CRICOS PROVIDER 00123M

School of Computer Science

COMP SCI 2103/7103 Algorithm Design & Data Structure

Review of Pointers

adelaide.edu.au

seek LIGHT

Course Outline

- Week 1-6: Review, ADTs, Inheritance, Design, Recursion, Polymorphism, Complexity.
- Week 7-12: Sorting and Searching, Tree and Graph, Algorithmic strategies.

Assumed Knowledge

You can design and build simple object-oriented programs in C++, including classes and memory management.

Review of Pointers

In this lecture, we are going to:

- Review the concepts of pointers
- Talk about pointer arithmetic
- Discuss arrays and pointers

Review of Pointers

- What are pointer variables?
  - A pointer variable is a variable pointing to the memory address of another variable. It gives us more control on the computer's memory.
- How do we create them?
  - You can create a pointer variable by using the *

```
double *ptr1, ptr2;

double* ptr1, ptr2;

double *ptr1, *ptr2;
```

```
typedef double* DoublePtr;

DoublePtr ptr1, ptr2;
```

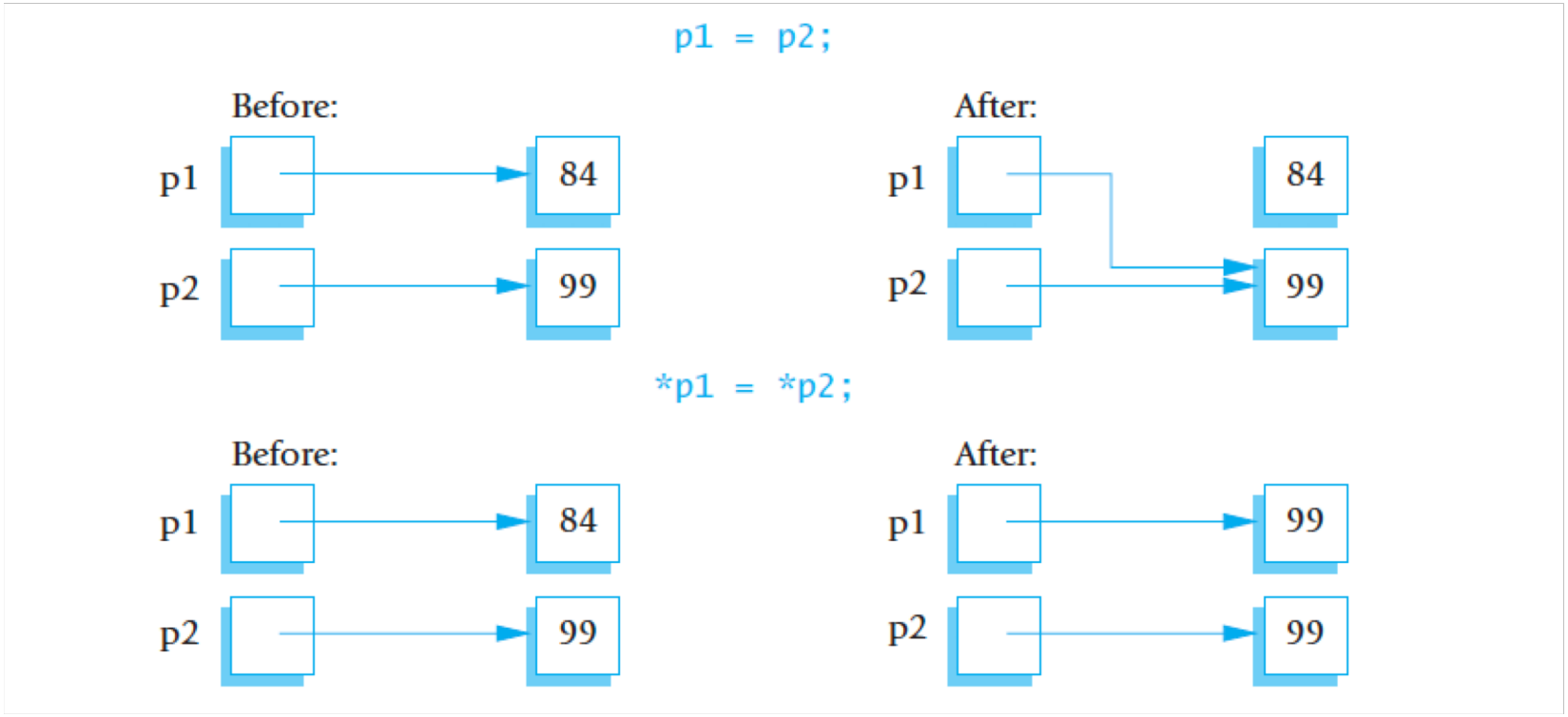
- How do we use them in C++?
  - You can refer to the address of a variable using &

```
double *p, v;
p = &v;
*p = 100;
```

v = ?

Dereferencing Operator

- The variable pointed to by *p* is accessed by **p*.
- What's the difference between p1 = p2 and *p1 = *p2?



Example 2

```
1 main()
2 {char c='c';
3 char* cp=&c;
4 int i=100;
5 int* ip=&i;
6 int j=*ip;
7 }
```

Questions:

1. *ip = 99;
2. *cp = *cp + 1;
3. &ip = ?

Addr	Name	Value
0xbffff374	i	100
0xbffff375		
0xbffff376		
0xbffff377		
0xbffff378	?	?
0xbffff379		
0xbffff37a		
0xbffff37b	c	'c'
0xbffff37c	cp	0xbffff37b
0xbffff37d		
0xbffff37e		
0xbffff37f		
0xbffff380	ip	0xbffff374
0xbffff381		
0xbffff382		
0xbffff383		
0xbffff384	j	100
0xbffff385		
0xbffff386		
0xbffff387		

Why Pointers/references?

