Section 1: Files

Talk about Files rules recursive rmdir –r and cp –r for directories

Filenames

按理来说，linux除了”/” 都可以在filename里

Rmdir

Only works if directory是空的,不然需要rm – r

CP

需要cp – r才能copy directory，不然会error

Mv

无论如何都不需要-r

mv
- move
- 重命名文件.

不能用于创建文件.

创建文件: vi / nano 用editor

touch 不用editor

cd ~

cd $HOME

最上一级 dir : /

关于/..

../../../../../../../../.. 最多回到“/”directory输再多也不会出error

Soft Links

ln –s target_directory link_name

⍰ Cd link_name

ls -la show detail，显示文件开头为"."的文件

ls 判断文件是 file : -rwxrwxrwx

dir : drwxrwxrwx

link : lrwxrwxrwx

Hard Links

ln target_FILE link_name

Hard link只能给予文件，不能给予directories

Section 2: Chmod

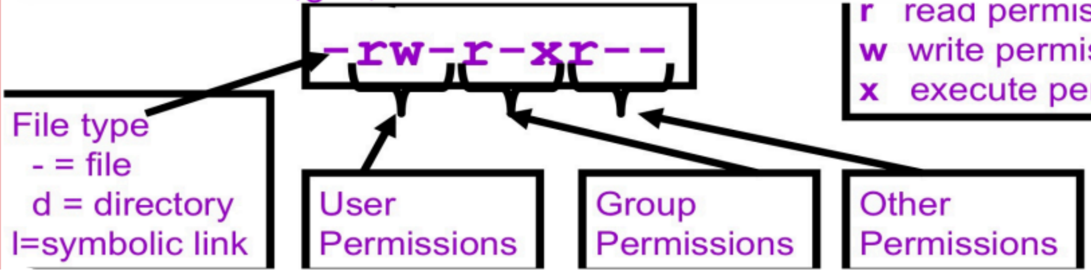
File Permissions (4)

chmod [ugoa][+-=][rwx] file [...]

- This is the “symbolic” method.
- chmod u+rwx file gives the User Read, Write, and eXecute
- chmod g+rx file gives the Group Read and eXecute
- chmod o-rwx file removes R, W, and X from Others
- chmod a+x file gives All eXecute permission
- chmod g=r file gives Group Read permission and makes sure it has nothing else

Symbolic modes can be appended with commas

- chmod u=rwx,g-w,o-rwx file for instance



Directory Permissions (3)

The -R option to chmod is useful when working with directories.

- It recursively changes the mode for each chmod operand that is a directory.
- All files and directories would receive those permissions.

- chmod -R a+rw dir gives everyone read and write permission to each file under dir (not execute though!!!)
- chmod -R a+rwx dir gives the executable access to allow people to actually access the files under dir
  - ❖Makes all files executable though ...
- chmod -R a+rwX dir gives the executable access only to those files already executable (programs, directories, …)

r read permission
w write permission
x execute permission

1	2	3	4	5	6	7
x	w	xw	r	rx	rw	rwx

Section 3: Redirections

Stdin?   <name   e.g. < text.txt

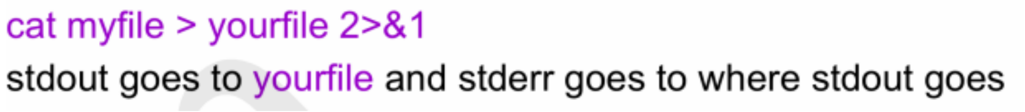
default terminal <= Stdout?   >name e.g. >results.txt

此时 stdout 仍输出 => Stderr?   2>errorfile.txt

BOTH stdout and stderr ?   &>outanderror.txt

<< appends to stdin

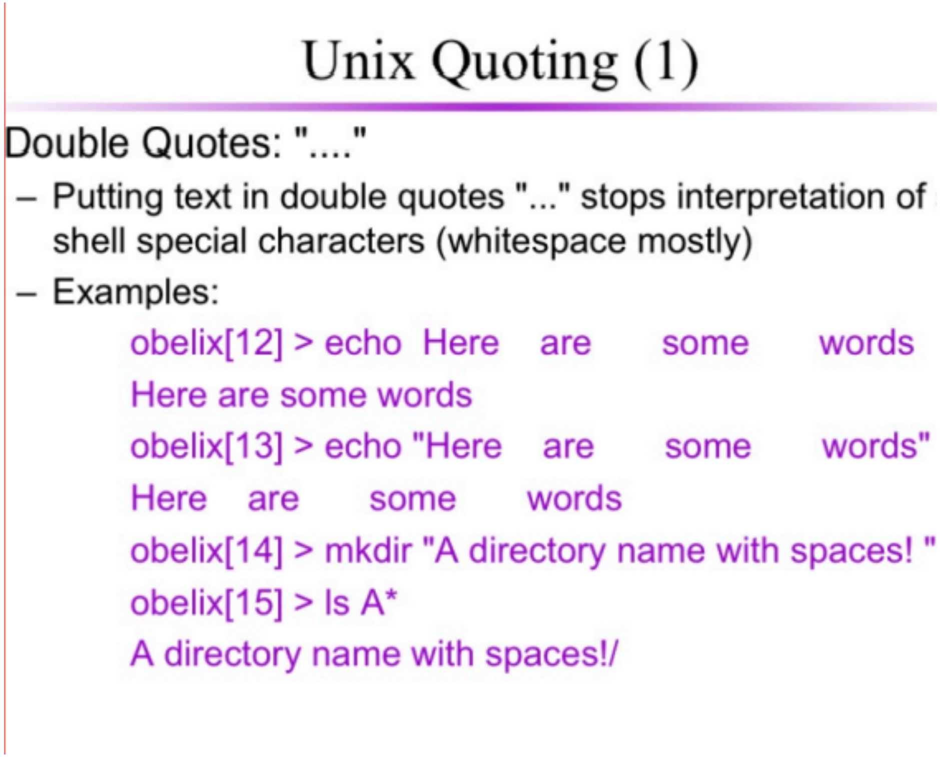
>> appends to stdout e.g. cat file1 file2 >> result.txt ? adds to result.txt, not overwrite



```
cat myfile > yourfile 2>&1
```

