Topic 2-2

Pointers

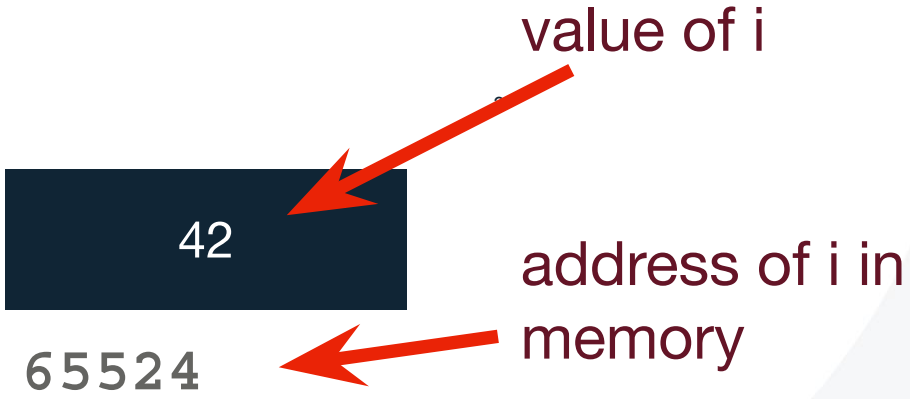
Welcome

- What are pointers.
- Pointers arithmetics.

THE UNIVERSITY of ADELAIDE

Data Variable vs a Pointer

```
int i=42;
```

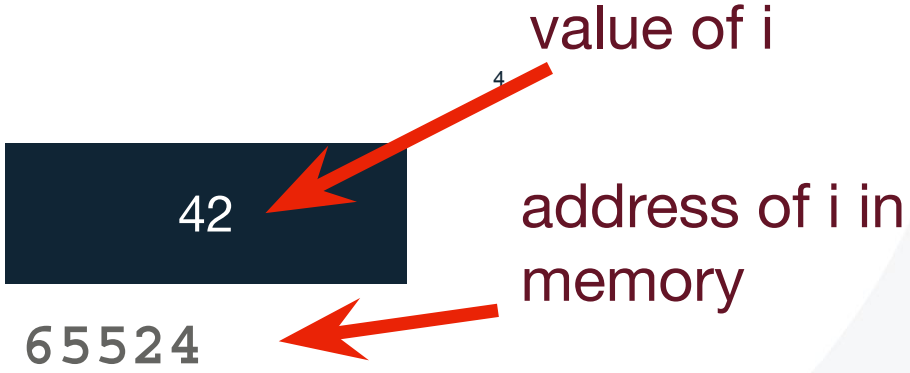


THE UNIVERSITY of ADELAIDE

Pointers store addresses

- A pointer stores the address of a memory location
- It is like a normal variable, but its value point to memory

```
int i=42;

int *j;

j = &i;
```



j IS CALLED A POINTER

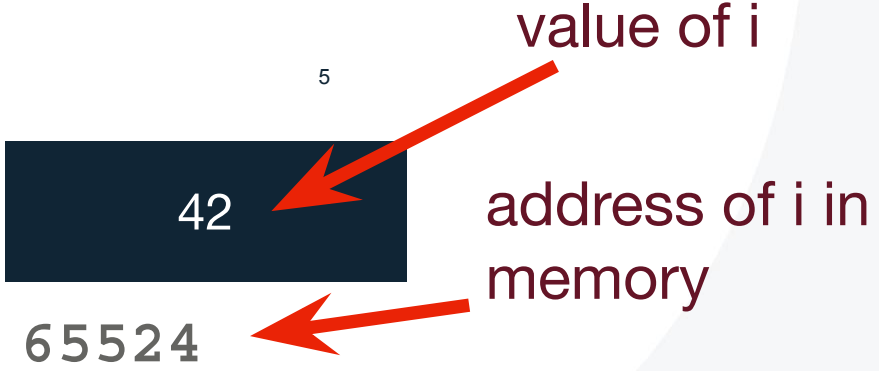
j stores the address of i in memory

THE UNIVERSITY of ADELAIDE

Pointer Definition: &, *

- **Pointers are annotated with a * symbol before their name**
- **& returns the address of any variable in memory, which can be assigned to a pointer**

```
int i=42;

int *j;

j = &i;
```



