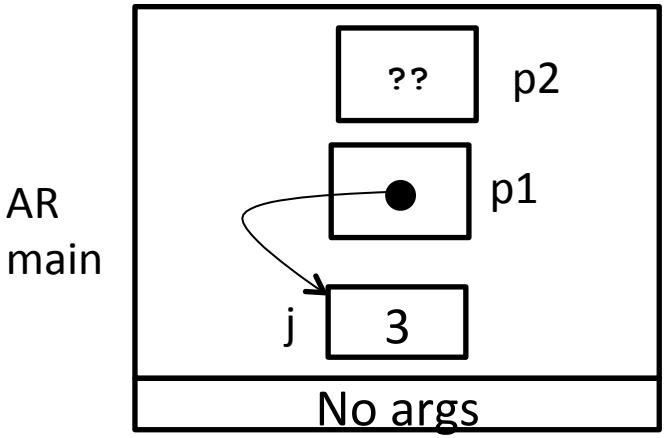
Pointer to Pointers

Pointer to Pointer

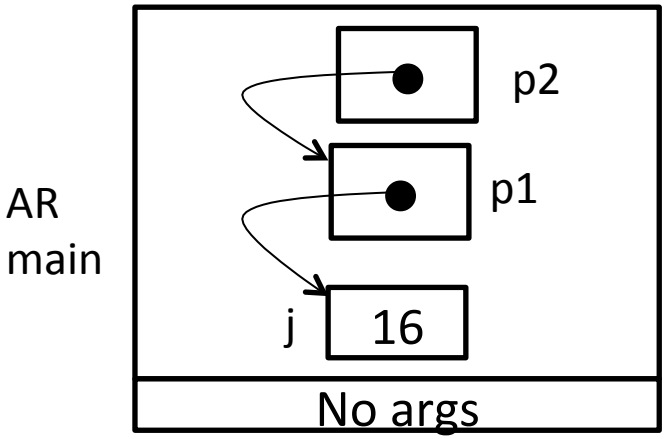
- A pointer that can hold the address of another pointer
- Example:

```
int main () {
   int j, *p1, **p2 ;

   p1 = &j;

   *p1 = 3;

   // point one

   p2 = &p1;

   **p2 = 16;

   // point two

    return 0;
}
```

Point 1



Point 2



This swap does not work properly

```
void main()
{
    int a = 23, b = 40,
    *p1 = &a,  *p2 = &b;
    swapPointers(p1, p2);
    cout << "a =  "<<*p1
    << " b = "<< *p2;
}
```

```
void swapPointers(int *x, int *y)
{
   int *temp;
   temp = x;
   x = y;
   y = temp;
}
```

- Output: a = 23 b=40
- Look at function definition and explain why swapPointer did not work?

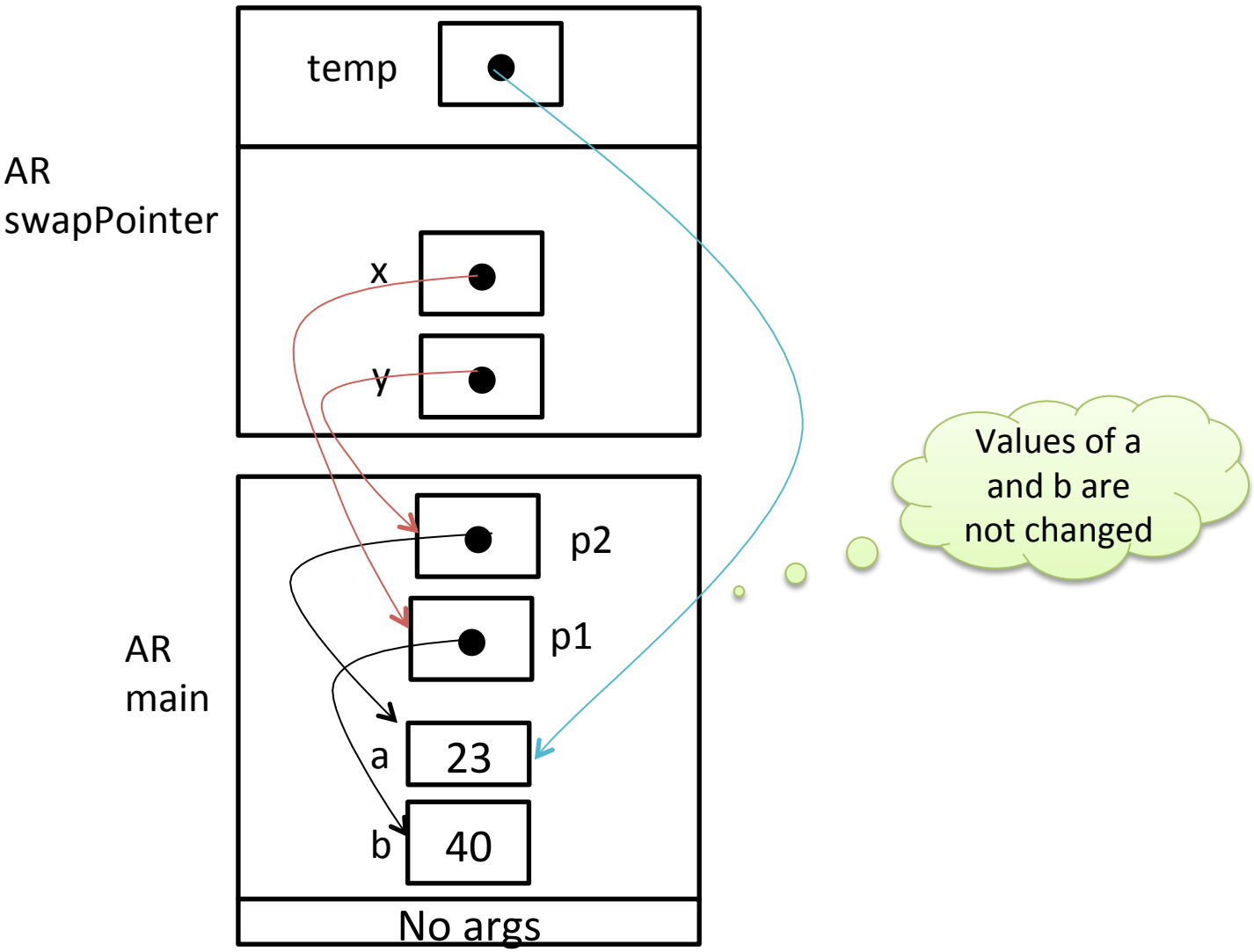
Solution:

```
#include <iostream.h>
void main()
{
 int a = 23, b = 40,
 *p1 = &a,  *p2 = &b;
 swapPointers (&p1, &p2);
 cout << "a =  "<<*p1
 << " b = "<< *p2;
}
```

