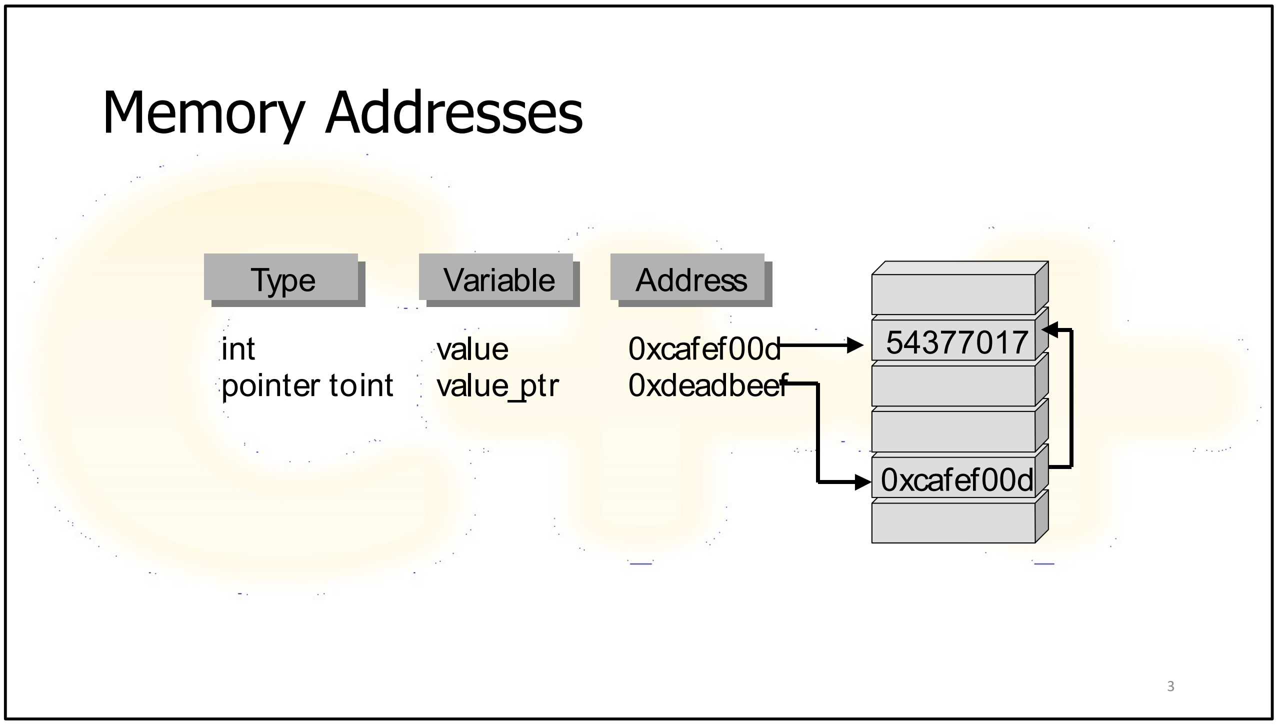
Pointers

Contents

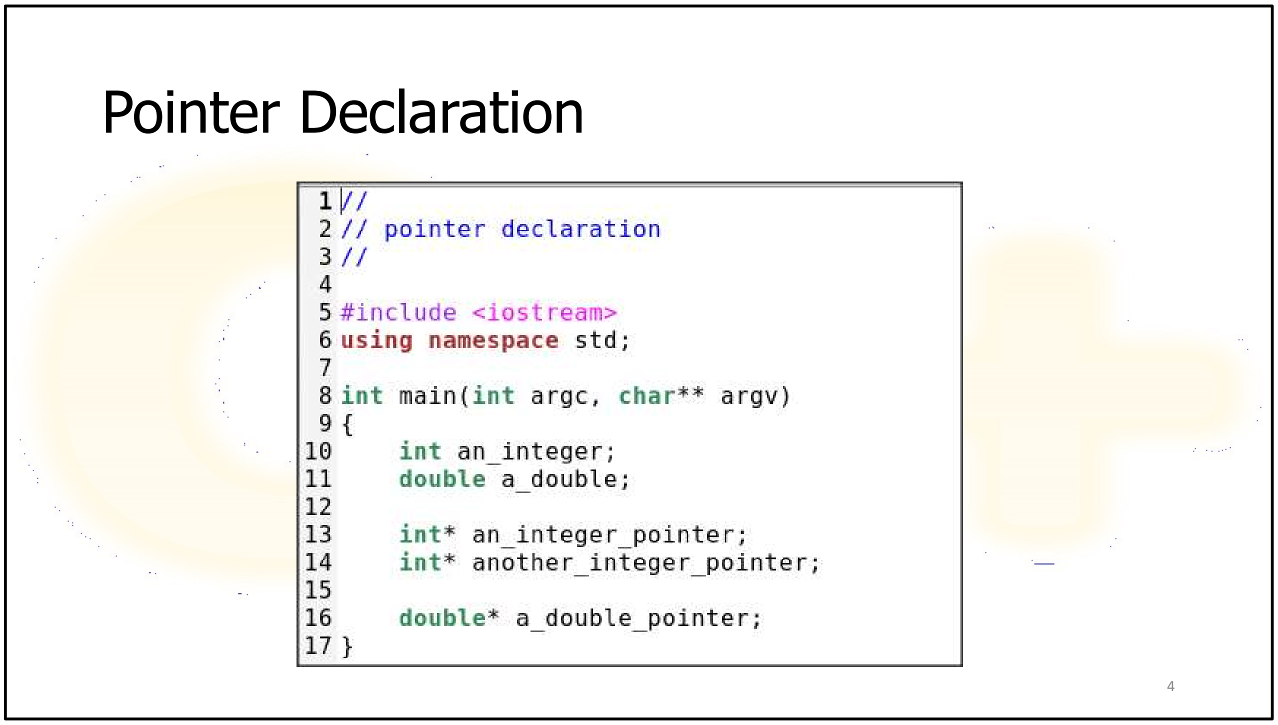
- Objectives
  - To be able to declare and use pointers.
- Contents
  - Memory Addresses
  - Pointer Declaration
  - Pointer Manipulation
  - Pointers to Structures
  - Strings
  - Pointer to char
  - Character Manipulation Functions
  - Pointers to Pointers
  - Null Pointers
  - Smart Pointers



Memory Addresses

Type	Variable	Address
int	value	0xcafef00d
pointer toint	value_ptr	0xdeadbeef

Pointers are variables that contain the addresses of another variable.



