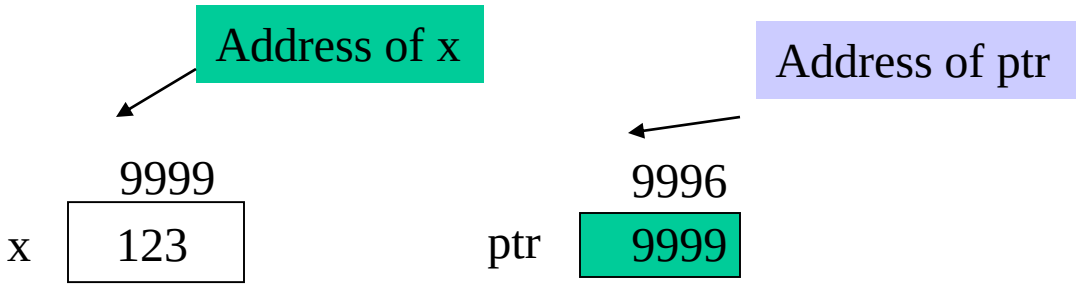
- Character Strings as Arrays of Characters
- Array as Function Arguments
- String-Handling Functions
- Real World Application: Computing A String's Length

Pointer variables

Definition: A pointer variable holds the address of a cell

```
int x =123;
int * ptr;
ptr=&x;
```



Pointer has its own address.

The content of pointer is the address of another cell, e.g., the addre
of x.

Applications of pointers

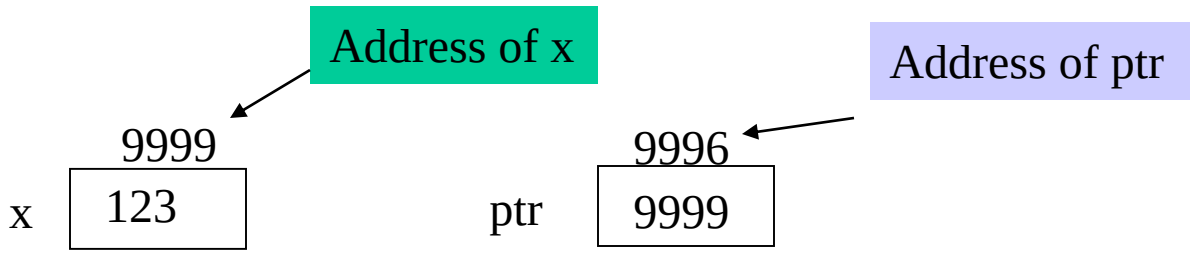
2 ways to access a variable:

I. int x = 123; /* access x through its name */

II ptr = &x; /* the address of variable x assign to ptr. */
*ptr=123

Note: There is no space between & and variable x.

A variable that holds an address, such as ptr, is called the **pointer variable** (simply call **pointer**).



To **dereference** the pointer ptr is to use the expression ***ptr.**
***ptr --** to access the contents of the cell whose address is stored in ptr.

```
ptr = &x;
*ptr = 123;
```

Notice that x and *ptr both have the same effect, that is, both store the value 123 in x.

Assignment using a pointer.

Pointer types

A pointer holding address of an integer:

```
int *pt
```

A pointer holding address of a character

```
char *ptr
```

3 ways to define pointers

```
int *pt
int* pt
int * pt                    (not a popular way)
```

defines pt as a pointer variable of type int*

```
int *pt1, *pt2 /*both are pointers*/
int* pt1, pt2 /*pt1 is a pointer; pt2 is an integer*/
```

```
void f( int* p, char* s )
{
    ...
}
```

```
void f( int *p, char *s )
{
    …
}
```

void* pointer type

data type for a generic pointer variable

a pointer to void may be converted to any other data type and vice versa


```
int *iptr;
void *vptr;
float real = 123;
iptr = &real;    /*** Error to hold the address of float ***/
iptr = ( int* ) &real;    /* It is ok because of cast operation. */
vptr = &real;   /*** It is ok  *** /
iptr = vptr;      /*** It is ok ***/
```

```
void *f( void );          /* f is a pointer holding the address
                             determined by the return value*/
char *g( void );

int *ret;

ret = f( );               /*** OK ***/

ret = g( );               /*** ERROR: cast needed ***/

ret = ( int* ) g( );              /*** OK ***/
```

```
	/* program swap.c */
#include <stdio.h>
#include <stdlib.h>
main( ) {
	char char1 = ‘A’;
	char char2 = ‘B’;
	char temp;
	char *char_ptr;
	char_ptr = &char1;
	temp = *char_ptr;
	*char_ptr = char2;	/* the contents of the variable whose address is in
				char_ptr */
	char2 = temp;
}
```

```
char char1 = 'A';
char char2 = 'B';
char temp;
```