stdout goes to yourfile and stderr goes to where stdout goes

Section 4: Quoting



Unix Quoting (1)

Double Quotes: "...."

- Putting text in double quotes "..." stops interpretation of shell special characters (whitespace mostly)
- Examples:

```
obelix[12] > echo  Here    are       some       words
Here are some words
obelix[13] > echo "Here    are       some       words"
Here    are       some       words
obelix[14] > mkdir "A directory name with spaces! "
obelix[15] > ls A*
A directory name with spaces!/
```

Unix Quoting (2)

- Single Quotes '...'
  - Stops interpretation of even more specials
    - Stop variable expansion ($HOME, etc.)
    - Backquotes `...` (execute a command and return result ...we’ll get to this later)
    - Note difference: single quote ('), backquote (`)
    - Examples:

```
obelix[16] > echo "Welcome $HOME"
Welcome /gaul/u0/usr/faculty/kzhang
obelix[17] > echo ‘Welcome $HOME’
Welcome $HOME
```

Unix Quoting (3)

Backslash \

- ‘quotes’ the next character
- Lets one escape all of the shell special characters

```
obelix[18] > mkdir Dir\ name\ with\ spaces\*\*
obelix[19] > ls Dir\ *
Dir name with spaces**/
```

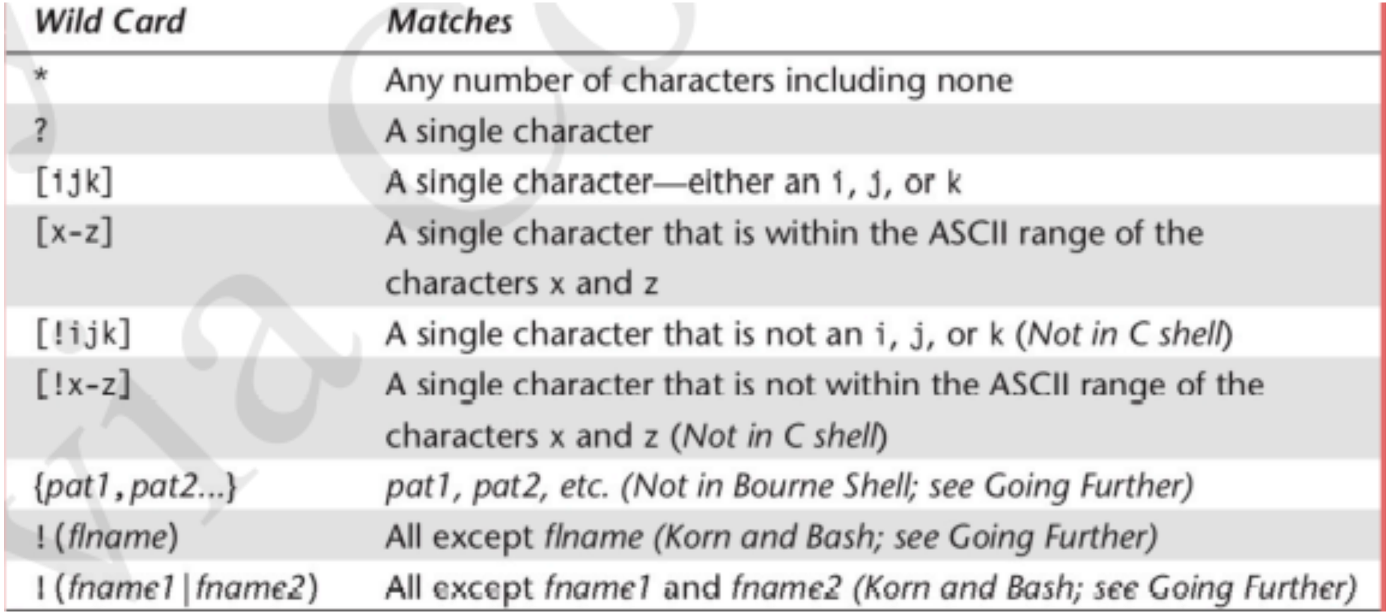
- Use backslash to escape a newline character

```
obelix[20]% echo "This is a long line and\
we want to continue on the next“
This is a long line and we want to continue on the next
```

- Use backslash to escape other shell special chars
    - ❖ Like quote characters

```
obelix[21] > echo \"Bartlett\'s Familiar Quotations\"
"Bartlett's Familiar Quotations"
```