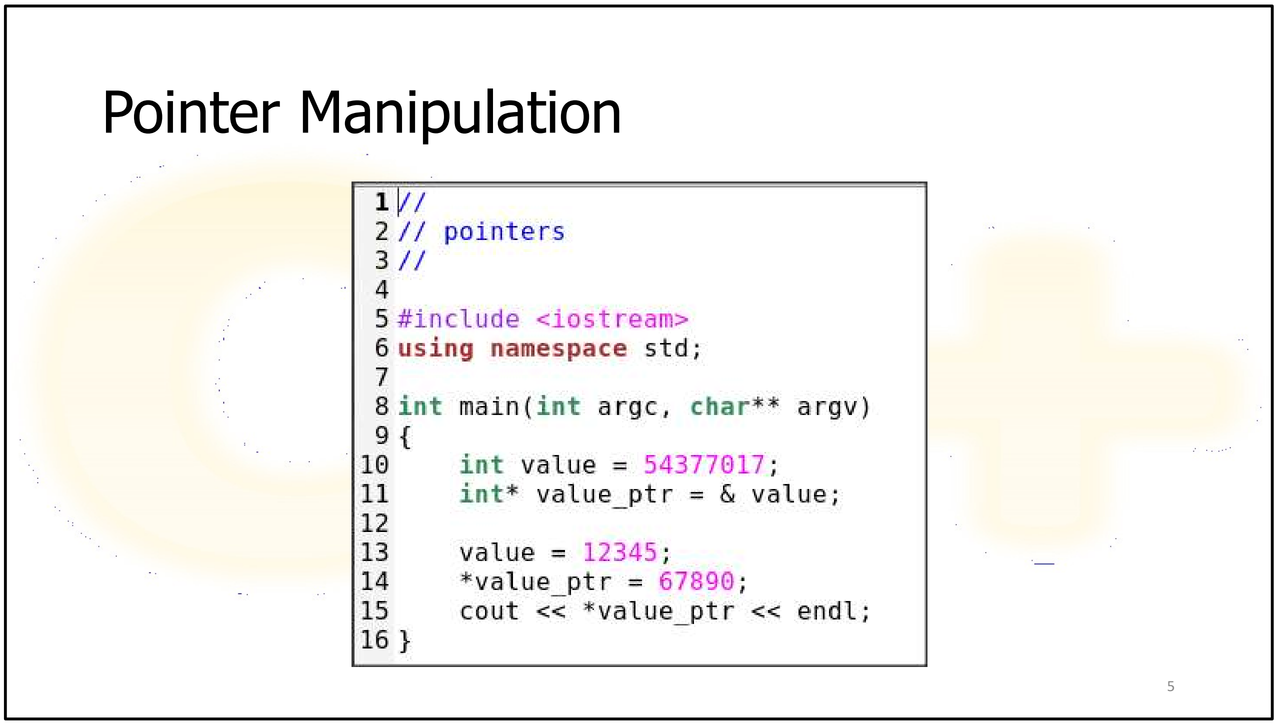
Pointer Declaration

```
//
// pointer declaration
//

#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
    int an_integer;
    double a_double;

    int* an_integer_pointer;
    int* another_integer_pointer;

    double* a_double_pointer;
}
```

4

Pointers are declared as pointer to type. The initial value of a pointer is undefined. Referencing the memory pointed to by an uninitialised pointer will have unpredictable and sometimes disastrous results. SOME compilers attempt to detect this & warn you at compile time.

Pointers, just like any other simple variable type, can be assigned to pointers of the same type. However, since C++ is a strongly typed language pointers to different types cannot be assigned.



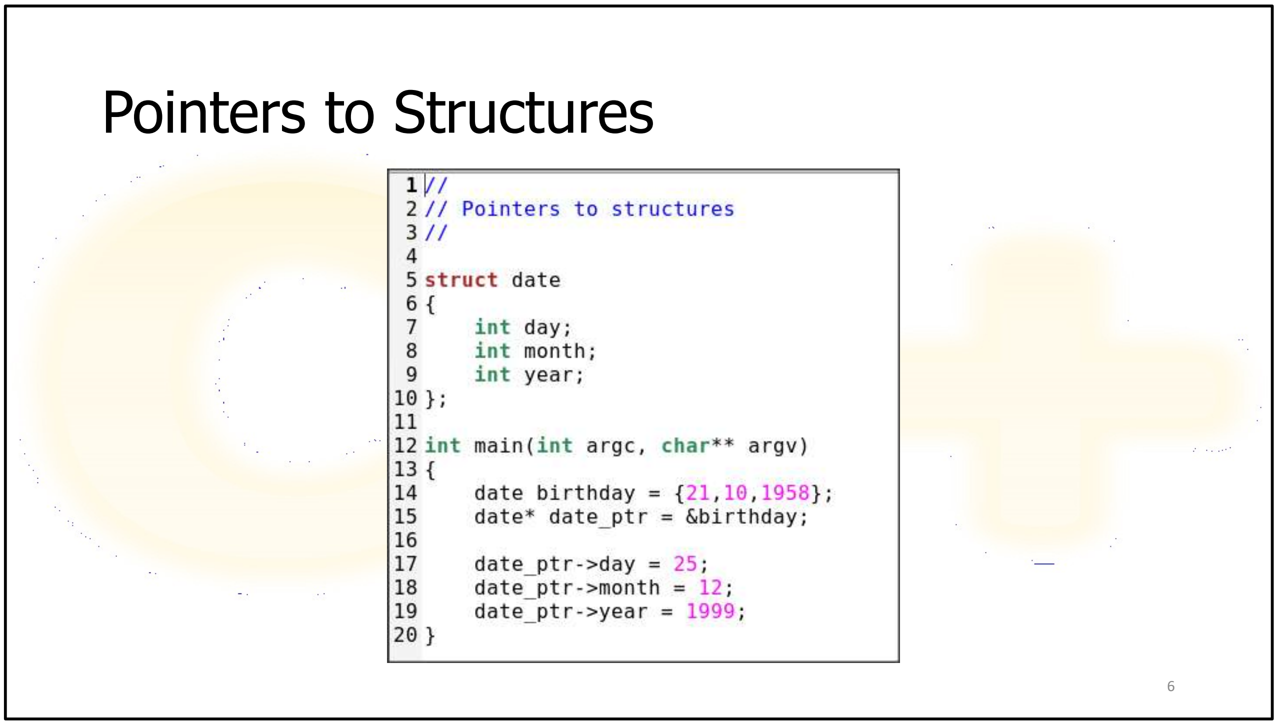
Pointer Manipulation

```
//
// pointers
//

#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
    int value = 54377017;
    int* value_ptr = & value;

    value = 12345;
    *value_ptr = 67890;
    cout << *value_ptr << endl;
}
```

5

The & operator gives the address of a previously defined variable. The compiler will pick up any misuse such as trying to assign the address of a double to a variable of type pointer to integer. The & operator is sometimes called the address of operator.