Output: a = 40 b=23

```
void swapPointers(int **x,  int **y)
{
   int *temp;
   temp = *x;
   *x = *y;
   *y = temp;
    // point one
}
```

AR at Point One in the Function SwapPointers



Array of Pointer

Array of Pointer

- The same way that we can have an array any type, we can declare an array of any type of pointers
- Examples:

```
int  *a[5];
double *d[3];
char* c[4];
MyString  *x[9];
string *z[5];
```

- In the above example we should read **a** is an array of 5 integer pointers. You can look at this statement in different way to make it easier to read it properly:

```
int*    a[5];  // an array of five elements of type int*
```

- Arrays of pointers are used to solve many practical problems in C and C++.

Precedence of operators * and []

- Asterisk, * is an operator that is used in C/C++ as both unary and binary operator:
  - Multiplication is a binary operation
  - Pointer notation and dereferencing of pointers are unary operators.
- In declaration of array of pointers two operators are involved:
  - [] that is a binary operator
  - * that is unary operator
- The operator [ ], has higher precedence than unary * operator. Therefore applying the order of precedence for [] and * implies to read the following declaration as: a is an array of 3 pointers:

```
int *a[3];
```

Using Array of Pointers to C-strings

```
#include<iostream.h>
 int main()
{
   const char *  p[3];     // p is an array of char* pointers
   p[0] = "XYZ";           // first element point to “XYZ on the static area
   p[1] = "KLM";
   p[2] = “ABC”

   cout << p[1] <<endl;        // p[1] is a pointer to “KLM’ – prints: KLM
   cout << *p[1]<< endl;       // *p[1] is a char – prints: K
   cout  << **p << endl;       // *(*p) is a char – it is same as *p[0]: X
   cout   << *(p+1) << endl;   // same as p[1] – prints: KLM
}
```

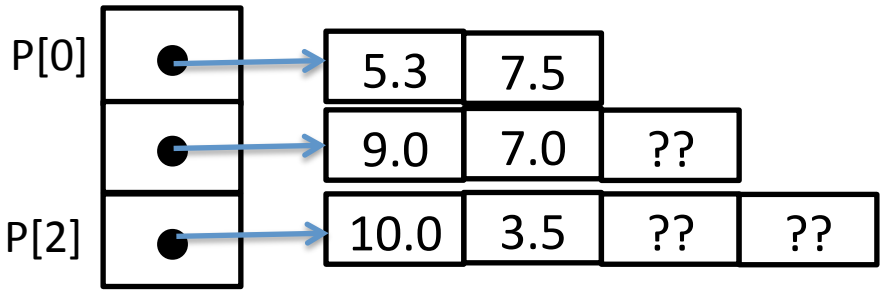
Initialization of Array of Pointers

- We can initialize an array of pointers in different forms:

```
char *arr1[ ] = { "ABC", "XYZ"};

char *arr2[3 ] = { "ABC", "XYZ"}; // third element is a null pointer

char *arr3[ ] = { "ABC", "XYZ"};

const char *s = "KLM";

char *arr4[ ] = {s , &s[1]};          // arr4[1] points to "LM"
```

Dereferencing Array of Pointers with Different Type of Notations

```
int main()
{
   double *p[3];

   int i;
   for (i = 0; i < 3; i++)
       p[i] = new double[i+2];

   **p = 5.3;
   p[1][0] = 9.0;
   p[1][1] = 7.0;
   p[2][1] = 3.5;
   *p[2] =  10.0;
   (*p)[1] = 7.5;
   return 0;
}
```



Example:

```
char *sort(char **s, int n)
{
   for(int i = 0; i < n-1; i++)
     for(int j = i + 1 ; j < n; j++)
       if(strcmp(s[i], s[j]) > 0) {
          char* tmp = s[i];
          s[i] = s[j];
          s[j] = tmp;
}


int main(void)  {
   char *arr[ ] = {”red", ”blue", ”white", ”green"};
   sort(arr, 4);
   …
   return 0;
}
```

Use of Array of Pointers for Arguments of main Function

- C and C++, allow to have a main function with two argumets:

```
int main(int argc, char **argv) {
        //
        ...
}
```

  - The first argument is a pointer that may point to an array of char*.
    - Each pointer may refer to a token from command line.
- Example:
  - Assume you have created an executable from you C or C++ program called a.exe. Now if you run this program from command line in the following form:

```
./a.exe cat cow dog
```

- The main function can have access to all tokens on the command line.

Command line Argument:

```
int main(int argc, char **argv) {
    for(int j = 0; j < argc; i++)
        cout << argv[i];
    // point one

    return 0;
}
// prints:
        ./a.exe
        cat
        cow
        dog
```