char1	'A'	char2	'B'	temp	?
1256		2468		5566	

Allocation and initialization.

```
char *char_ptr;
```

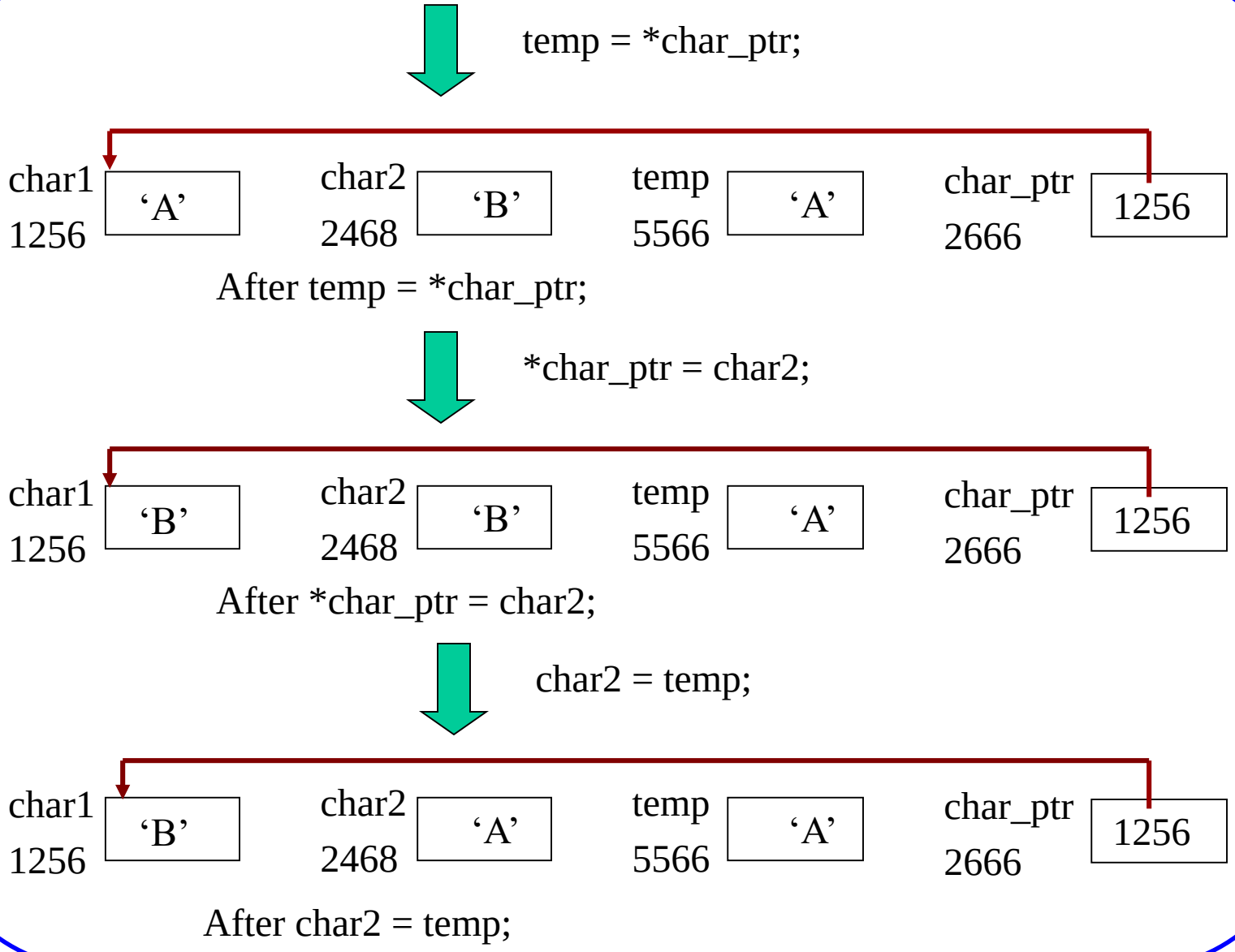
char1	'A'	char2	'B'	temp	?	char_ptr	?
1256		2468		5566		2666	

