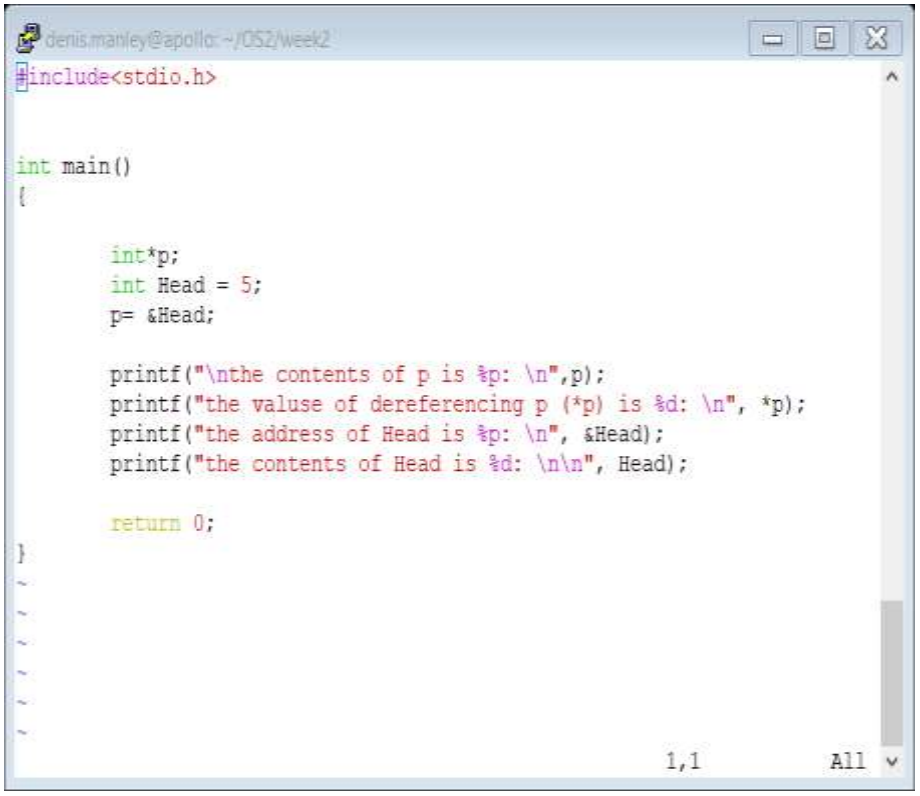
Lecture 3

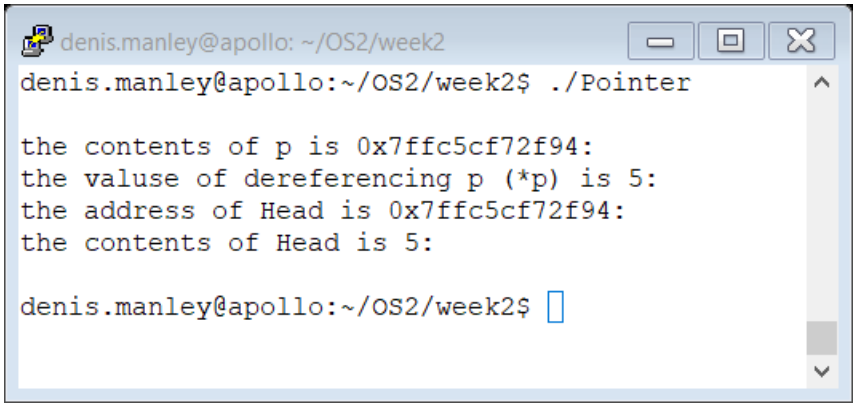
pointers, pointer arithmetic, functions and strings

C pointers

- The name of an Array (e.g. ages) stores an address.
- A variable that stores the *address of a standard variable (int, float, char) is referred to as a "pointer"*;
- The array name is called a constant pointer (the address is a constant…)
- Declaration and assigning an address value:
  - int *p; (pointer variable)
  - int Head =5; (standard variable)
  - p = & Head;
- Printing the value of a pointer (p); and the value of the variable whose address is stored in p :

