Pointers, Reference & l/r values

04/01/2021

Pointers

- Point to the memory address of an object/variable
- int j = 4; //has some memory address (0x7fffffffd9d0)

  int* ptr = &j // * is telling us that ptr is an int pointer (and not an int)
- & is the “address-of” operator – meaning it returns the memory address (0x7fffffffd9d0) of j

  cout << ptr << endl; //prints “0x7fffffffd9d0”

  cout << *ptr << endl; //prints 4
- int *pptr; //can declare a pointer without defining it (or assign it to nullptr)
- pptr = &ptr //stores memory address of ptr (0x4fffffffaa10) to pptr
- cout << pptr << endl; //prints 0x4fffffffaa10
- cout << *pptr << endl; //prints “0x7fffffffd9d0”: the * here means “thing stored at”
- cout << **pptr <<endl; //prints 4

References

- Similar to pointers but are aliases for a variable

  int j = 4;

  int &jref = j; //& on the left side of assignment (=) means that jref is an alias for j, so anything you do to jref is the same as if you used j instead
- jref and j share the same memory address

  jref++;
- cout << j << endl; //prints 5
- int &jjref; //this is illegal: references must be instantiated when declared
- Int &&jjref = jref; //also illegal

Usage in function calls

void square_int(int a) {a = a*a;}

int j = 4; square_int(j); cout << j <<endl; //prints 4 because we just passed the value 4 into square_int

- By reference

  void square_int(int &a) {a = a*a;}; //passed by reference

  int j = 4; square_int(j); cout << j << endl; //we get 16

- By pointers

  void square_int(int * a) {*a = (*a)*(*a);} //function now takes int pointers as inputs

  int j = 4; int* jptr = &j; square_int(jptr); cout << j << endl; //we get 16

- Can also call square_int(&j); and skip creation of jptr

Pointers vs references

Pointer	Reference
Can declare without defining (int* ptr;)	Must define on declaration (int &intref; bad)
Can be reassigned ptr = &a; later, ptr = &b;	Cannot be reassigned
The pointer is stored in a new memory address	Reference has the same memory address as the thing it's referencing (aka it is just an alias)
Can have pointers of pointers (int *ptr = &a; int **ptrptr = &ptr)	Cannot have reference of reference (but int &p = a; int &q = a; is ok)
Can increment pointers to access the next memory address (ie for arrays) – part of pointer arithmetic	No such thing as reference arithmetic
Useful for different data structures (linked lists, trees)	Useful for passing things by reference into functions and managing return types

L and R values

int j = 4;

- j: l-value (left of the =, has a memory address)
- 4: r-value (right of the =, just a temporary value)

we can’t do

- 4 = j; (“error: lvalue required as left operand of assignment”)
- Similarly: int &jref = 4; (not allowed! & for references requires an l-value on the other side (4 has no memory address/is an r-value))
- const int &jref = 4; //apparently this is ok, but dont do this
- cout << jref << endl; //prints 4
- Also: int *ptr = &4; (also not allowed for same reason, 4 is an r-value, and & (address-of) operator needs an l-value)

L and R values (cont.)

- int setValue() { return 6;} //returns an r-value

  setValue() = 3; // error! Left side is an r-value
- but
- int fun = 20;

  int& setFun() { return fun;} //returns an l-value

  setFun() = 40; //ok, because setFun() returns an l-value
- fun = 40;
- ^^(the & specifies that setFun() returns an int reference)