p = add;
    int r1 = p(x, y);
```

```
    p = mul;
    int r2 = p(x, y);
```

```
    int r3 = calculate(x, y, add);
```

```
}
```

Other types of pointers



■ Pointer to fix-sized memory:

■ Address of static array:

- What address type of static array?

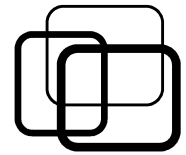
```
int  a[ 10 ];  
int  *p = a      // p and a store address of a[ 0 ].  
??? q = &a;
```

■ Pointer to fix-sized memory:

- Pointer stores address of static array.
- Declaration:

```
<array type> (*<pointer name>)[<array size>];  
int  a[ 10 ];  
int  ( *p )[ 10 ] = &a;  // p points to 10-element array.
```


Other types of pointers

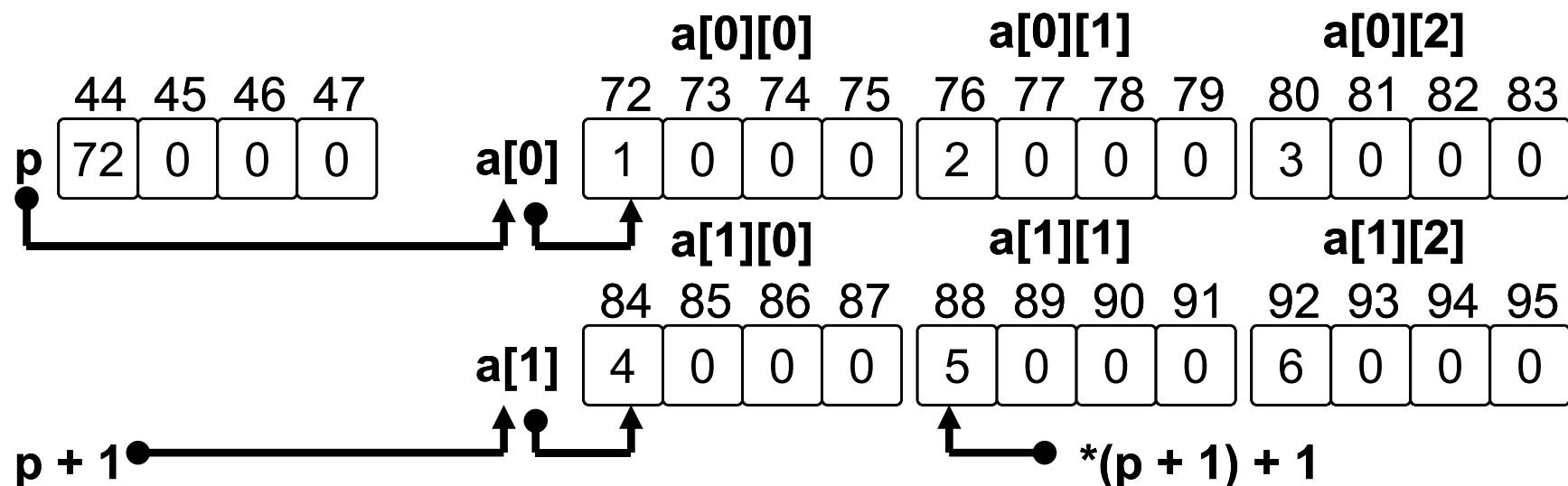


■ Pointer to fix-sized memory:

■ Static 2-D array in C:

- Is pointer to fix-sized 1-D array.
- Stores address of the first row.

```
int a[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };  
int ( *p )[ 3 ] = a; // a = &a[ 0 ].  
printf( "%d\n", *( *(p + 1) + 1 ) );
```



Other types of pointers



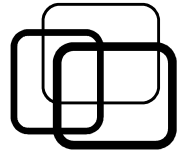
■ Pointer to fix-sized memory:

■ Passing static 2-D array to function:

- Not passing whole array.
- Only passing address of first row.

```
void printMatrix(int a[ ][20], int rows, int cols) { // pass &a[ 0 ].
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++)
            printf("%d ", a[ i ][ j ] );
        printf("\n");
    }
}