Code and output of Pointer.c





Pointer arithmetic

- An array can be transverse using **subscripts** but can also transverse it using (variable) pointer notation. // example from C programming by Paul Kelly

denis.manley@apollo: ~/OS2/week2

```
/* Program to access the elements of an array using
    element addresses rather than subscripts.      */
#include <stdio.h>
#define NO_OF_ELS 5
int main()
{
  int a[NO_OF_ELS] = { 10, 13, 15, 11, 6 } ;
  int i ;

  printf("the contents of the array name is %p\n", a);

  for ( i = 0 ; i < NO_OF_ELS ; i++ )
  {
    printf( "Element %d is %d\n", i, *(a+i) ) ;
    printf("the address of the element is: %p\n\n", a+i);
  }

  printf("press return to exit: ");
  getchar();

  return 0; // to indicate the end of the program
}
~
"array_using_pointers.c" 22L, 535C                    1,1           All
```

- What is stored in a
- Can the value stored in a be modified.
- Change a to a character and then a float array. What is the difference?

Explain “Output” of following code

denis.manley@soc-apollo-dk: ~/OS2/week2

```c
/* Program to access the elements of an array using
    element addresses rather than subscripts.      */
#include <stdio.h>
#define NO_OF_ELS 5
int main()
{
  int a[NO_OF_ELS] = { 10, 13, 15, 11, 6 } ;
  int i ;

  printf("the contents of the array name is %p\n", a);

  for ( i = 0 ; i < NO_OF_ELS ; i++ )
  {
    printf( "The contents of element %d is %d\n", i, *(a+i) ) ;

    printf("the address of the element is: %p\n\n", a+i);
  }

  printf("press return to exit: ");
  getchar();

  return 0; // to indicate the end of the program
}
```

1,1 All

```
denis.manley@soc-apollo:~/OS2/week2$ ./array_using_pointers
the contents of the array name is 0x7ffda7c7c110
The contents of element 0 is 10
the address of the element is: 0x7ffda7c7c110

The contents of element 1 is 13
the address of the element is: 0x7ffda7c7c114

The contents of element 2 is 15
the address of the element is: 0x7ffda7c7c118

The contents of element 3 is 11
the address of the element is: 0x7ffda7c7c11c

The contents of element 4 is 6
the address of the element is: 0x7ffda7c7c120

press return to exit:
```

the above is a integer

Pointer arithmetic

- **Using ++/--**…. (See Char_pointer_arithmetic.c)
- You can not increment an array name as it is a **constant** pointer (a++ is invalid)
- However by assigning a standard pointer to an array you can increment using "pointer arithmetic"…
  - int *ptr;
  - int Numbers[25] = {1,2,3,4,5,6};
  - prt = Numbers; //assign the array pointer to the pointer variable;
  - Now can use p++ to increment elements… but ensure it remains within the "*bounds of the array*".

