Introduction to Programming (CS 101)

Spring 2024

Lecture 16:

Pointers continued (arrays as pointers, pointers in struct, dynamic allocation)

Instructor: Preethi Jyothi

Based on material developed by Prof. Abhiram Ranade and Prof. Manoj Prabhakaran

Recap (I): Pointer Operations

What is the output of the last `cout` statement?

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

int main() {
  int i = 1, j = 2;
  int* p = &j, * q = &i;
  *p = *q;
  cout << i << " " << j << endl;
}
```

1 1

1 2

2 1

2 2

Note the syntax for defining two pointers (p, q) in the same statement.

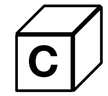
Recap (I): Pointer Operations

What is the output of the last `cout` statement?

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

int main() {
  int i = 1, j = 2;
  int* p = &j, * q = &i;
  *p = *q;
  cout << i << " " << j << endl;
  p = q; i += 2; j += 2;
  cout << i << " " << j << endl;
}
```

1 1

3 2



3 3

4 4

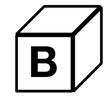
`p = q;` means both pointers point to the same variable `i`. Both `i` and `j` altered thereafter will reflect the modified values.

Recap (I): Pointer Operations

What is the output of the last `cout` statement?

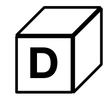
```
#include <iostream>
using namespace std;

int main() {
  int i = 1, j = 2;
  int* p = &j, * q = &i;
  *p = *q;
  cout << i << " " << j << endl;
  p = q; i += 2; j += 1;
  cout << i << " " << j << endl;
  j = *(*(&p));
  cout << i << " " << j << endl;
}
```

1 1



3 2

3 3



4 4

`*(&p)` is the address in the pointer `p`; dereferencing it would yield the value in `i`

Recap (II): Pointer Initialization

What is the output of the following program?

```
int main() {
  int* p;
  int* q = nullptr;
  int i = 2;
  if((!q && (p = &i)), (*p) += 2)
    cout << i << endl;
}
```

2

UB
(undefined behaviour)

Compiler error

4

`q` initialized as `nullptr`, will evaluate to false, when evaluated as an expression

Pointers in `struct`

CS 101, 2025

Pointers to structures and the –> operator

Consider the following structures:

```
struct Point {
  double x; double y;
};
struct Circle {
  Point center;
  double radius;
};

Circle c1={{1,2},3}, *circ;
circ = &c1;
(*circ).radius = 5;
```

C++ provides the operator –> to use as shorthand. If x is a struct object and y is a member variable then:

`x -> y is the same as (*x).y`

```
Circle c1={{1,2},3}, *circ;
circ = &c1;
circ->radius = 5;
```

Pointers inside structures (I)

Pointers can be members of a `struct`

```
struct Point {
  double x; double y;
};
struct Circle2 {
  double radius;
  Point* cptr;
};

Point p1 = {10.0, 20.0};
Circle2 cc1, cc2;
cc1.radius = cc2.radius = 5;
cc1.cptr = cc2.cptr = &p1;
```

How do we access the `x` coordinate of the centre of both circles `cc1` and `cc2`?

```
cout << cc1.cptr->x; //prints 10
```

If we change `cc2.cptr->y`, this would change the `y` coordinate of `p1` and hence the centre of both circles

Pointers inside structures (II)

Using a pointer to a struct inside its own definition:

```
struct Student {
  int rollno;
  Student* TA;
};

int main() {
  Student s1, s2, s3;
  s1.rollno = 1; s2.rollno = 2;
  s3.rollno = 3;
  s1.TA = &s2;
  s2.TA = &s3; s3.TA = &s1;
  cout << ((s1.TA)->TA)->rollno; //prints 3
}
```