Allocation of a cell to hold an address.

```
char_ptr = &char1;
```

char1	'A'	char2	'B'	temp	?	char_ptr	1256
1256		2468		5566		2666	

After char_ptr = &char1;


temp = *char_ptr;
char1
1256
'A'
char2
2468
'B'
temp
5566
'A'
char_ptr
2666
1256
After temp = *char_ptr;
*char_ptr = char2;
char1
1256
'B'
char2
2468
'B'
temp
5566
'A'
char_ptr
2666
1256
After *char_ptr = char2;
char2 = temp;
char1
1256
'B'
char2
2468
'A'
temp
5566
'A'
char_ptr
2666
1256
After char2 = temp;

The contrast between the variables **char_ptr** and **char1**.

•The variable **char1** holds a character, and we can access a character directly through **char1**.

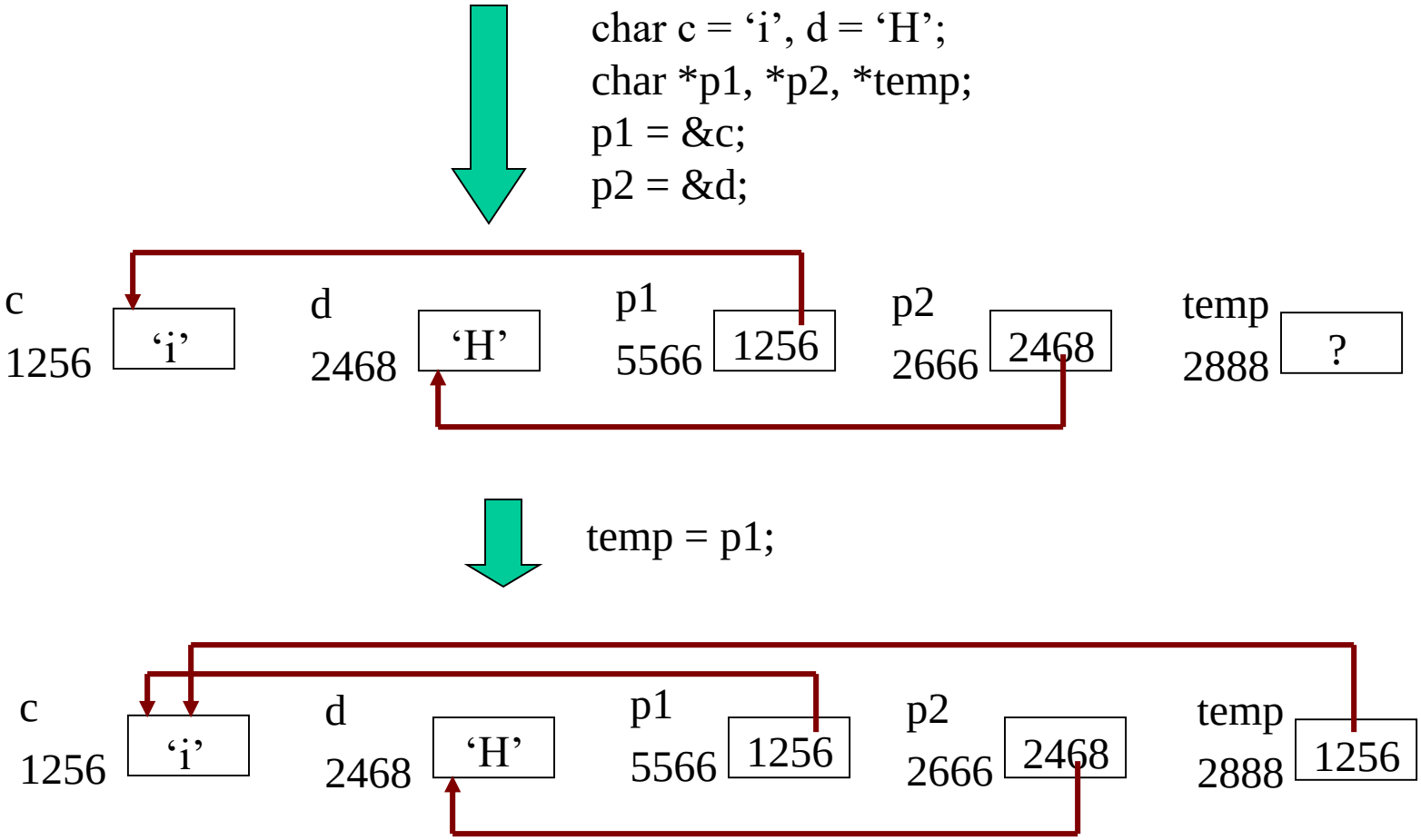
```
putchar ( char1 );
```

•The variable char_ptr holds an address, to access a character through char_ptr, we must dereference the pointer.