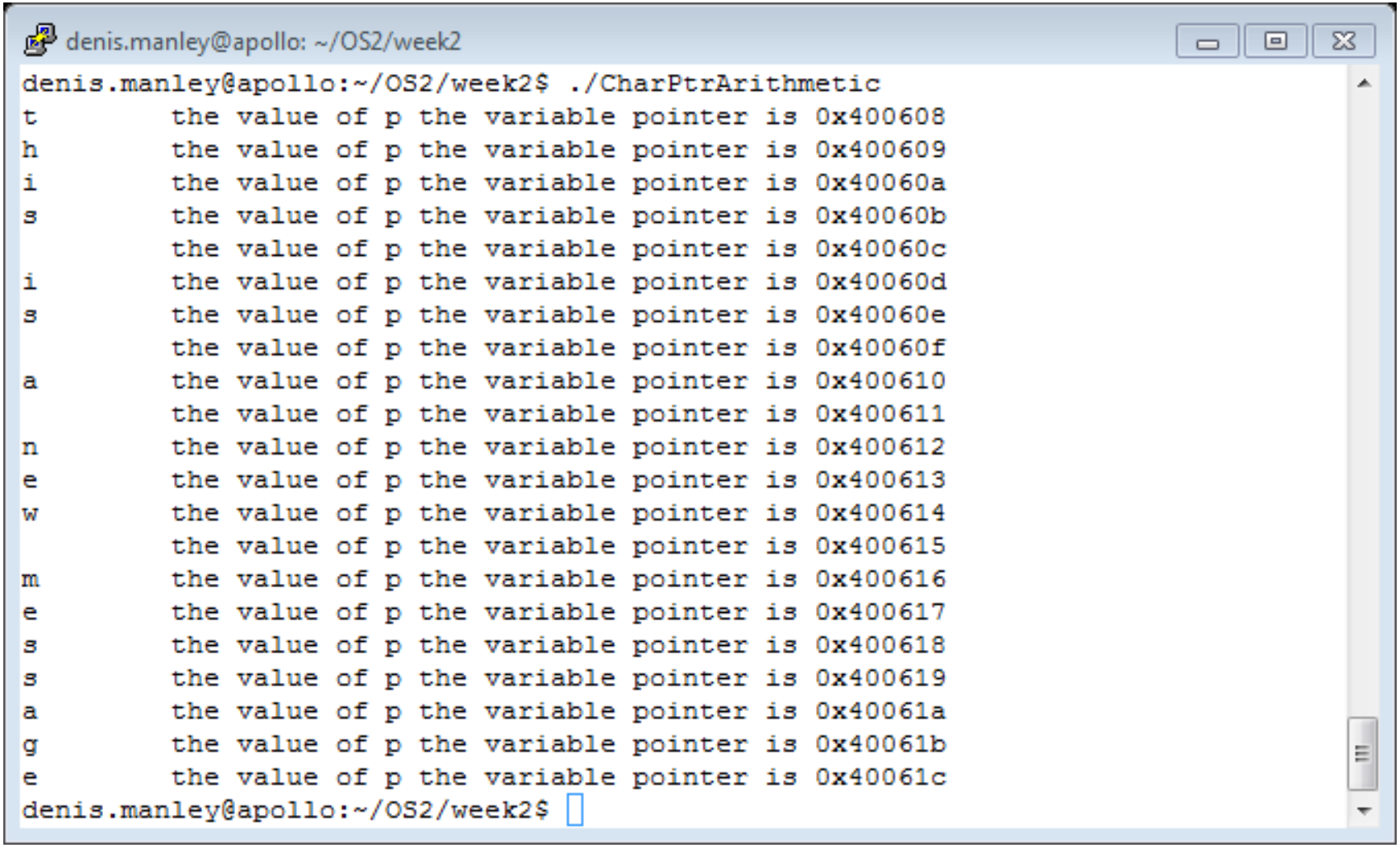
Char_pointer_arithmetic.c

denis.manley@apollo: ~/OS2/week2

```c
#include <stdio.h>

// Char_pointer_arithmetic.c
// What does this do:


int  main()
{
  char *p = "this is a new message";       /* p points to first character*/

  /* The next while loop is performed until p points to the null
          character '\0' at the end of the string */

  while ( *p != '\0' )
  {
          printf("%c\t the value of p the variable pointer is %p \n", *p, p);
   /* print each character in string */
          p++;                                  /* p now points to next character */
  }


  return 0;

}
```