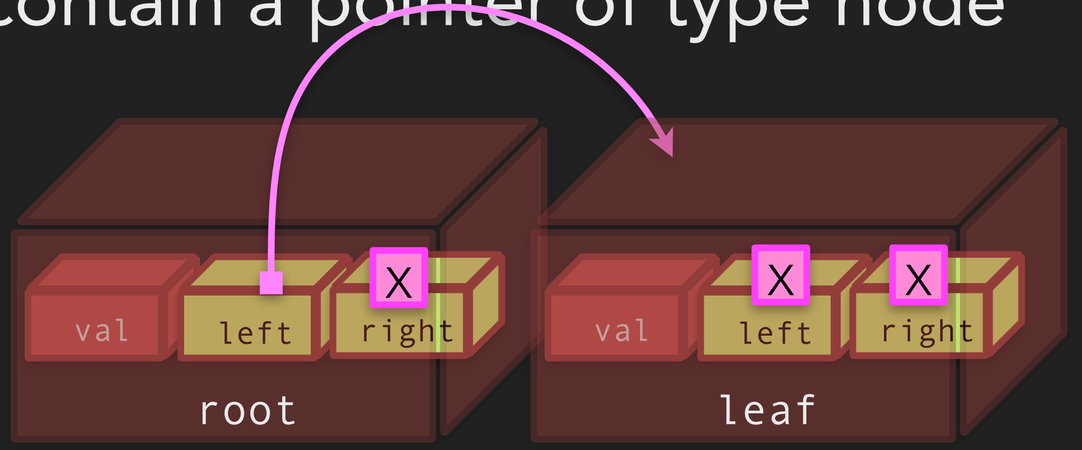
Pointers in Structs

- Consider a struct to hold the information about a node in a binary tree

```
struct leafNode { int value; };
struct parentNode { int value; leafNode leftChild; leafNode rightChild};
struct grandParentNode { int value; parentNode leftChild; parentNode rightChild};
```

- Will need a different struct for nodes in each level of the tree!
- Problem: A node cannot contain itself
- Alternative, use pointers; a node can contain a pointer of type node

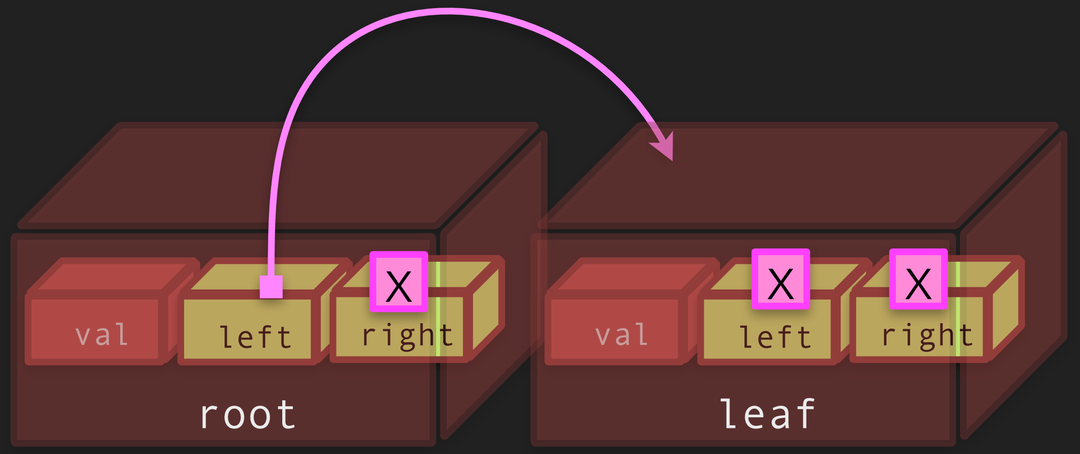
```
struct node {
  int value;
  node* leftChild;  // nullptr if no child
  node* rightChild; // nullptr if no child
};
...
node leaf = { 1, nullptr, nullptr };
node root = { 2, &leaf, nullptr };
...
```