The * operator returns the contents of what is pointed to. This is known as dereferencing a pointer. It is sometimes called the contents of operator.

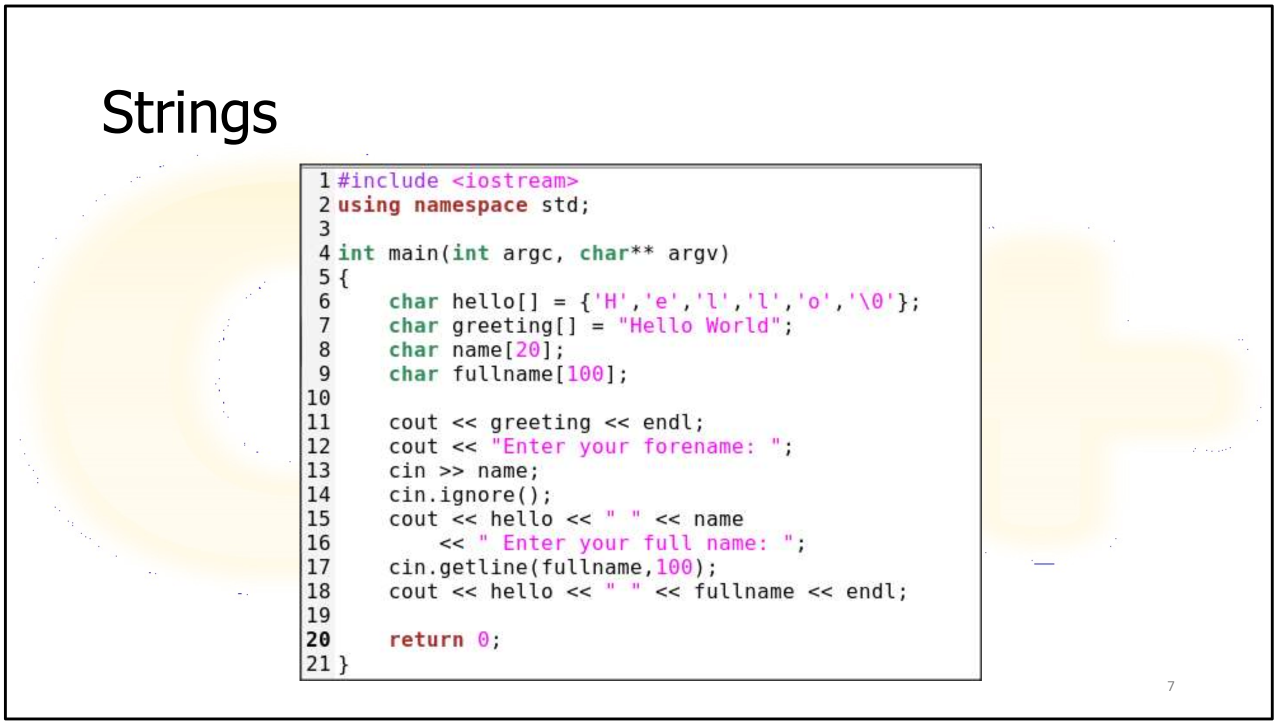


Pointers to Structures

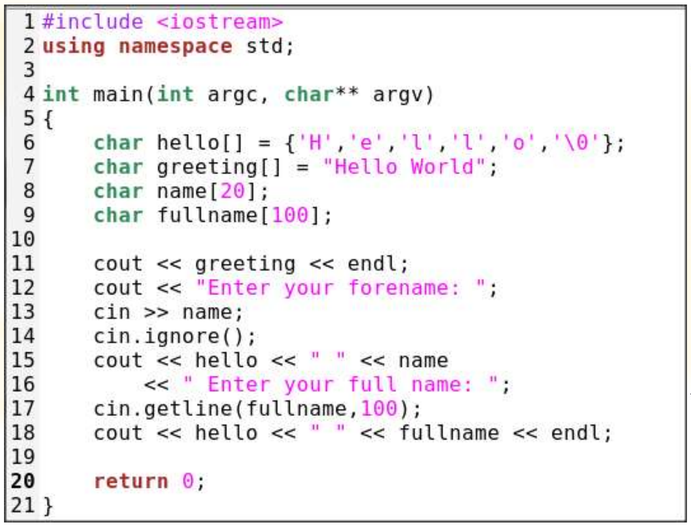
```
//
// Pointers to structures
//

struct date
{
    int day;
    int month;
    int year;
};

int main(int argc, char** argv)
{
    date birthday = {21,10,1958};
    date* date_ptr = &birthday;

    date_ptr->day = 25;
    date_ptr->month = 12;
    date_ptr->year = 1999;
}
```

6

Note the line date_ptr->day = 25; This line could also be written (*date_ptr).day = 25; It could not be written *date_ptr.day = 25; This is because the dot operator has a higher precedence than the * operator and we need to dereference the pointer before we attempt to reference a member of the structure. No one in their right minds would choose the second way of writing the statement.