Demo 1

```cpp
#include <iostream>

using namespace std;


int main()
{
    int *ptr, i;
    i = 11;

    /* address of i is assigned to ptr */
    ptr = &i;

    /* show the value of ptr */

    cout << "Value of ptr: "  << ptr << endl;

    /* show i's value using ptr variable */
    cout << "Value of i: "  << *ptr << endl;

    return 0;
}
```

Pointer Operation: *

- **Pointers are just like any other variable, with the understanding that they refer to memory location**
- **If I know a memory location, can I get the value stored at that location?**

```
int i=42,k;
int *j;
j = &i;
k = *j;
```

LINGO

- * is called the “value-at” operator
- the **operation** is called de-referencing

THE UNIVERSITY of ADELAIDE

De-referencing

- **Any operation performed on the de-referenced pointer directly affects the variable it points to**
- **What is the value of i after this?**


```
int i=42;
int *j;
j = &i;
(*j)++;
```

- note the precedence: *j++ is different from (*j)++
- also, int *j is different from *j

THE UNIVERSITY of ADELAIDE

Demo 2

```cpp
#include <iostream>

using namespace std;

int main()
{
    int *ptr, i;
    i = 11;

    /* address of i is assigned to ptr */
    ptr = &i;

    /* increment the value of i */
    (*ptr)++;

    /* show i's value using ptr variable */
    cout << "Value of i: "  << *ptr << endl;

    return 0;
}
```

Um ... but why?

- **Why go through all this trouble, to use *, & and combinations thereof?**
- **Instead of passing large quantities of data between functions, we can just pass the location (pointer!) to the start of the data**
  - saves memory, saves computation
- **Analogy: let's say an Adelaide painter needs to paint three blocks of flats and a mall in Ballarat, Victoria**
  - I do not transport the blocks of flats and the mall to the painter in Adelaide, but give the painter the address in Ballarat where the painting needs to happen
- **Pointer arithmetic allows the painter (or the function) to work with memory locations starting from a given address**

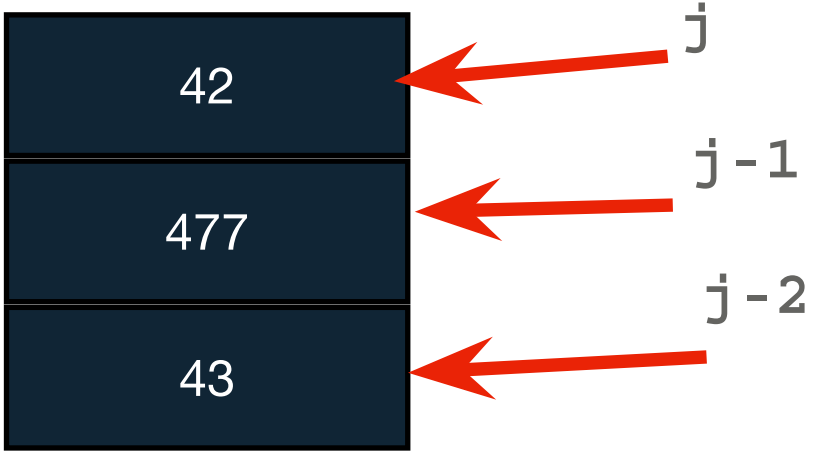
THE UNIVERSITY of ADELAIDE

Pointer Arithmetic

- **Pointer = a variable that stores the address of another variable**
- **Still a variable! => subject to arithmetic operations**
  - ... but dependent on size of the pointer