Section 5: Wildcard



Wild Card	*Matches*
*	Any number of characters including none
?	A single character
[ijk]	A single character—either an i, j, or k
[x-z]	A single character that is within the ASCII range of the characters x and z
[!ijk]	A single character that is not an i, j, or k (*Not in C shell*)
[!x-z]	A single character that is not within the ASCII range of the characters x and z (*Not in C shell*)
{*pat1*,*pat2...*}	*pat1, pat2, etc. (Not in Bourne Shell; see Going Further)*
!(*flname*)	All except *flname (Korn and Bash; see Going Further)*
!(*fname1*|*fname2*)	All except *fname1* and *fname2 (Korn and Bash; see Going Further)*

Ls, cp, mv, find -name 都能用wildcards

Section 6: Grep

grep

- Finally, some common grep options you should be aware of:
  - -c – "Count" the number of matches
  - -i – Case insensitive matching
  - -v – "invert" match. (Find the lines that do NOT match the expression)

Section 7: Regex

[ ] – Match any characters in the brackets

[^] – Match any characters NOT in the brackets

- ^ - Matches the start of the line
- $ - Matches the end of the line

[abc] – Matches a or b or c

[^abc] – Matches any character except a or b or c

[a-z] – Matches a, b, c, … y, z

[^a-z] – Matches any character except for a, b, c, … y, z

REGEX1\|REGEX2 – Matches either REGEX1 or REGEX2

E.g. cat\|dog – Matches cat or dog

. – Matches any single character

* or \? - Matches 0 or more characters

+ - Matches 1 or more characters

\{m,n\} – Matches the preceding character at least m times but no more than n times.

- \{m} – A shorthand for \{m,m\}
- \{m,} or \{,n} – A shorthand for \{m,∞\} and \{0,n\}

[:alpha:] – Alphabetic characters

[:digit:] – Digits

[:space:] – Whitespace characters

\(REGEX\) – "Save" anything between the parentheses matched by REGEX

\1, \2, …, \9 – "Recall" the first, second, …, ninth REGEX match

Regular Expressions

*Matches >= 0 occurrences of prev char | .Matches a single char

[pqr] a single char p q or r | [c1-c2] a single char w/i range

[^pqr] a single char not p q or r | .* nothing or any number of chars

^pat pattern pat at the beginning |pat$ pattern pat at the end of line

\<Fo words that begin with Fo | ox\> words that end with ox

[^aeiou] any non vowel | ^[^a-z]*$ any line w/o lower case

g* Nothing or g, gg, ggg, etc | gg* g, gg, ggg, etx.

[1-3] a digit between 1 and 3 | [^a-zA-Z] a non alphabetical char

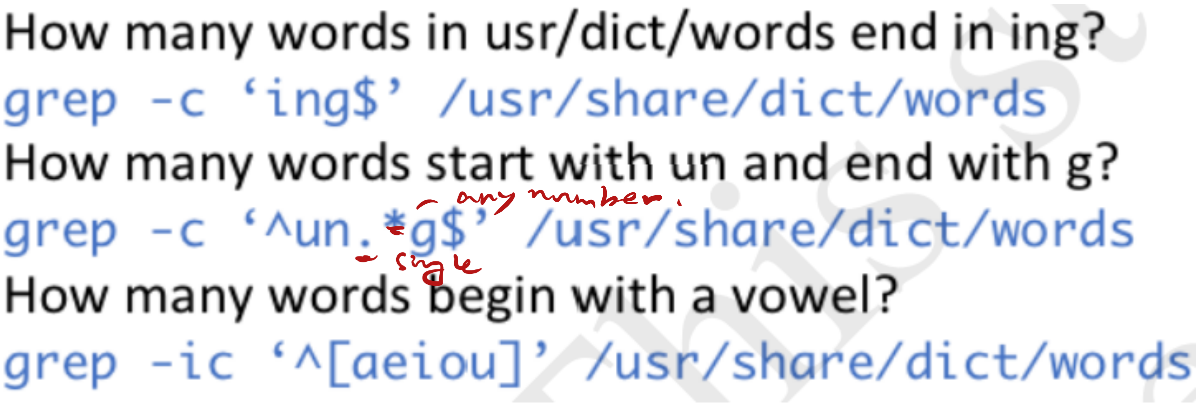
bash$ bash at the end of line |^bash$ bash as the only word in line

^$ lines containing nothing |^\([a-z]\)\1 lines beginning with duplicate

^.*\([a-z][a-z]).*\1.*\1 lines containing >=3 copies that contain []

[a-z]\{2-10\} all sequences of 2-10 lowercase letters |#\{23\} 23 #s

^[aeiou]\{2,\} >=2 vowels in a row at beginning of line



How many words in usr/dict/words end in ing?

```
grep -c ‘ing$’ /usr/share/dict/words
```

How many words start with un and end with g?