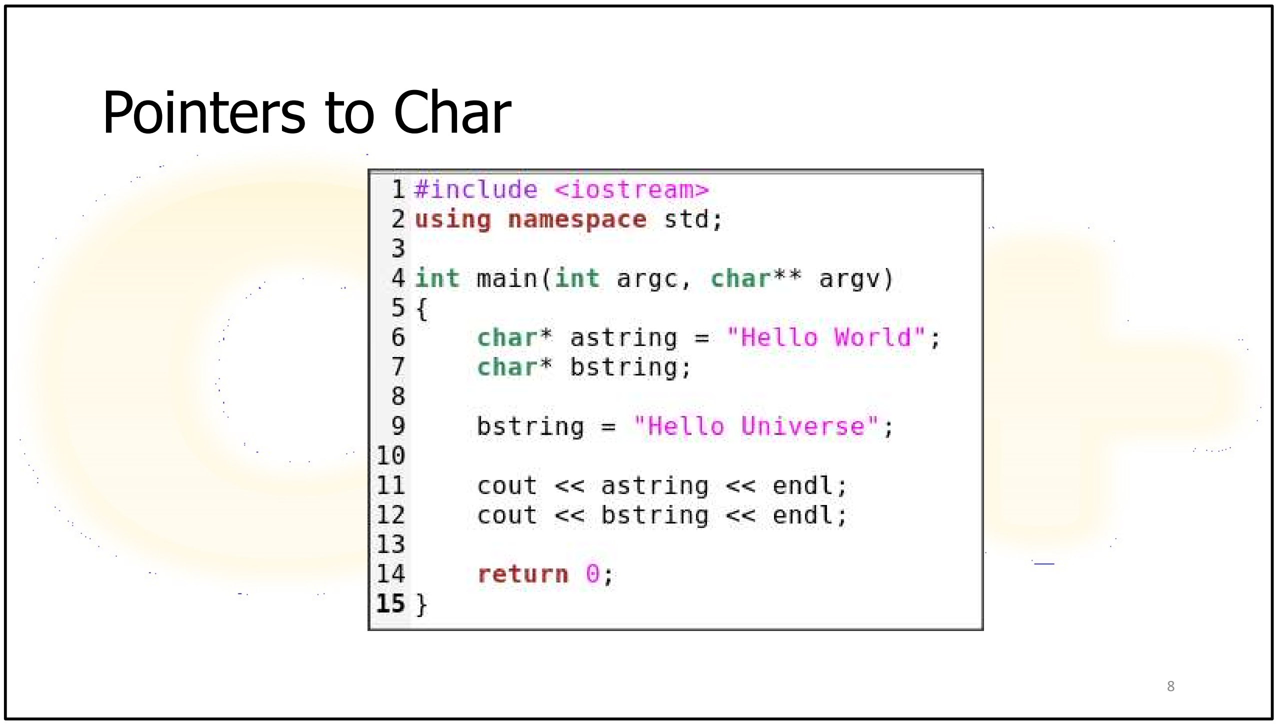
Strings



```
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
    char hello[] = {'H','e','l','l','o','\0'};
    char greeting[] = "Hello World";
    char name[20];
    char fullname[100];

    cout << greeting << endl;
    cout << "Enter your forename: ";
    cin >> name;
    cin.ignore();
    cout << hello << " " << name
        << " Enter your full name: ";
    cin.getline(fullname,100);
    cout << hello << " " << fullname << endl;

    return 0;
}
```

7

C++ does not have a native string data type. If you want to store strings in your program you must use another method. What is a string? It is a series of characters terminated by a null character. Null can be represented as '\0'. We could use a char array to store a series of characters.



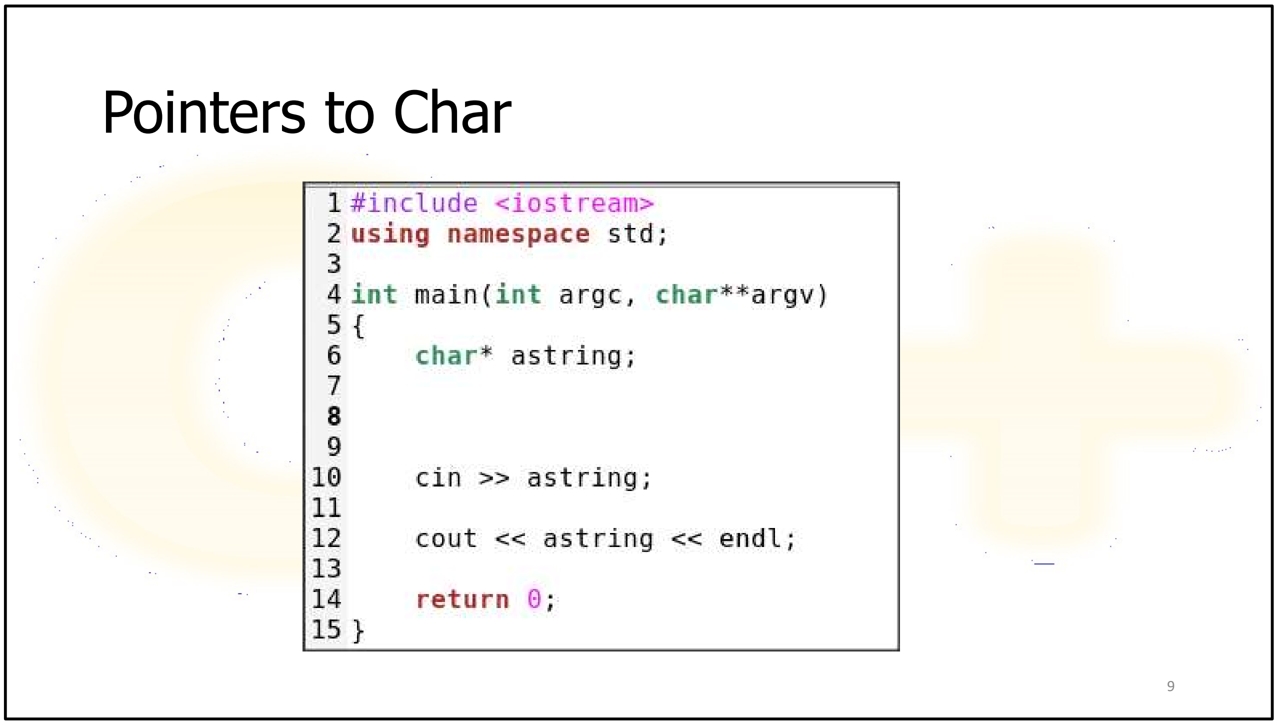
Pointers to Char

```
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
    char* astring = "Hello World";
    char* bstring;

    bstring = "Hello Universe";

    cout << astring << endl;
    cout << bstring << endl;

    return 0;
}
```

8

Pointers to characters are often used to store strings. In fact, char* is far more common than a char array. The char * pointer is treated by C++ in a slightly different way to other pointers. Instead of pointing to a single character, as it's declaration would suggest, a char * pointer points to the first character in a null terminated string.

the storage that is allocated is in a special area of memory that is reserved for constants. Any attempt to read into the astring or bstring variables will result in an error.