- Instead of passing large quantities of data between functions, we can
  - just pass the pointer to the start of the data
  - saves memory and data transfer

- Dynamic memory usage

- Pass parameters to a function by reference

Pass-by-reference

```
void swapByValue(int x, int y){
  int temp;
  temp = x;
  x = y;
  y = temp;
}



void swapByPointer(int* x, int* y){
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}
```

Dynamic Variables

- What are dynamic variables?
  - Variables that are created by using *new* operator.
  - They are created and destroyed while the program is running.

```
1  //Program to demonstrate pointers and dynamic variables.
2  #include <iostream>
3  using namespace std;
4
5  int main( )
6  {
7      int *p1, *p2;
8
9      p1 = new int;
10     *p1 = 42;
11     p2 = p1;
12     cout << "*p1 == " << *p1 << endl;
13     cout << "*p2 == " << *p2 << endl;

14     *p2 = 53;
15     cout << "*p1 == " << *p1 << endl;
16     cout << "*p2 == " << *p2 << endl;

17     p1 = new int;
18     *p1 = 88;
19     cout << "*p1 == " << *p1 << endl;
20     cout << "*p2 == " << *p2 << endl;
21     cout << "Hope you got the point of this example!\n";
22     return 0;
23 }
```

```
*p1 == 42
*p2 == 42

*p1 == 53
*p2 == 53

*p1 == 88
*p2 == 53
```

Dynamic Variables

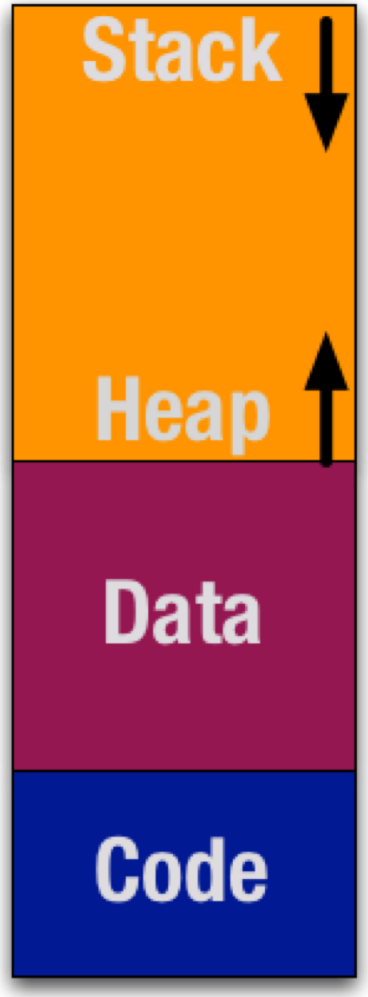
- "new" allocates a part of memory as the dynamic variable.
- We need to "delete" dynamic variables after use, to prevent memory leak

  - type * ptr= new type;
  - Set a value to *ptr and use it
  - When you no longer need it:
    delete ptr;

- The C++ standard specifies that if there is no sufficient memory available to create the new variable -> the new operator, by default, terminates the program.

Where are the variables stored?

- Static variables and parameters are stored in Stack.
- The rest of the allocated memory is used as the heap; for dynamic memory allocation
- The stack and heap are in a shared area.
- The memory that is allocated to variables and parameters from stack is not released until their function is returned. Unlike memory allocation in Heap.



Pointers vs Integers

- Pointers are variables of the same size as integers

- But are they exactly the same thing?

```
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;
```

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
```

Pointer Arithmetic

- You cannot perform the normal arithmetic operations on pointers.
  - multiplication or division is not allowed
  - Addition and subtraction are different
- The pointer arithmetic depends on the size of the type that pointer is of that type.
- Pointer arithmetic (+/-) shifts the address by a number of bytes equal to the size of the pointer type

```
int i=100;
int* ip=&i;
```

Addr	Name	Value
0xbffff380	ip	0xbffff374
0xbffff381		
0xbffff382		
0xbffff383		

What is the value of ip+1?
0xbffff378

Pointer Arithmetic

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

Example

```
int *ptr1, *ptr2;

ptr1 = 100;
ptr2 = 108;

cout << ptr2-ptr1 << endl;
```

Output: 2

What about ‘cout << *ptr2 - *ptr1 << endl;’ ?

Pointers and Arrays

- In C++, an array variable is actually a pointer variable that points to the first indexed variable of the array.

```
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;
```

By the way! You can go beyond the size of the array

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.

C-string

- A pointer-based string in C++ is an array of characters ending with the null terminator (‘\0’).
- The null terminator indicates where a string terminates in memory.
- A C-string can be accessed via a pointer

```
char city[9] = "Adelaide";
char *ptrCity = "Adelaide";
```

char city[] = {‘A’, ’d’, ’e’, ‘l’, ‘a’, ‘i’, ‘d’, ‘e’};

not equivalent!

```
cout << city[1] << endl;
cout << *(city+1) << endl;
cout << ptrCity[1] << endl;
cout << *(ptrCity+1) << endl;
```

All output the second element of the string.

More examples on pointers and arrays

```
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	0xbffff374
0xbffff381		
0xbffff382		
0xbffff383		
0xbffff384	p2	0xbffff374
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 0xbffff391 0xbffff392 0xbffff393	s2	0x8048484
0xbffff394 0xbffff395 0xbffff396 0xbffff397	p1	0xbffff38c

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?