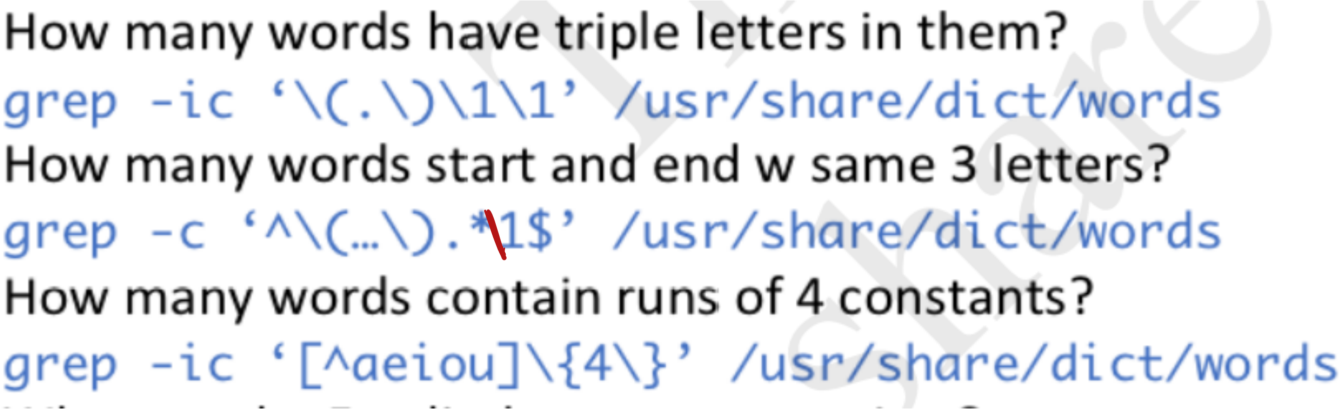
any number.

```
grep -c ‘^un.*g$’ /usr/share/dict/words
```

single

How many words begin with a vowel?

```
grep -ic ‘^[aeiou]’ /usr/share/dict/words
```



How many words have triple letters in them?

```
grep -ic ‘\(.\)\1\1’ /usr/share/dict/words
```

How many words start and end w same 3 letters?

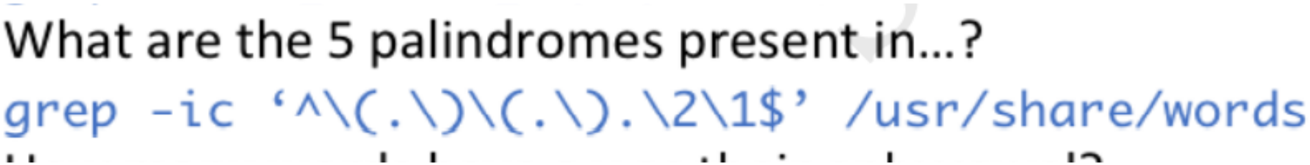
```
grep -c ‘^\(…\).*\1$’ /usr/share/dict/words
```

How many words contain runs of 4 constants?

```
grep -ic ‘[^aeiou]\{4\}’ /usr/share/dict/words
```

How many words dont start and end w the same 3 letters?

```
grep -ivc ‘^\(…\)’.*\1$ /usr/share/dict/words
```



What are the 5 palindromes present in...?

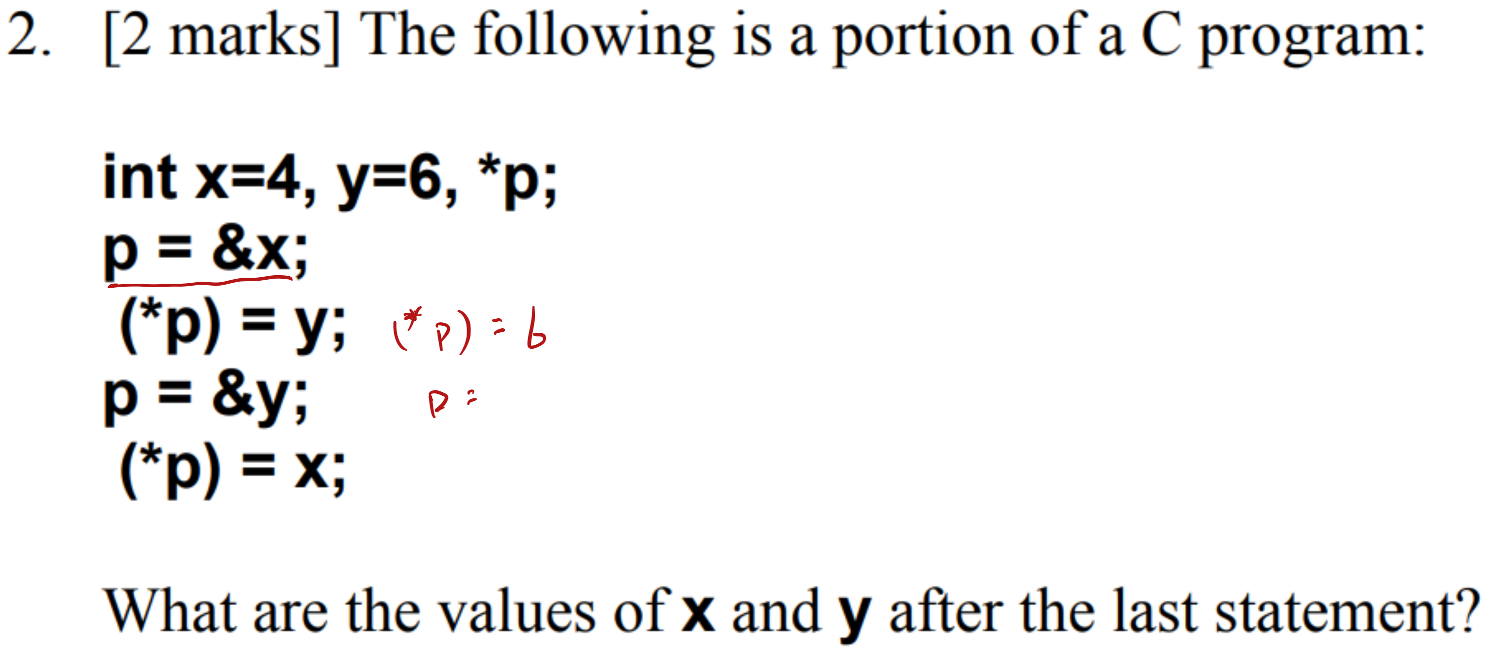
```
grep -ic ‘^\(.\)\(.\).\2\1$’ /usr/share/words
```