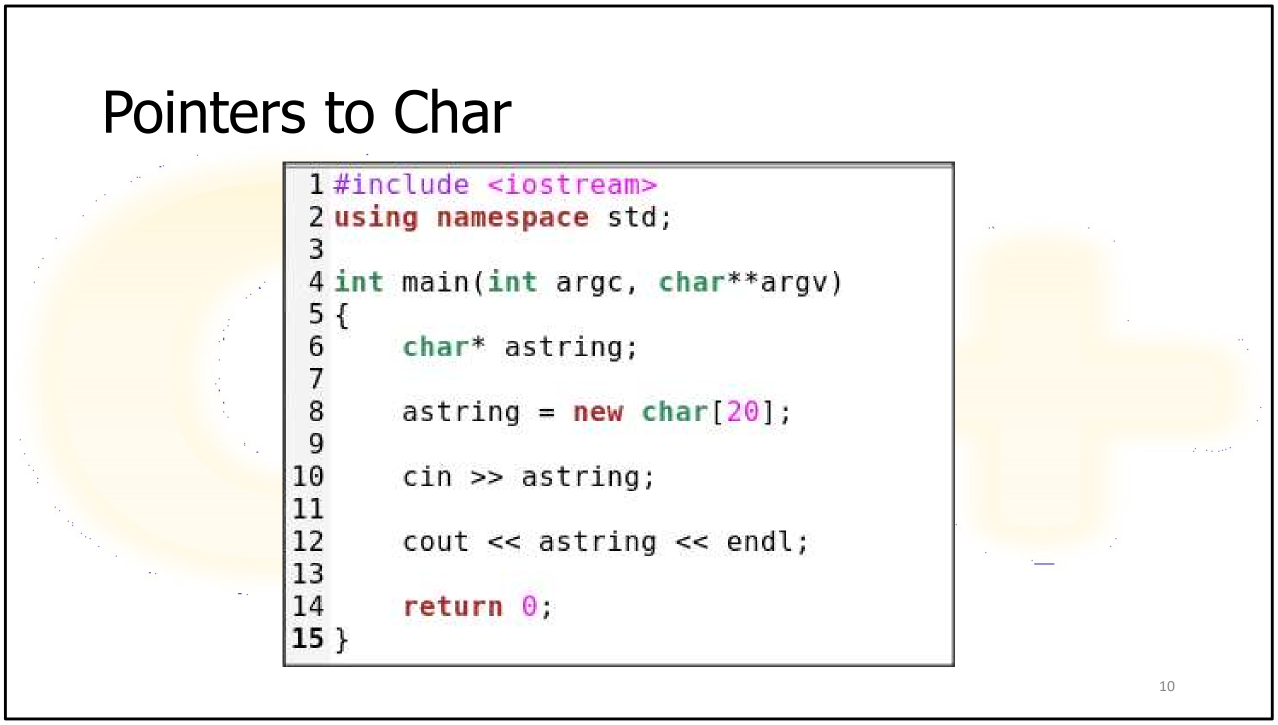


Pointers to Char

```
#include <iostream>
using namespace std;

int main(int argc, char**argv)
{
    char* astring;



    cin >> astring;

    cout << astring << endl;

    return 0;
}
```

9

In order for this example to work we must first allocate some memory to astring at run time for the string to be stored in.

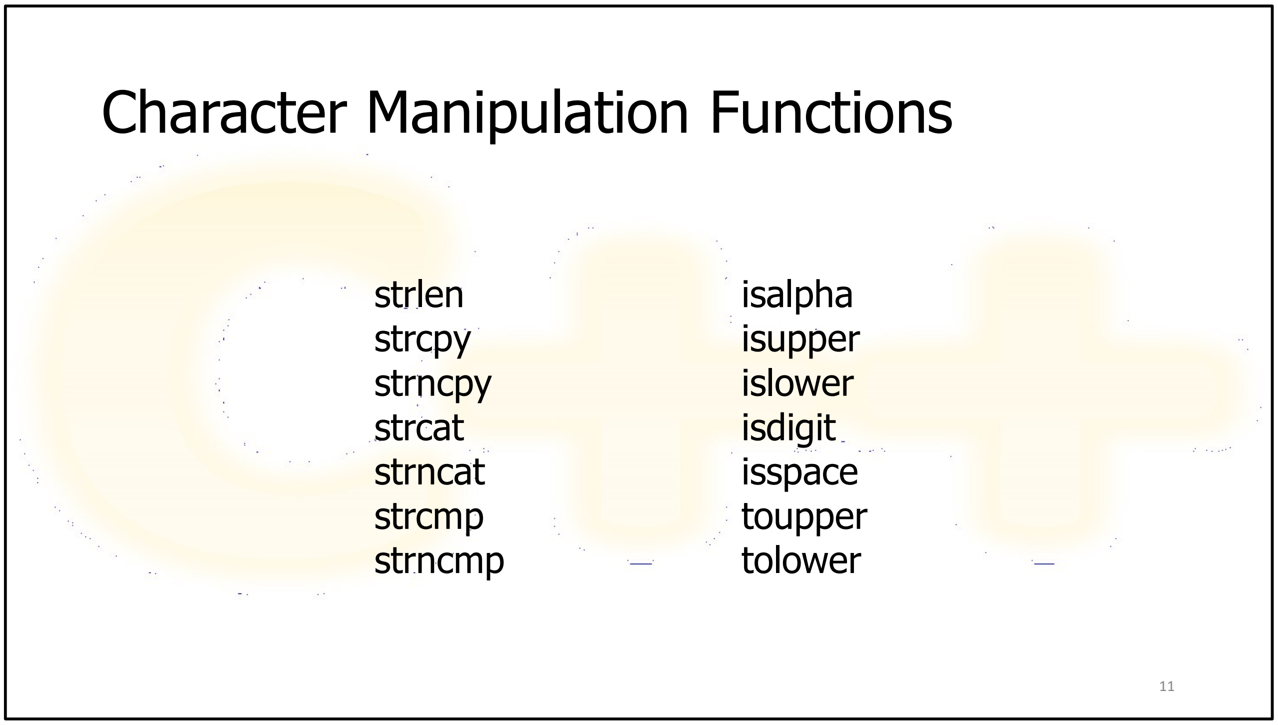


Pointers to Char

```
#include <iostream>
using namespace std;

int main(int argc, char**argv)
{
    char* astring;

    astring = new char[20];

    cin >> astring;

    cout << astring << endl;

    return 0;
}
```

10

This example allocates the memory before the string is used. Provided the string does not exceed 19 bytes (plus the null terminator) the code will work.



