char *s[5];

Lines 11-12: We need the address of the string "Neal", or with characters values stored in it.

Lines 3-4: These two function prototypes are essentially identical, since as parameters, char * and char [] are the same. In a prototype, we don't need the name of the parameter, though we could have written char *a[] and char a[].

Notes and Comments:

1. Lines 3-4: We need a string.h for strlen and strcpy, and

2. Lines 5-6: These two function prototypes are essentially identical, since as parameters, char * and char [] are the same. In a prototype, we don't need the name of the parameter, though we could have written char *a[] and char a[].

3. Line 10: This lays out a char array of size 5, with uninitialized values.

4. Line 14: The variables p, q, and r are declared to be of type "pointer to char", or "address of char". Thus, p = s is legal, since p is not a constant, but s = p is illegal, since s is a constant pointer to char. This is very similar to the types of variables s, t, and u, except that those latter "pointers to char", when declared to be of type pointer to char, p, q, and r, are declared to be of type pointer to char, p, q, and r, and are declared to be of type pointer to char, p, q, and r.

5. Lines 3-4: These two function prototypes are essentially identical, since as parameters, char * and char [] are the same. In a prototype, we don't need the name of the parameter, though we could have written char *a[] and char a[].

6. Line 10: This lays out a char array of size 5, with uninitialized values.

7. Line 14: The variables p, q, and r are declared to be of type "pointer to char", or "address of char". Thus, p = s is legal, since p is not a constant, but s = p is illegal, since s cannot be changed (cannot be on the left side of an assignment statement).
Strings in C.

'\texttt{malloc}' is the most common and flexible way to work with

arrays. The method of

allocation requires

\texttt{malloc} to be

passed a parameter, which

is the number of bytes

allocated.

Finally, array

allocation produces a

separate block of memory

for each

\texttt{malloc}.

The new block can be

used independently of

the original array.

Initially, I will have

useless garbage stored in it.

The array, \texttt{s}, will

have useless garbage stored in it.

The address of the second item, and so forth.

Given the address of

something, C's operation

returns

\texttt{malloc} or

\texttt{free} is the address of

the second item, and so forth.

Then, the address of

this storage is stored in the variable

\texttt{p}.

This is the first element

of the character array.

The other

\texttt{malloc} function

is just a fancy way to write

\texttt{malloc}.

The other

parameter is the size of

the block to be allocated.

In C, the

non-null characters in

a character string

are always stored

in a contiguous

block of memory.

The

\texttt{malloc} function

returns

a pointer to the first item

in the array.

Given an

address of a character

string, C provides

\texttt{sizeof} to find

the size of the block

held by the

\texttt{malloc} function.

Thus, the

variable

\texttt{p} holds the address
to the end is copied also.) Finally,

is the most common and flexible way to work with

arrays. The method of

allocation requires

\texttt{malloc} to be

passed a parameter, which

is the number of bytes

allocated.

Finally, array

allocation produces a

separate block of memory

for each

\texttt{malloc}.

The new block can be

used independently of

the original array.

Initially, I will have

useless garbage stored in it.

The array, \texttt{s}, will

have useless garbage stored in it.

The address of the second item, and so forth.

Given the address of

something, C's operation

returns

\texttt{malloc} or

\texttt{free} is the address of

the second item, and so forth.

Then, the address of

this storage is stored in the variable

\texttt{p}.

This is the first element

of the character array.

The other

\texttt{malloc} function

is just a fancy way to write

\texttt{malloc}.

The other

parameter is the size of

the block to be allocated.

In C, the

non-null characters in

a character string

are always stored

in a contiguous

block of memory.

The

\texttt{malloc} function

returns

a pointer to the first item

in the array.

Given an

address of a character

string, C provides

\texttt{sizeof} to find

the size of the block

held by the

\texttt{malloc} function.

Thus, the

variable

\texttt{p} holds the address
to the end is copied also.) Finally,

is the most common and flexible way to work with

arrays. The method of

allocation requires

\texttt{malloc} to be

passed a parameter, which

is the number of bytes

allocated.

Finally, array

allocation produces a

separate block of memory

for each

\texttt{malloc}.

The new block can be

used independently of

the original array.

Initially, I will have

useless garbage stored in it.

The array, \texttt{s}, will

have useless garbage stored in it.

The address of the second item, and so forth.

Given the address of

something, C's operation

returns

\texttt{malloc} or

\texttt{free} is the address of

the second item, and so forth.

Then, the address of

this storage is stored in the variable

\texttt{p}.

This is the first element

of the character array.

The other

\texttt{malloc} function

is just a fancy way to write

\texttt{malloc}.

The other

parameter is the size of

the block to be allocated.

In C, the

non-null characters in

a character string

are always stored

in a contiguous

block of memory.

The

\texttt{malloc} function

returns

a pointer to the first item

in the array.

Given an

address of a character

string, C provides

\texttt{sizeof} to find

the size of the block

held by the

\texttt{malloc} function.

Thus, the

variable

\texttt{p} holds the address
to the end is copied also.) Finally,

is the most common and flexible way to work with

arrays. The method of

allocation requires

\texttt{malloc} to be

passed a parameter, which

is the number of bytes

allocated.

Finally, array

allocation produces a

separate block of memory

for each

\texttt{malloc}.

The new block can be

used independently of

the original array.

Initially, I will have

useless garbage stored in it.

The array, \texttt{s}, will

have useless garbage stored in it.

The address of the second item, and so forth.

Given the address of

something, C's operation

returns

\texttt{malloc} or

\texttt{free} is the address of

the second item, and so forth.

Then, the address of

this storage is stored in the variable

\texttt{p}.

This is the first element

of the character array.

The other

\texttt{malloc} function

is just a fancy way to write

\texttt{malloc}.

The other

parameter is the size of

the block to be allocated.

In C, the

non-null characters in

a character string

are always stored

in a contiguous

block of memory.

The

\texttt{malloc} function

returns

a pointer to the first item

in the array.

Given an

address of a character

string, C provides

\texttt{sizeof} to find

the size of the block

held by the

\texttt{malloc} function.

Thus, the

variable

\texttt{p} holds the address
to the end is copied also.) Finally,