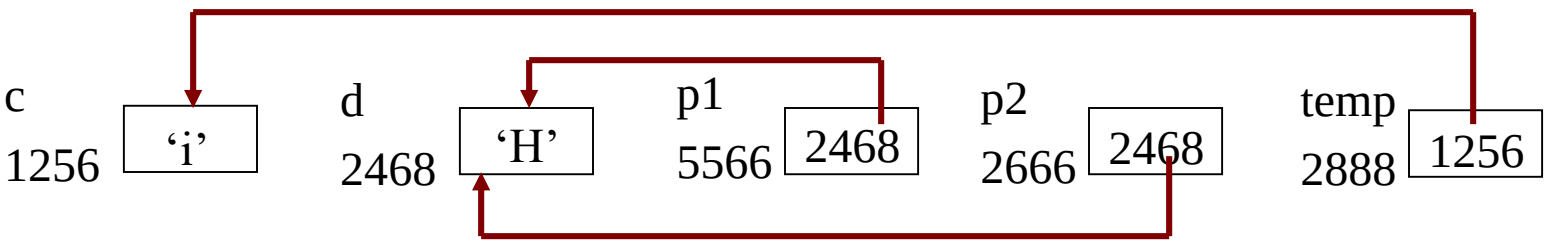
```
putchar( *char_ptr );
```

```
#include <stdio.h>
#include <stdlib.h>
main ( ) {
        char c = ‘i’, d = ‘H’;
        char *p1, *p2, *temp;
        p1 = &c;
        p2 = &d;
        temp = p1;
        p1 = p2;
        p2 = temp;
        printf( “%c%c,\n”, *p1, *p2 );
        return EXIT_SUCCESS;
}
```

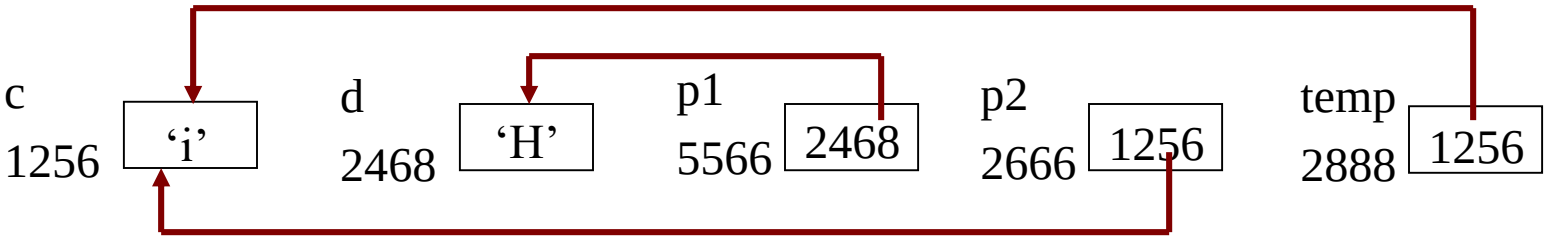
The output is
Hi,



p1 = p2;



p2 = temp;



Printf("%c%c,", *p1, *p2);

We print

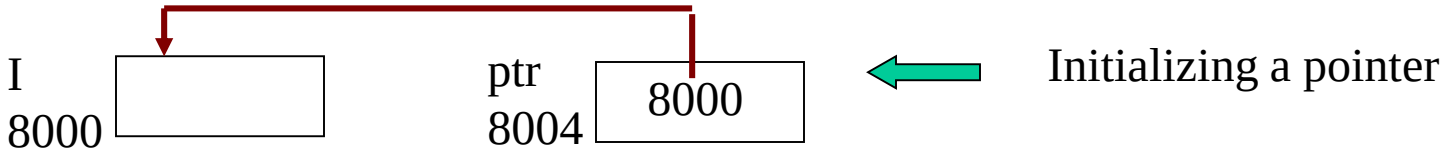
Hi,

♦ Initializing a Pointer

```
int I;
int* ptr = &I;      /* initialize pointer ptr  instead of  *ptr. */
```

int* ptr = &I; ⟺

```
int *ptr;
ptr = &I;
```



```
#include <stdio.h>
#include <stdlib.h>
main( ) {
        char c1 = 'J', c2 = 'O', c3 = 'G';
        char* p1 = &c1;
        char* p2;
        p2 = &c2;
        printf( "%c%c%c\n", *p1, *p2, c3 );          /* print JOG */
        return EXIT_SUCCESS;
}
```