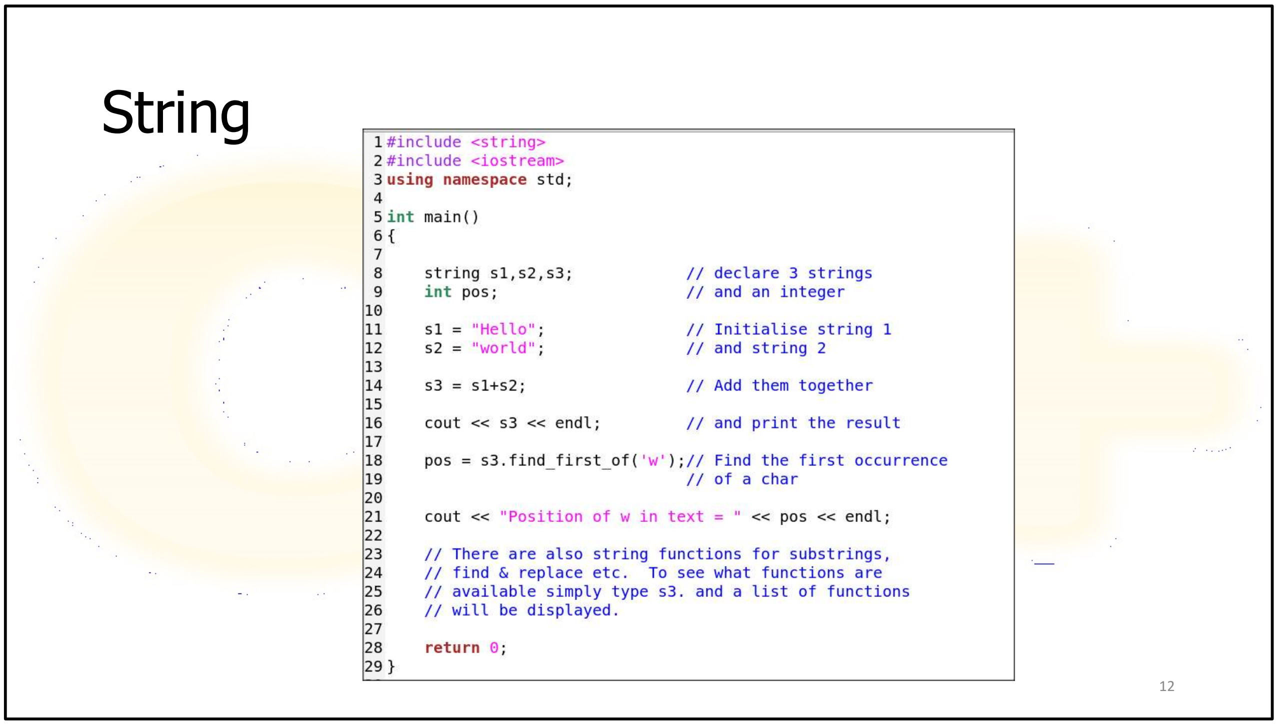
Character Manipulation Functions

strlen
strcpy
strncpy
strcat
strncat
strcmp
strncmp

isalpha
isupper
islower
isdigit
isspace
toupper
tolower

11

The standard C library contains a set of functions to perform both simple and complex character manipulations using char* pointers (or char arrays!). The C library has been considerably changed in the C++11 standard. However, the changes are optional & some compilers do not implement them as standard.



String

```cpp
#include <string>
#include <iostream>
using namespace std;

int main()
{

    string s1,s2,s3;            // declare 3 strings
    int pos;                    // and an integer

    s1 = "Hello";               // Initialise string 1
    s2 = "world";               // and string 2

    s3 = s1+s2;                 // Add them together

    cout << s3 << endl;         // and print the result

    pos = s3.find_first_of('w');// Find the first occurrence
                                // of a char

    cout << "Position of w in text = " << pos << endl;

    // There are also string functions for substrings,
    // find & replace etc.  To see what functions are
    // available simply type s3. and a list of functions
    // will be displayed.

    return 0;
}
```

12

As you will well know, C++ has little or no string handling capabilities built into it. The string class is designed to rectify this shortcoming of the language. It provides rich, simple string manipulation.

However, functionality comes at a price. This price is paid both in terms of memory usage and also processor usage.

Exercise

You will be shown a series of pointer code snippets. Your task is to work out the answers to the questions after each snippet.

Exercise

a)

```
int m, n;
int *iptr;

m = 38;
iptr = &m;
n = *iptr;
```

What is the value of `n`?

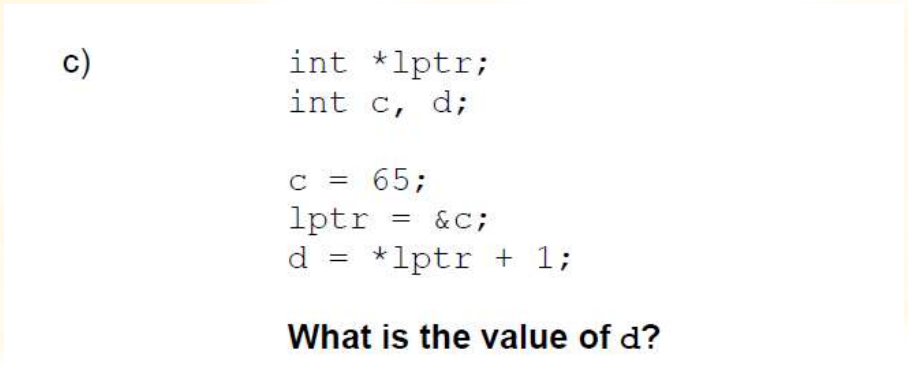
Exercise

b)

```
int m, n;
int *iptr;

n = 10;
iptr = &n;
n = 11;
m = *iptr;
```

What is the value of `m`?