How many words have a y as their only vowel?

```
grep ‘^[^aAeEiIoOuU]*$’ /usr/words|grep -ci ‘y’
```

Section 8: C Pointers Basics

Basic * and & memory



2. [2 marks] The following is a portion of a C program:

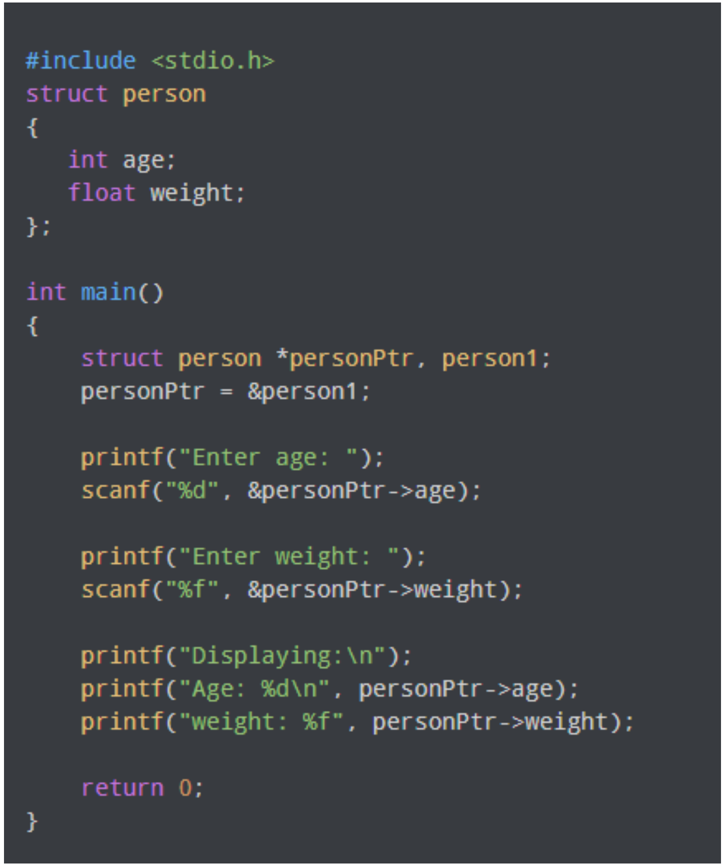
```
int x=4, y=6, *p;
p = &x;
 (*p) = y;
p = &y;
 (*p) = x;
```

(*p) = 6

p =

What are the values of **x** and **y** after the last statement?

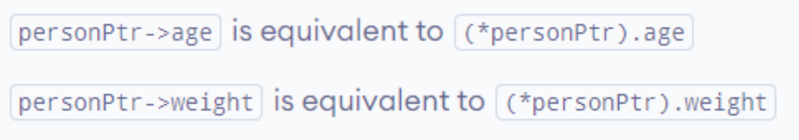
6; 6.



```c
#include <stdio.h>
struct person
{
   int age;
   float weight;
};

int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;

    printf("Enter age: ");
    scanf("%d", &personPtr->age);

    printf("Enter weight: ");
    scanf("%f", &personPtr->weight);

    printf("Displaying:\n");
    printf("Age: %d\n", personPtr->age);
    printf("weight: %f", personPtr->weight);

    return 0;
}
```