Initializing a Pointer

- The address operator cannot be applied to a constant.

```
&77     /*** ERROR ***/
```

- The address operator cannot be applied to an expression involving operators such as + and /

```
int num = 6;
&(num + 11)   /*** ERROR ***/
```

Exercises 7.1

1. What is the error?

```
char var1, ptr1;
var1 = 'x';
ptr1 = &var1;
```

2.
```
double var1, *ptr1, *ptr2;
float*   ptr3;
int      var2, *var4;
```

data type of above?

Exercises 7.1

1. What is the error?

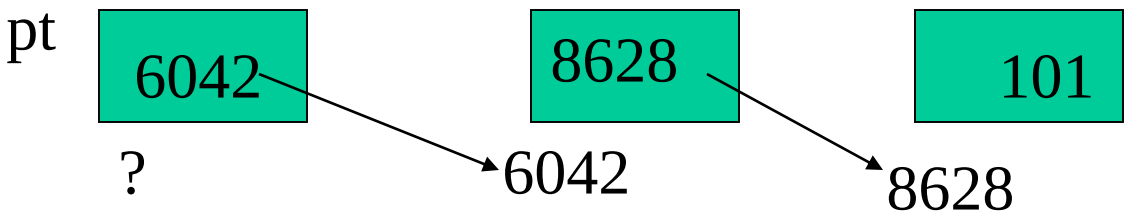
```
char var1, ptr1;
var1 = 'x';
ptr1 = &var1/*ptr1 not defined as a pointer*/
```

2.
```
double var1, *ptr1, *ptr2;
float*   ptr3;
int      var2, *var4;
```
var1 (double); var2 (int); var4 (pointer to int)
*var4(int); ptr1(pointer to double);
*ptr1(double)

➢Levels of Indirection

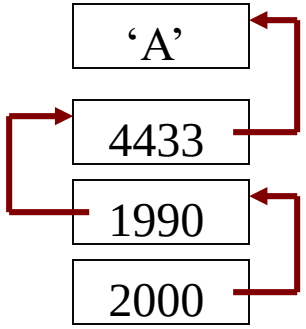
int** pt; /* Each star is read “pointer to” */

pt is a pointer to a pointer that points to an int cell



Level of indirection -- The number of arrows that we must follow to access the datum or, equivalently, the number of stars that must be attached to the variable to reference the value to which it points.

Line	C Program	Cell Name	Cell	Cell Address
	main( ) {			
1	char char1 = ‘A’;	char1	‘A’	4433
2	char* ptr1;	ptr1		1990
3	char** ptr2;	ptr2		2000
4	char*** ptr3;	ptr3		9994
5	ptr1 = &char1;	char1	‘A’	4433
		ptr1	4433	1990
6	ptr2 = &ptr1;	ptr2	1990	2000
7	ptr3 = &ptr2;	ptr3	2000	9999
	}			



3 char** ptr2; /* two star, its level of indirection is two. To access a char through ptr2, we need to dereference it twice */

Level of indirection

```
char char1 = 'A';
        level =0        char1 ->        char
char*   ptr1;
        level = 1       *ptr1 ->        char
char** ptr2;
        level = 2       ptr2=&ptr1;
char*** ptr3;
        level = 3       ptr3=&ptr2;
***ptr3='z' /*storing 'z' in char1 cell*/
```

This definition and initialization contain several pieces of information:

- The expression ***ptr** is of type **char** rather than, say, **int** or **float**.
- Through the triple dereferencing of **ptr3**, we can access a **char**, or equivalently,
  ptr3 should point to a pointer that points to a pointer that points to a **char**.
- The variable **ptr3** is initialized to the address, not the contents, of **ptr2**. This address happens to be 2000. The equivalent assignment statement is thus

  ptr3 = &ptr2;