Exercise

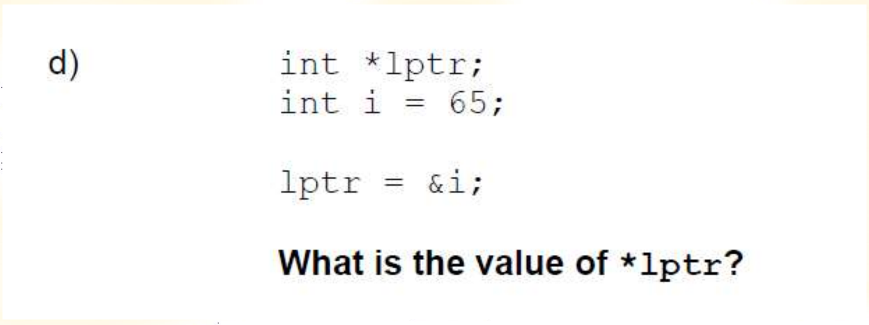


c)

```
int *lptr;
int c, d;

c = 65;
lptr = &c;
d = *lptr + 1;
```

What is the value of `d`?

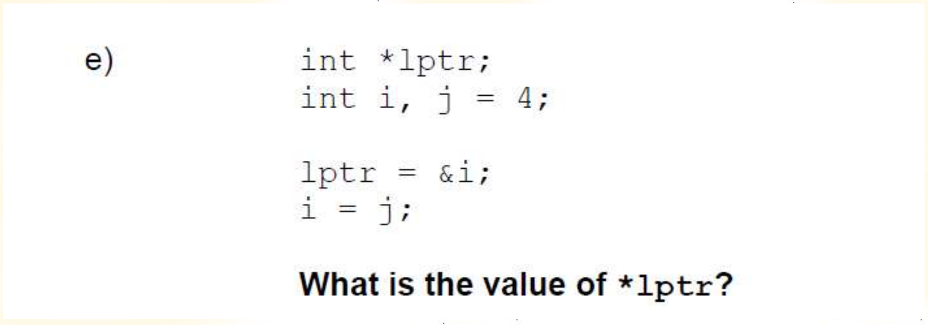
Exercise


d)

```
int *lptr;
int i = 65;

lptr = &i;
```

What is the value of `*lptr`?

Exercise

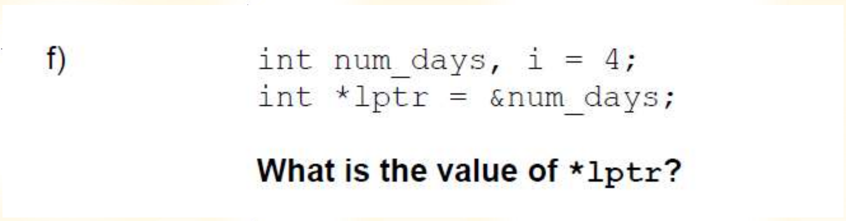


e)

```
int *lptr;
int i, j = 4;

lptr = &i;
i = j;
```

What is the value of `*lptr`?

Exercise

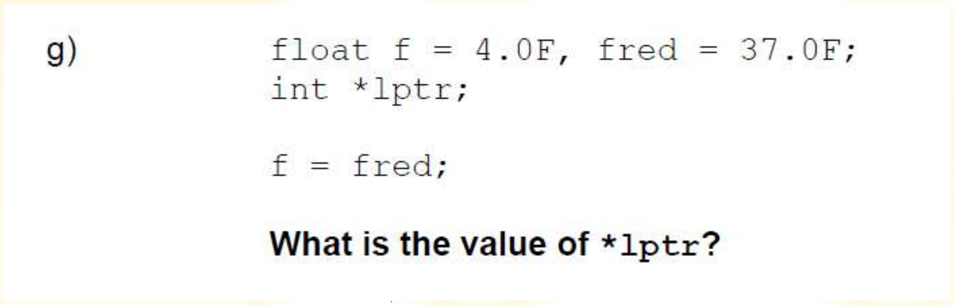


f)

```
int num_days, i = 4;
int *lptr = &num_days;
```

What is the value of `*lptr`?

Exercise



g)

```
float f = 4.0F, fred = 37.0F;
int *lptr;

f = fred;
```

What is the value of `*lptr`?

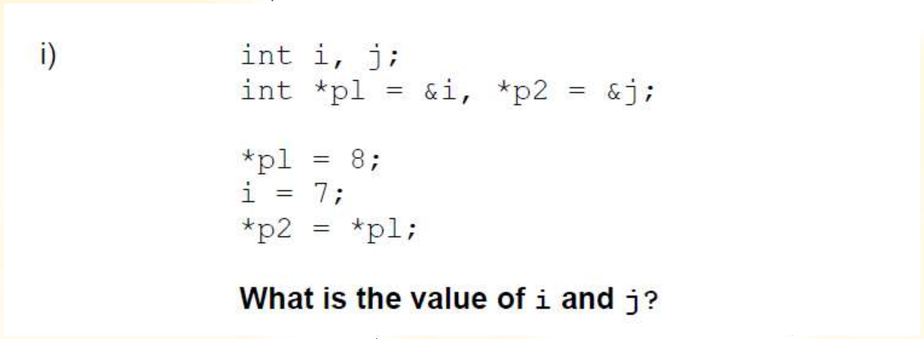
Exercise

h)

```
int i = 9, j = 10;
int *lptr = &i;

*lptr = i;
j = i;
```

What is the value of `*lptr`, `i` and `j`?

Exercise

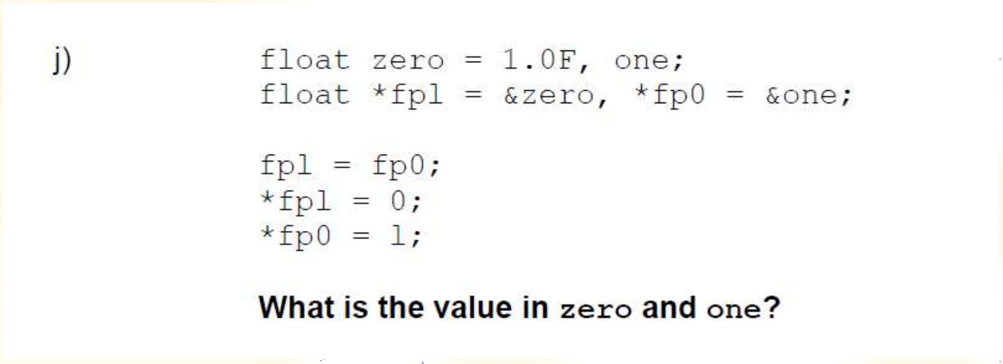


i)

```
int i, j;
int *pl = &i, *p2 = &j;

*pl = 8;
i = 7;
*p2 = *pl;
```

What is the value of `i` and `j`?