24,0-1 All

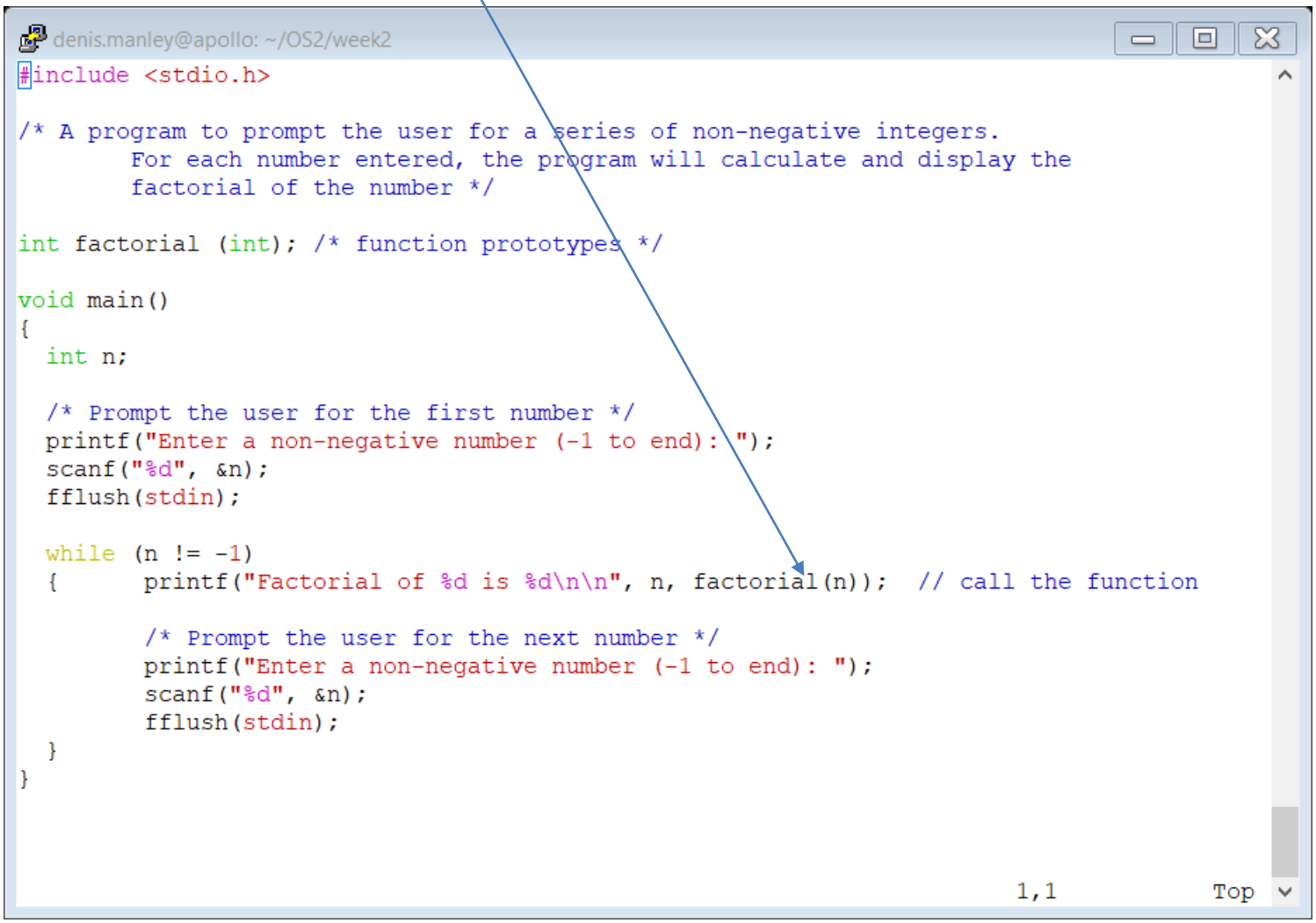
Explain Sample output



Functions

- A function is a way to allow the reuse of code
- There is three steps:
  - **Declare prototype** (tells compiler value returned, number of parameters, parameter type…)
  - **Call a function**: This is where program execution is directed to the function (most calls are in the main function but can be in any function: a function calls a function)
    - **DataType FunctionName (parameter list)**
  - **Function definition**
    - DataType FunctioName(parameter list including datatypes)
    - {
      - Function statements
      - Return value (if not void)
    - }

Calling the function



Function definition

```
/* Function definition */

int factorial (int num)    // function heading
{
  int fact = 1;

  while (num > 1)
  {
         fact = fact * num;
         num --;
  }

  return fact;  // return value (note must be of same data type)
}
```

Pass by Value/reference

- By Value
  - Does not change the variable (actual) that is being passed to a function
  - Just pass an variable by using variable name…
  - The *formal variable (in the function heading)* is then a copy of the *actual variable*

- By reference
  - When you want “to see if” the variable has been changed by the function
  - Must pass the address of the variable
  - Any changes are made by referencing the address passed to the function; the address of actual variable
  - By default an array is passed by reference. in order to prevent this use **const** in the array declaration

Swap example (by reference)

```
// reference Paul Kelly chapter 11
/* Program to demonstrate passing two arguments by reference. */
#include <stdio.h>
main()
{
  void swap( float *ptr1, float *ptr2 ) ;
  float num1, num2 ;

  printf( "Please enter two numbers: " ) ;
  scanf( "%f", &num1 ) ;
  scanf( "%f", &num2 ) ;     // step 1

  /* Swap values around so that the smallest is in num1. */
  if ( num1 > num2 )
    swap( &num1, &num2 ) ;
  printf( "The numbers in order are %.1f  %.1f\n", num1, num2 );
}
```