THE UNIVERSITY of ADELAIDE

Arithmetic



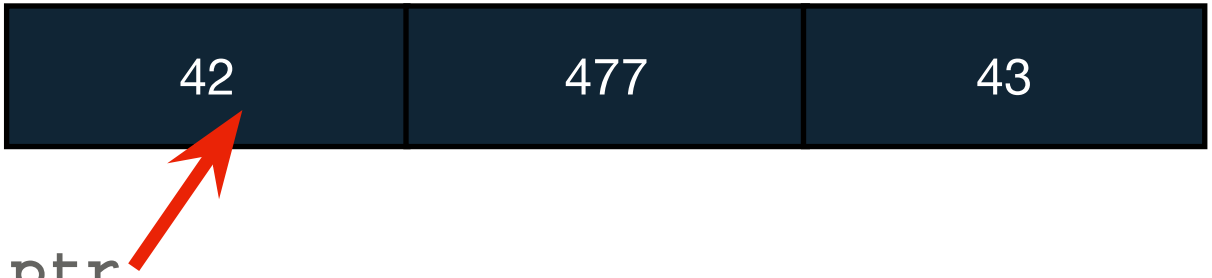

```
int* j;

assume 3 integers are
allocated one after
another

int i=42,k=477,l=43;
j=&i;
```

THE UNIVERSITY of ADELAIDE

Demo 3

```
#include <iostream>

using namespace std;

int main()
{
    int *ptr, i=42, k=477, l=43;

    /* address of i is assigned to ptr */
    ptr = &i;

    cout<< "Value of i " << *ptr << endl;
    cout<< "Value of k " << *(ptr+1) << endl;
    cout<< "Value of l " << *(ptr+2) << endl;

    // on some computers you need to do
    //cout<< "Value of i " << *ptr << endl;
    //cout<< "Value of k " << *(ptr - 1) << endl;
    //cout<< "Value of l " << *(ptr - 2) << endl;

    return 0;
}
```

Pointers and Arrays




```
int* ptr;
int a[3]={42,477,43}
ptr = &a[0];
```

THE UNIVERSITY of ADELAIDE

Demo 4

```cpp
#include <iostream>

using namespace std;

int main()
{
    int *ptr, a[3]={43,477,34};

    /* address of a[0] is assigned to ptr */
    ptr = &a[0];

    cout << "Value of a[0] " << *ptr << endl;
    cout << "Value of a[1] " << *(ptr+1) << endl;
    cout << "Value of a[2] " << *(ptr+2) << endl;


    return 0;
}
```

sizeof

- **Pointer arithmetic (+/-) shifts the address a number of bytes equal to the size of the pointer type**
- **You can get the size of the type by using the sizeof operator**
  - will return the number of bytes used to represent any variable of that type

```
int *ptr,i,k;
sizeof(i);
4
```

```
char c;
sizeof(c);
1
```

THE UNIVERSITY of ADELAIDE

Demo 5

```cpp
#include <iostream>

using namespace std;

int main()
{
    int *ptr,i,k ;
    int *ptr2,L ;

    /* address of i is assigned to ptr */
    ptr = &i;

    /* ptr2 is assigned address of next int after i */
    ptr2 = ptr + 1 ;

    /* L is the number of ints between ptr and ptr2 */
    L = ptr2 - ptr ;

    cout << "Value of i    " << *ptr << endl;
    cout << "Value of ptr? " << *(ptr+1)<< endl;
    cout << "Value of ptr  " <<  (unsigned long) ptr << endl;
    cout << "Value of ptr2 " << (unsigned long) ptr2 << endl;
    cout << "Value of L    " << L << endl;

    return 0;
}
```

What does this do?

```
int *ptr,i,k;
int *ptr2, L;
ptr = &i;
ptr2 = ptr + 1;
L = ptr2 - ptr ;
```


THE UNIVERSITY of ADELAIDE

What does this do?

```
int *ptr,i,k;
int *ptr2,L;
ptr = &i;
ptr2 = ptr + 1;
L = ptr2 - ptr ;
```

- addr(ptr + i) = addr(ptr) + [sizeof(T) * i]
- where T is the pointer type (int, char, ...)
- when two pointers are of the same type then the difference between them is defined the number of things of the pointed-to type between them.

THE UNIVERSITY of ADELAIDE

What is allowed?

Address + Number	Address
Address - Number	Address
Address - Address	Number
Address + Address	Illegal

THE UNIVERSITY of ADELAIDE