Exercise

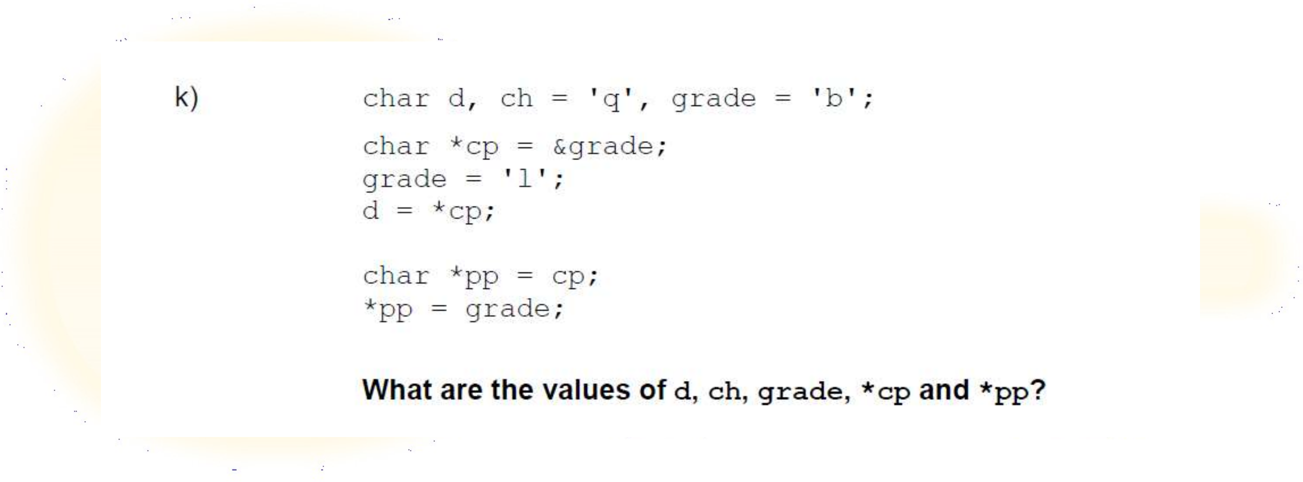


j)

```
float zero = 1.0F, one;
float *fpl = &zero, *fp0 = &one;

fpl = fp0;
*fpl = 0;
*fp0 = l;
```

What is the value in `zero` and `one`?

Exercise



k)

```
char d, ch = 'q', grade = 'b';
char *cp = &grade;
grade = 'l';
d = *cp;

char *pp = cp;
*pp = grade;
```

What are the values of `d`, `ch`, `grade`, `*cp` and `*pp`?

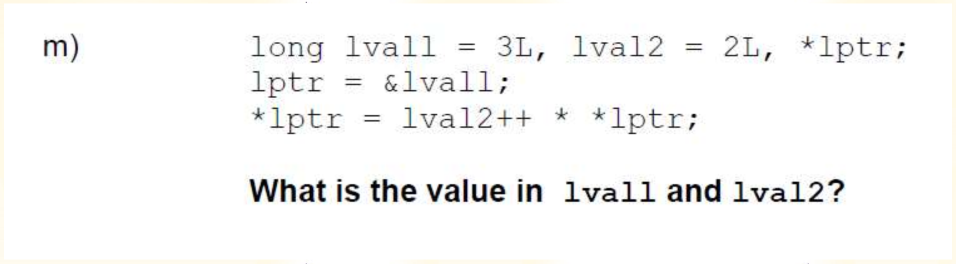
Exercise

l)

```
float f, fl = 4.0F, f2 = 1.5F;
float *fpl = &fl, *fp2 = &f2;

f = *fpl * *fp2;
```

What is the value in `f`?

Exercise

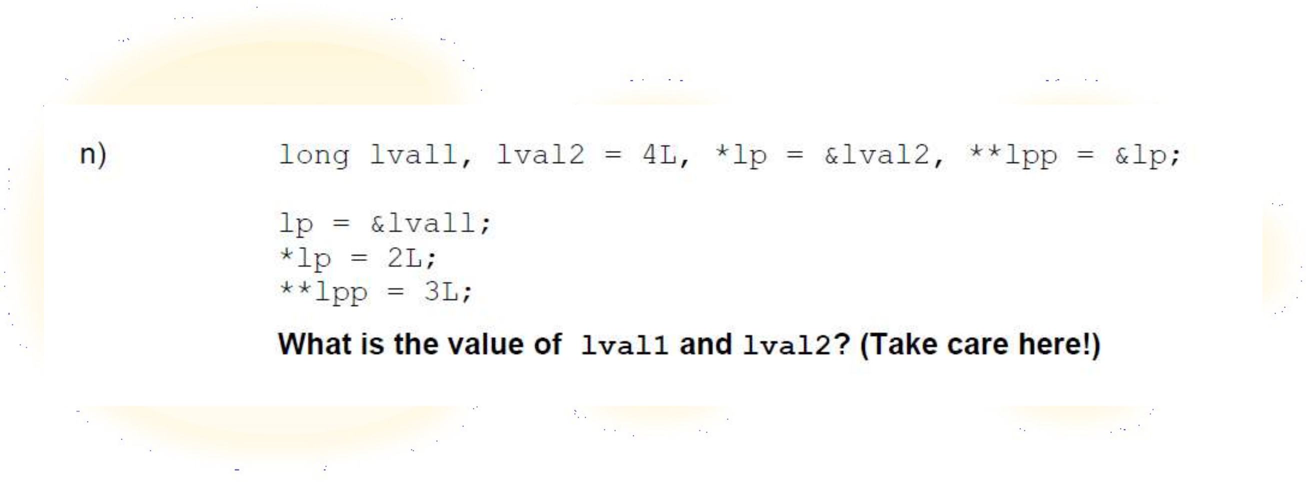


m)

```
long lval1 = 3L, lval2 = 2L, *lptr;
lptr = &lval1;
*lptr = lval2++ * *lptr;
```

What is the value in `lval1` and `lval2`?

Exercise



n)

```
long lval1, lval2 = 4L, *lp = &lval2, **lpp = &lp;

lp = &lval1;
*lp = 2L;
**lpp = 3L;
```

What is the value of `lval1` and `lval2`? (Take care here!)