pointer 就为简写

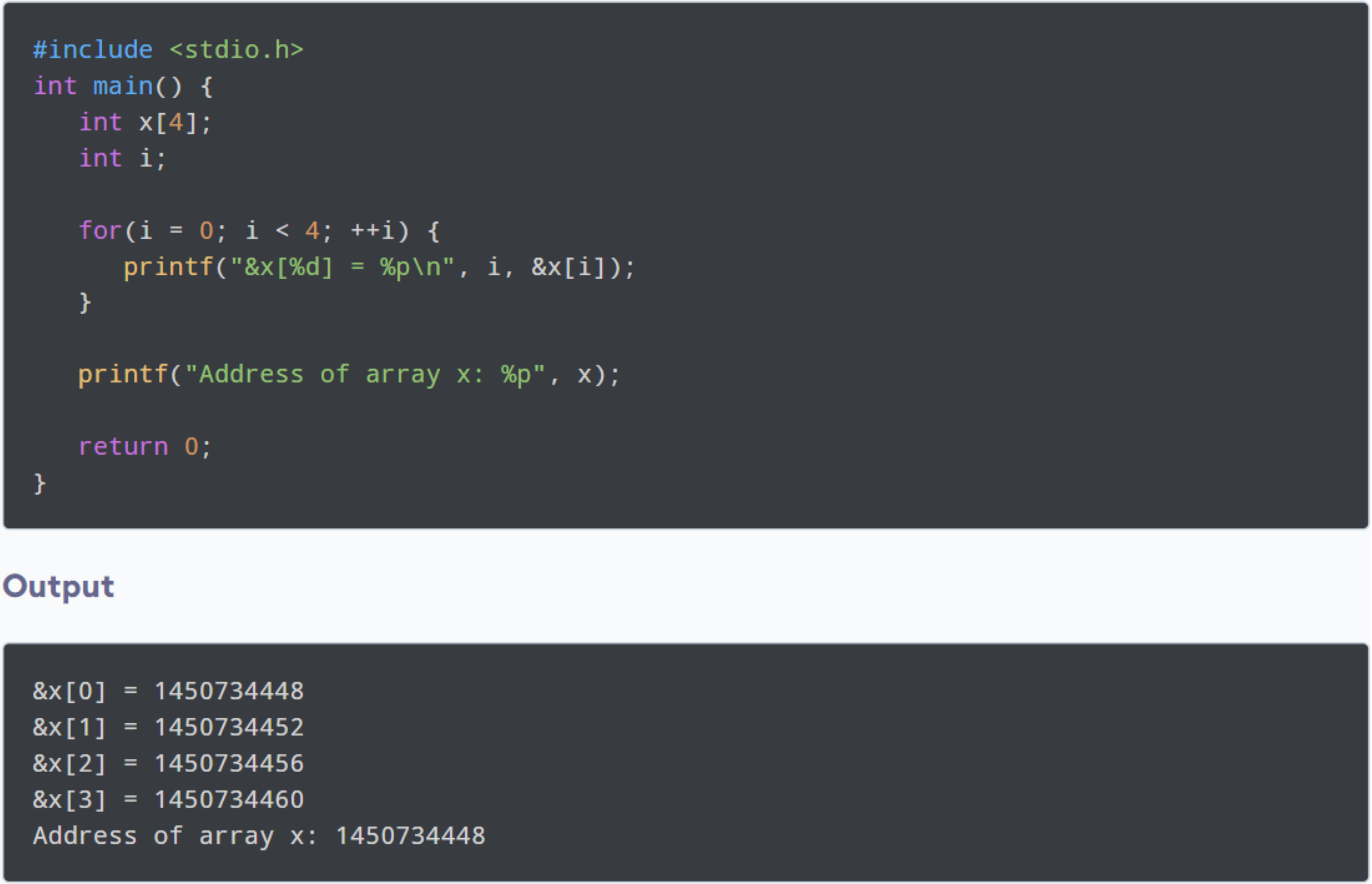
非pointer 只能用 " -> "



`personPtr->age` is equivalent to `(*personPtr).age`

`personPtr->weight` is equivalent to `(*personPtr).weight`

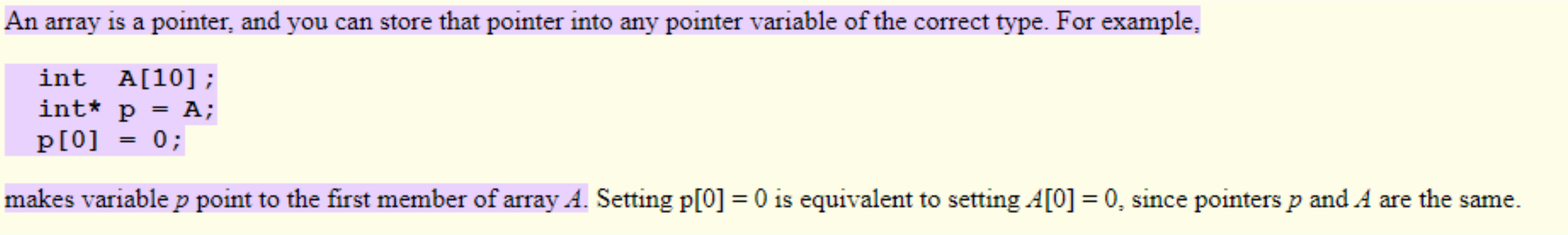
Section 9: C Arrays



```c
#include <stdio.h>
int main() {
   int x[4];
   int i;

   for(i = 0; i < 4; ++i) {
      printf("&x[%d] = %p\n", i, &x[i]);
   }

   printf("Address of array x: %p", x);

   return 0;
}
```

Output

```
&x[0] = 1450734448
&x[1] = 1450734452
&x[2] = 1450734456
&x[3] = 1450734460
Address of array x: 1450734448
```



An array is a pointer, and you can store that pointer into any pointer variable of the correct type. For example,

```
  int  A[10];
  int* p = A;
  p[0] = 0;
```

makes variable *p* point to the first member of array *A*. Setting p[0] = 0 is equivalent to setting *A*[0] = 0, since pointers *p* and *A* are the same.