Recursively print a binary tree

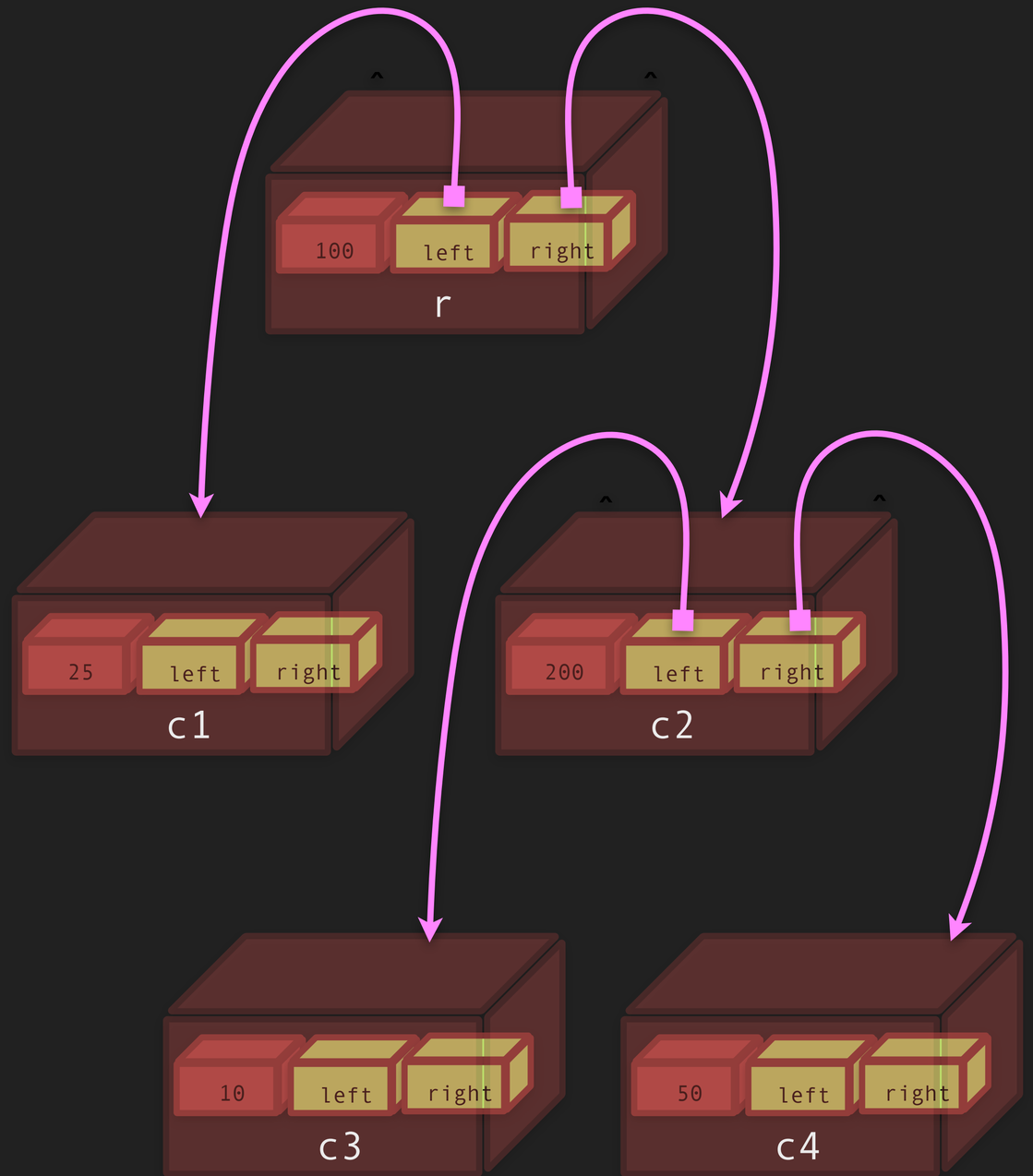
- Recall member X in a struct pointed to by p can be accessed as p->X

```
struct node {
  int val;
  node* left;  // nullptr if no child
  node* right; // nullptr if no child
};
void printTree(node* root) {
  if (!root)  // equivalently, if(root == nulptr)
    return;
  cout << " (" << root->val;
  printTree(root->left);
  printTree(root->right);
  cout << ") ";
}
```