Swap function

- /* Function : swap
- Purpose : This function swaps two floating-point values.
- Arguments : pointers to the variables to be swapped. */
- void swap(float *ptr1, float *ptr2) //**step 2**
- {
- float temp ;

- temp = *ptr1 ; //**step 3**
- *ptr1 = *ptr2 ; //**step 4**
- *ptr2 = temp ; // **step 5**
- }

Strings

- A string is very similar to a character array:
  - What is the difference between them? Hint : '/0'

- Declaration of a string
  - Use char name[SIZE];
  - Input values to a string using scanf ("%s", …) or fgets(name, SIZE, stdin).
  - Output results with **printf("%s", name)** or **puts** (name)

Common String Functions

- #include<string.h>
- **Strlen**: determines number of characters (not including the '\0';
  - len = strlen(string);
- **strcpy (destination , source)**; copies contents of source to destination
  - Destination and source are strings
  - {ensure destination is big enough to take source}.

Common String Functions

- **Strcmp(str1, str2)**
  - Compares two null terminated string and returns
    - Zero if str1 == str2
    - Returns A negative value if str1 < str2
    - Returns A positive value if str1 > str2

  - if **(strcmp(password, user) == 0)**
  - printf(“Password correct. Welcome to the system”);
  - else
  - printf(“Invalid password”);

Write the strlen using char pointers

- Modify the char pointer arithmetic code to calculate strlen function.

denis.manley@apollo: ~/OS2/week2

```
#include <stdio.h>

// Char_pointer_arithmetic.c
// What does this do:


int  main()
{
  char *p = "this is a new message";       /* p points to first character*/

  /* The next while loop is performed until p points to the null
          character '\0' at the end of the string */

  while ( *p != '\0' )
  {
          printf("%c\t the value of p the variable pointer is %p \n", *p, p);
   /* print each character in string */
          p++;                                  /* p now points to next character */
  }


  return 0;

}
```

24,0-1 All

Pointers to pointers

- A pointer can also pointer to another pointer which in turn pointers to a “standard” variable:

  - **int i=3; // an integer variable**
    int *j; // a pointer
    int **k; // a pointer to a pointer (double pointer)

  - **j=&i; //line 1 (assigned address of an integer)**

  - **k=&j; //line 2 (assigned the address of a pointer)**

Examples

- **What is output of the following statements**

  - **printf(“%d”, **k);**
  - **printf(“%p”, *k);**
    printf(“%d”,*j);
    printf(“%d”,i);

  - **Assume the following: i = 3, j = &I; k = &j**

Double_pointer code and output

denis.manley@apollo: ~/OS2/week2

```
#include<stdio.h>

int  main()
{

    int i = 3;
    int *j;   //pointer to integer
    int **k;  // pointer to pointer (double indirection)

    // assign value to pointer and double pointer
    j = &i;
    k = &j;

    // ouput results

    printf(" the value of i is %d\n", i);
    printf("the value of j is %p\n", j);
    printf ("the value of k is %p\n\n", k);

    // output addresses

   printf(" the address if i is: %p \n", &i);
   printf(" the address of j is %p\n", &j);
   printf(" the address of k is %p\n", &k);

    // using indirection to get value of i;

    printf(" the value of i is %d\n", i);
    printf("the value of *j is %d\n", *j);
    printf("the value of k is %p\n", k);
    printf("the value of *k is %p\n", *k);
    printf ("the value of **k is %d\n", **k);

   return 0;
-- INSERT --                                  6,5           Top
```

denis.manley@apollo: ~/OS2/week2

```
denis.manley@apollo:~/OS2/week2$ ./L2Q3_double_pointer
 the value of i is 3
the value of j is 0x7ffd6a0f4e4c
the value of k is 0x7ffd6a0f4e50

 the address if i is: 0x7ffd6a0f4e4c
 the address of j is 0x7ffd6a0f4e50
 the address of k is 0x7ffd6a0f4e58
 the value of i is 3
the value of *j is 3
the value of k is 0x7ffd6a0f4e50
the value of *k is 0x7ffd6a0f4e4c
the value of **k is 3
denis.manley@apollo:~/OS2/week2$
```