&x[0] = x

&x[i] = x+I

Pointer + 1 不等于pointer的值+1，而是pointer的值+ (1 x 指向type占用的byte数)

Section 10: C Strings

```
#include <string.h>
```

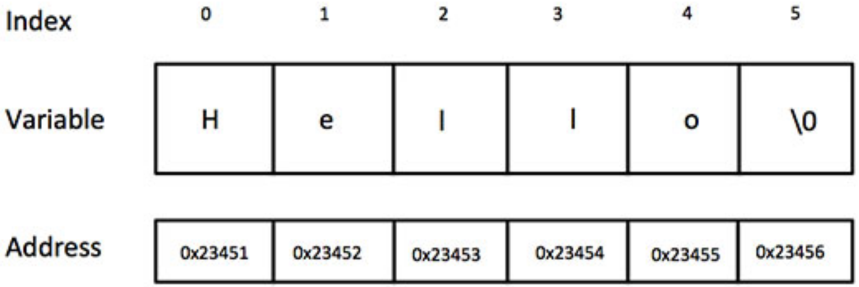
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization then you can write the above statement as follows −

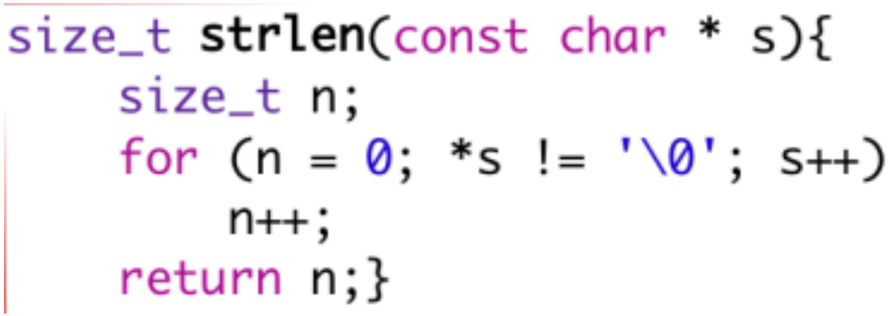
```
char greeting[] = "Hello";
```

Following is the memory presentation of the above defined string in C/C++ −

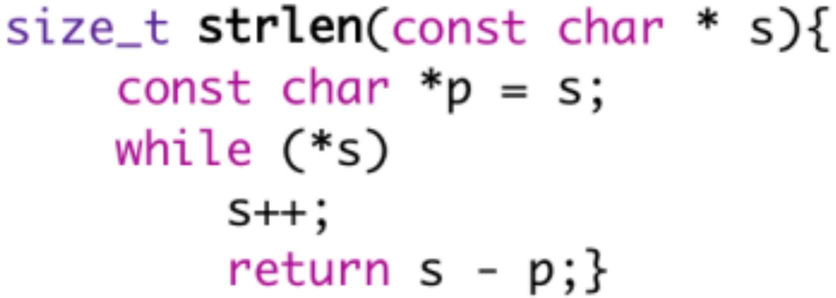


Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

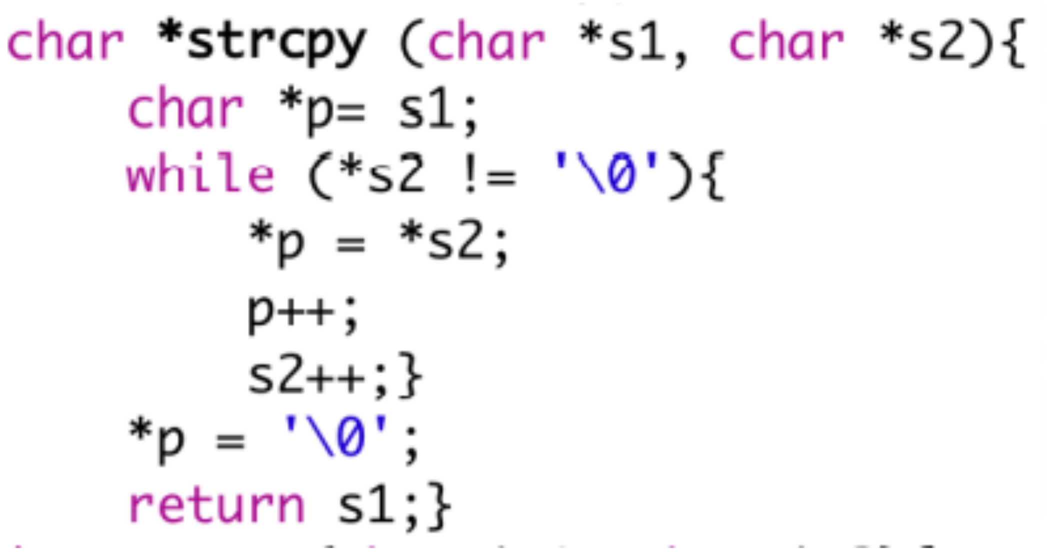
Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string −