int main() {
    int a[10][20];
    printMatrix(a, 10, 20);
}
```



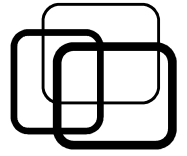
■ String manipulations:

- Dynamic string = pointer + memory management.
- Copy string: strcpy, strncpy.
- Join string: strcat (not efficient).
- Compare string: strcmp, strcasecmp, stricmp.
- Find sub char/string: strchr, strstr.
- Find any of char in/not in set: strpbrk, strspn.
- Convert to number: atoi, strtoll, atof, strtod.

■ Types of pointers:

- Different types → different address types.
- Each address type stored by one pointer type.

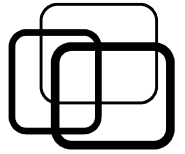




■ Types of pointers:

- Pointer of pointer → stores address of pointer.
- Constant pointer → stores constant address.
- Pointer to constant → stores address of constant.
- void pointer → stores address of any types.
- Function pointer → stores address of function.
- Pointer to fix-sized memory → stores address of static array.





■ Practice 8.1:

Write C/C++ program to find and replace all as follow:

- Enter a sentence of words S.
- Enter word to find F, and replacing word R.
- Find and replace all F in S with R.
- Print the result sentence.

Input format:

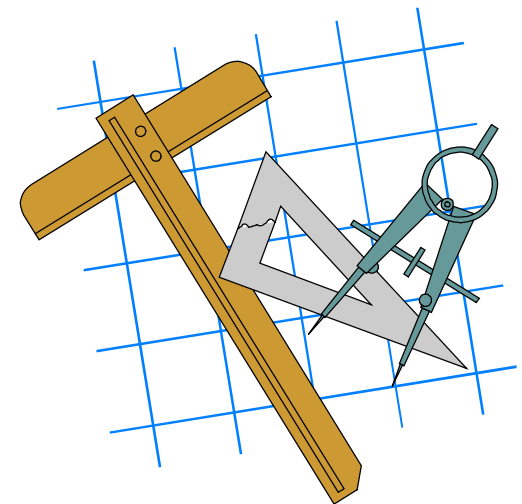
Enter a sentence = the good, the bad, the ugly

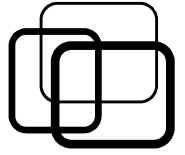
Enter find word = the

Enter replacing word = (a very)

Output format:

(a very) good, (a very) bad, (a very) ugly





■ Practice 8.2:

Write C/C++ program to increase numbers in sentence as follow:

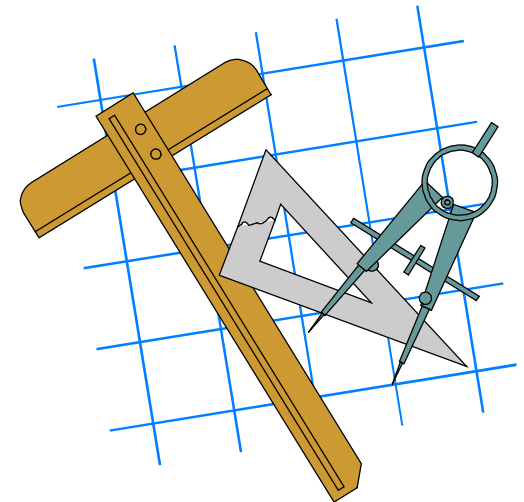
- Enter a sentence.
- Find all numbers in the sentence and increase each one to 1.
- Print the result sentence.

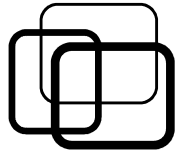
Input format:

Enter a sentence = there are 9 cats and 19 dogs.

Output format:

there are 10 cats and 20 dogs.



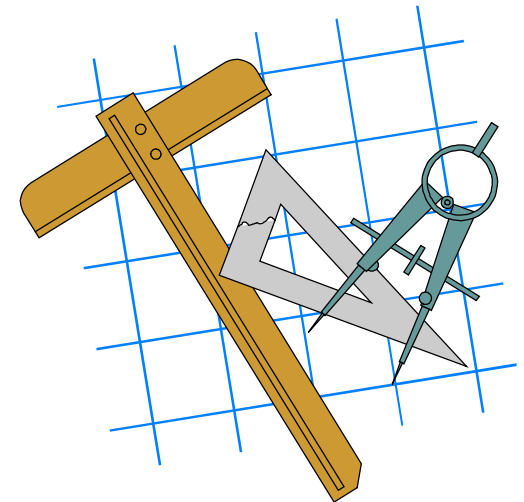


■ Practice 8.3:

Write C/C++ program to input a paragraph with arbitrary length:

- Enter a long paragraph until input '.' and new line.
- Print the paragraph.

Note: use dynamic string.



Practice



■ Practice 8.4:

Given static 2-D array as follow:

```
int m[4][6];
```

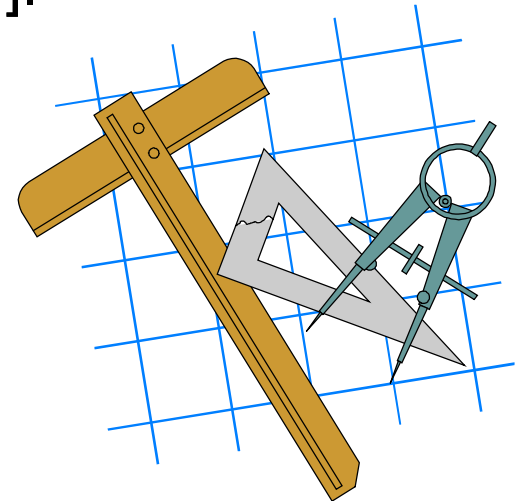
What types of addresses of the following variables?

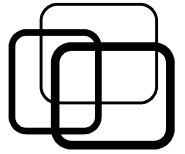
a) `m[1][3]`.

b) `m[0]`.

c) `m`.

Write code to access `m[2][4]` without using operator `[]`.





■ Practice 8.5:

Given the following C code:

```
void initialize(double **p)
{
    for (int i = 0; i < 5; i++)
        *(p + i) = new double[ i + 1 ];
}
```

Answer the following questions:

- How many bytes are allocated at each line of `main()` ?
- How memory are allocated by `initialize()` function in `main()` ?
- Write `release()` function to avoid memory leak.

```
int main()
{
    double *p[10];
    initialize(p + 3);
    release(p);
}
```