  and not

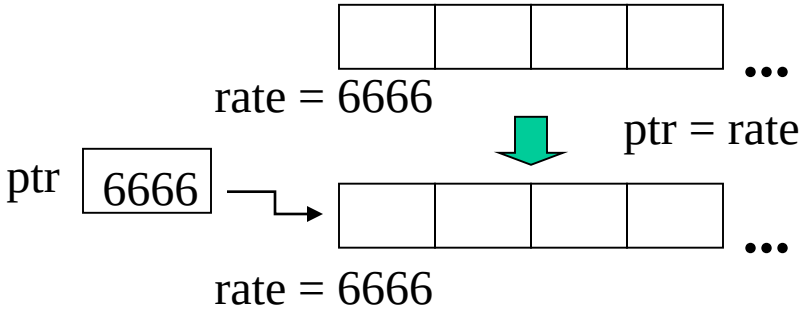
  ***ptr3 = &ptr2; /* ERROR, ***ptr evaluates to a char, not to a pointer. */

char* ptr1	
*ptr1	a char
ptr1	a pointer to a char
---	
char*** ptr3	
***ptr3	a char
**ptr3	a pointer to a char
*ptr3	a pointer to a pointer to a char
ptr3	a pointer to a pointer to a pointer to a char

➢Pointers and Arrays

An array ‘s name is a constant whose value is the address of the array’s first element. So the value of an array’s name cannot be changed by an assignment statement or by any other means.

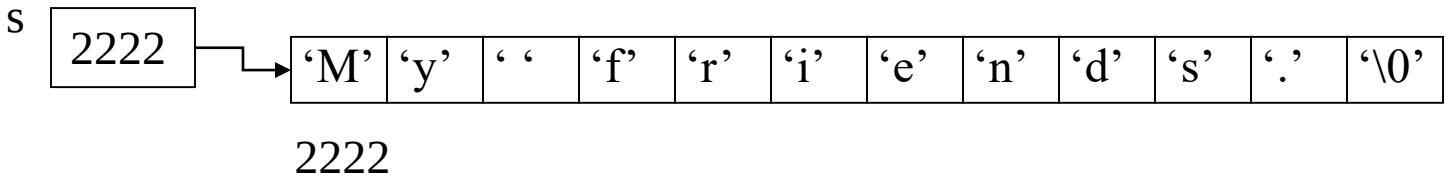
```
float rate[20];
float *ptr;
ptr = rate; /* or ptr = &rate[0] */

rate = ptr;        /*** ERROR ***/
&rate[0] = ptr;    /*** ERROR ***/
```



rate=ptr is as erroneous as 30=x

♦ Pointers to char and Arrays of Type char

char* s = “My friends.”; ⟺ { char*s; s = “My friends.”;

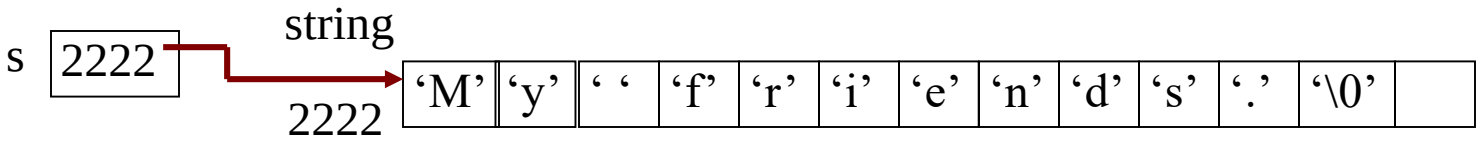


char* s	Defines s as a pointer to char
s=“My friends.”	storage is allocated for the string “My friends.” and the address of the first cell is stored in s s is not an array. s is a pointer variable.
char s2[]=“My friends.”	s2 is an array; also a pointer constant

```
char s3[ 30 ];
s3 = “My friends.”;                    /***** ERROR: s3 is a pointer
                                              constant!*****
char s3[ ] = “My friends.”;            /* It is ok. */

char s3[30] = “My friends.”;
```

```
char* s;
scanf( “%s”, s );          /* ERROR: s has not been initialized. */
```

```
char string[ 13 ];     /*allocate storage cells for string*/
char* s = string;      /* allocates storage cell for s and stores the address of the
                          first cell of string to s */
scanf( “%s”, s );            /* It is ok */
suppose we type “My friends.” in keyboard.
```



- It is legal to take the address of an array s, the result is typically equal to the address of the first element of s.

```
char s[ ] = “Hi”, **p;
p = &s;
scanf( “%s\n”, *p )
```

/* ERROR: **p = &s . p** references the content of the array **s**. It prints garbage */

```
char s[ ] = “Hi”, *t, **p;
t  = s;
p = &t;
scanf( “%s\n”, *p );
```

/*stores address of s to pointer t*/
/*stores address of t to p*/
/* It is correct */