
CRICOS PROVIDER 00123M

School of Computer Science

# COMP SCI 2103/7103 Algorithm Design & Data Structure

## Review of Pointers

adelaide.edu.au

seek LIGHT

# Review of Pointers

In this lecture, we are going to give more details and see more examples on:

- The concepts of pointers
- Talk about pointer arithmetic
- Discuss arrays and strings

# Review of Pointers

- Pointer: the memory address of a variable.

- How to define and use them?

```
double d;
double * ptr=&d;
cout << ptr <<endl << *ptr << endl;
```

# Review of Pointers

- Pointers are variables of fixed size.

- But are they exactly the same thing as static variables?

```
int p;
int *ptr;
```

  - Abstraction
  - Compile time: the compiler knows some more information about a pointer variable

ptr=ptr+1; (increments ptr by sizeof(*ptr))

# Pointer Arithmetic

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 main() {
 5     int x, *xp;
 6     double y, *yp;
 7     char z, *zp;
 8     class {
 9         int temp[1000];
10     } c, *cp;
11
12     cout << sizeof(x) << endl;
13     cout << sizeof(y) << endl;
14     cout << sizeof(z) << endl;
15     cout << sizeof(c) << endl;
16
17     cout << sizeof(xp) << endl;
18     cout << sizeof(yp) << endl;
19     cout << sizeof(zp) << endl;
20     cout << sizeof(cp) << endl;
21 }
22
```

```
cout<< sizeof(*xp)<<endl;
cout<< sizeof(*yp)<<endl;
cout<< sizeof(*zp)<<endl;
cout<< sizeof(*cp)<<endl;
```

# Pointer Arithmetic

| Operation | Result |
|---|---|
| Address + number | Address |
| Address - number | Address |
| Address - Address | Number |
| Address + Address | Illegal |

# Example

```
#include <iostream>
using namespace std;
int main(void) {
   int *p1, *p2;

   p1= new int;
   *p1=100;
   p2=new int;
   *p2=120;

   cout << *p2-*p1 << endl;
   cout << p2-p1 << endl;


}
```

Output:
20
4

# Example

```
int *xp=&p;
double *yp=&d;
class {
    int temp[1000];
} c1, *cp=&c1;

cout<< xp<<endl;
cout<< yp<<endl;
cout<< cp<<endl;
```

```
0x7fff5fbff598
0x7fff5fbff5a8
0x7fff5fbfe5d8
```

```
cout<< xp+1<<endl;
cout<< yp+1<<endl;
cout<< cp+1<<endl;
```

```
0x7fff5fbff59c
0x7fff5fbff5b0
0x7fff5fbff578
```

# Remember this example?

```
int *ptr;
int a[10];
int i;

for(i = 0; i<10; i++){
  a[i] = i*2;
 }

ptr = a;

for(i = 0; i<10; i++){
  cout << ptr[i] << " ";
 }
cout << endl;

ptr[5] = 5;

for(i = 0; i<10; i++){
  cout << a[i] << " ";
 }
cout << endl;
```

0 2 4 6 8 10 12 14 16 18

Iterating through ptr is the same as iterating through array a.

0 2 4 6 8 5 12 14 16 18

# Pointers and Arrays

- We said that we can define a pointer and assign the (address of the) array to it, and work with it just like we work with an array
- What happens here:

- int b[10];
- b=ptr;
- cout<<b[0];

You cannot change the pointer value in an array variable.

# Example

```
1 main()
2 {int a[3]={2012,2,14};
3 int* p1=&a[0];
4 int* p2=a;
5 }
```

a=?
a+1 = ?
*(a+2) = ?

*(a+2) = 3; What happened?

*(a+3) = ?

*(a+3) = 0xbffff37c;
What happened?

| Addr | Name | Value |
|---|---|---|
| 0xbffff374 | a[0] | 2012 |
| 0xbffff375 | | |
| 0xbffff376 | | |
| 0xbffff377 | | |
| 0xbffff378 | a[1] | 2 |
| 0xbffff379 | | |
| 0xbffff37a | | |
| 0xbffff37b | | |
| 0xbffff37c | a[2] | 14 |
| 0xbffff37d | | |
| 0xbffff37e | | |
| 0xbffff37f | | |
| 0xbffff380 | p1 | |
| 0xbffff381 | | |
| 0xbffff382 | | |
| 0xbffff383 | | |
| 0xbffff384 | p2 | |
| 0xbffff385 | | |
| 0xbffff386 | | |
| 0xbffff387 | | |

# Example

```
1 main()
2 {char s1[]="abc";
3 char* s2="def";
4 char* p1=s1;
5 }
```

s1 is a constant address, and its value is not stored in the memory.

Address slots 0x8048484-7 are read-only.

What will happen if …
s1[0] = ‘z’;
s2 += 1;
p1 += 1;

| Addr | Name | Value |
|---|---|---|
| 0x8048484 | *s2 | 'd' |
| 0x8048485 | *(s2+1) | 'e' |
| 0x8048486 | *(s2+2) | 'f' |
| 0x8048487 | *(s2+3) | '\0' |
| ⋮ | | |
| 0xbffff38c | s1[0] | 'a' |
| 0xbffff38d | s1[1] | 'b' |
| 0xbffff38e | s1[2] | 'c' |
| 0xbffff38f | s1[3] | '\0' |
| 0xbffff390 | s2 | |
| 0xbffff391 | | |
| 0xbffff392 | | |
| 0xbffff393 | | |
| 0xbffff394 | p1 | |
| 0xbffff395 | | |
| 0xbffff396 | | |
| 0xbffff397 | | |

# Worksheet

- Predict the outcome of the code in the lecture 1-3 worksheet (under this weeks module in myUni).
- Do the online quiz for the lecture 1-3.
- After you have predicted the result, run the code to check your understanding.

# Summary

- Pointers allow you to access storage but it's extremely manual. Misjudging your pointer arithmetic will have strange results.
- Space is finite - management is important.
- C++ arrays and pointers are very easy to cause problems!


# Questions?