Recursively print a binary tree

```
struct node {
  int val;
  node* left;  // nullptr if no child
  node* right; // nullptr if no child
};
void printTree(node* root) {
  if (!root)  // equivalently, if(root == nulptr)
    return;
  cout << " (" << root->val;
  printTree(root->left);
  printTree(root->right);
  cout << ") ";
}
int main() {
  node c3 = {10, nullptr, nullptr};
  node c4 = {50, nullptr, nullptr};
  node c1 = {25, nullptr, nullptr};
  node c2 = {200, &c3, &c4}, r = {100, &c1, &c2};
  printTree(&r);
}
```



Arrays as Pointers

CS 101, 2025

Arrays as Pointers

- Recall that C-style arrays can be passed to functions as arguments (and are always passed by reference)
  - Then the array is implicitly converted to a pointer to the first element of the array

```
void f(int A[]);
```

same as

```
void f(int* A);
```

- Given any pointer, can use array-like indexing
  - Works correctly if the pointer is indeed pointing to the first element of an array

```
int* p;
p = A;
p[2] = 1;
```

Arrays as pointers

```
int main() {
  float scores[] = {12.5,25.0,28.75,18.3};
  float* s = scores;
  cout << s[2]; //prints 28.75
}
```

- The assignment `s = scores` is valid, since the array `scores` is implicitly a pointer (of type `float*`) to the first element of the array
- `s[2]` means the same as `scores[2]`
- `s[2]` is the float value stored at the address `s + k*i` where `k` is the size of `float` (in bytes) and `i` is the index value (i.e., `k = 4` and `i = 2`)

Pointer Arithmetic

- Adding/subtracting integers, incrementing/decrementing, and comparisons are allowed on pointers, by interpreting them as pointing to elements in an array
  - `p+i` is the same as `& p[i]`
    - And, `*(p+i)` same as `p[i]`
  - `p<q` if `p == A+i` and `q == A+j` where `i<j`

```
void printArray(int* p, int* q) { while(p<q) cout << *p++ << " "; }
```

```
int A[10];
...
printArray(A,A+10);
```

Copying a string from a source array to a destination array

Recall:

```
void scopy(char from[], char to[]) {
  int i;
  for(i = 0; from[i] != '\0'; i++) to[i] = from[i];
  to[i] = from[i]; //copy the '\0'
}
```

Using pointers:

```
void scopy(char* from, char* to) {
  while(*from != '\0') {
    *to = *from; //dereferencing to, from
    to++; from++; //increment both pointers to the next array elements
  }
  *to = *from;
}
```

Convert a char array to an integer

Fill in the blanks below to convert a char array containing a non-negative number to an integer of the same magnitude.

```
int convert2int(char* c) {
  int result = 0;
  while(*c != '\0') {
    result = (result * 10) + (*c - '0');
    c++;
  }
  return result;
}
int main() {
⋮
char str[] = "3396";
cout << convert2int(str) + 4 << endl; //prints 3400
}
```

Sum of array slice

Fill in the blanks to complete the C++ function below that acts on a *slice* of an integer array (i.e., sequence of consecutive elements) and computes the sum of the slice (begin element included, end element excluded)

```
int slicesum(int* begin, int* end) {
  int sum = 0;
  for(int* p = begin; p != end; p++) {
    sum += (*p);
  }
  return sum;
}

int main() {
  int arr[] = {1,-4,55,7,8,2,9};
  cout << slicesum(arr+2, arr+4) << endl; //prints 62
}
```