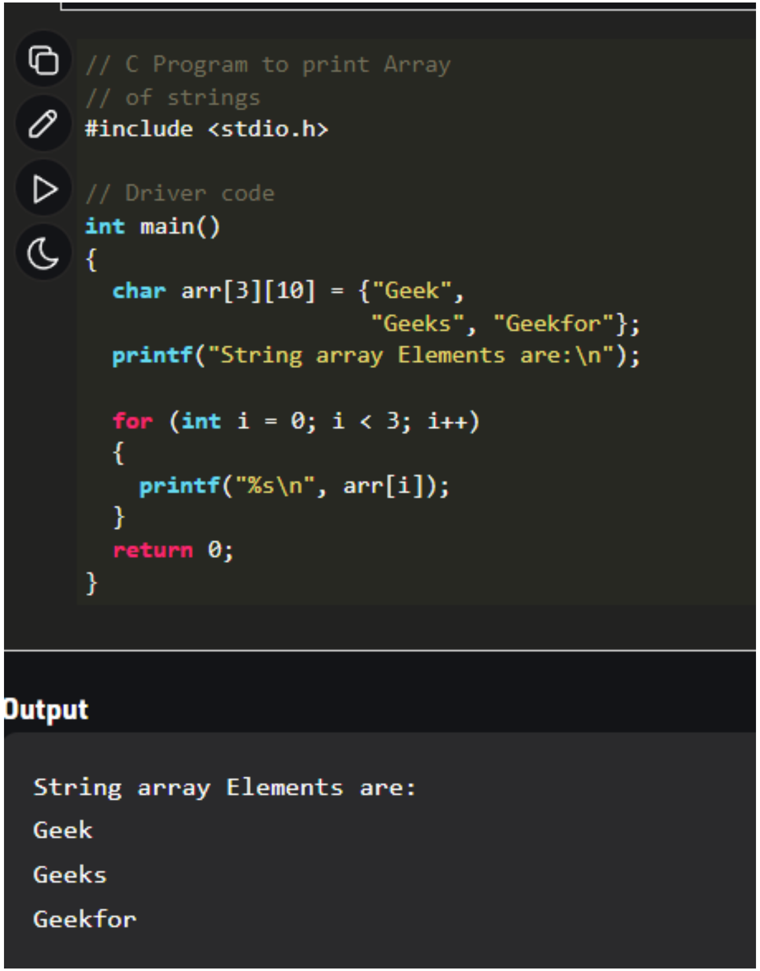
```c
size_t strlen(const char * s){
    size_t n;
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;}
```



```c
size_t strlen(const char * s){
    const char *p = s;
    while (*s)
        s++;
        return s - p;}
```



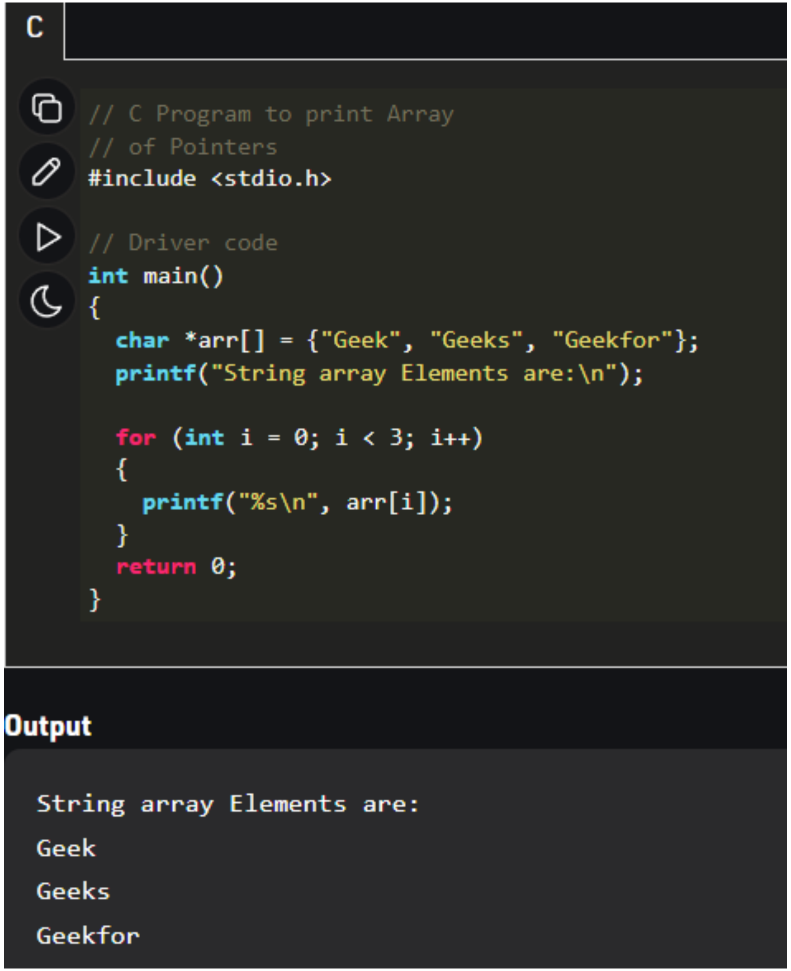
```c
char *strcpy (char *s1, char *s2){
    char *p= s1;
    while (*s2 != '\0'){
        *p = *s2;
        p++;
        s2++;}
    *p = '\0';
    return s1;}
```



```c
// C Program to print Array
// of strings
#include <stdio.h>

// Driver code
int main()
{
  char arr[3][10] = {"Geek",
                     "Geeks", "Geekfor"};
  printf("String array Elements are:\n");

  for (int i = 0; i < 3; i++)
  {
    printf("%s\n", arr[i]);
  }
  return 0;
}
```

Output

```
String array Elements are:
Geek
Geeks
Geekfor
```



```c
// C Program to print Array
// of Pointers
#include <stdio.h>

// Driver code
int main()
{
  char *arr[] = {"Geek", "Geeks", "Geekfor"};
  printf("String array Elements are:\n");

  for (int i = 0; i < 3; i++)
  {
    printf("%s\n", arr[i]);
  }
  return 0;
}
```

Output

```
String array Elements are:
Geek
Geeks
Geekfor
```