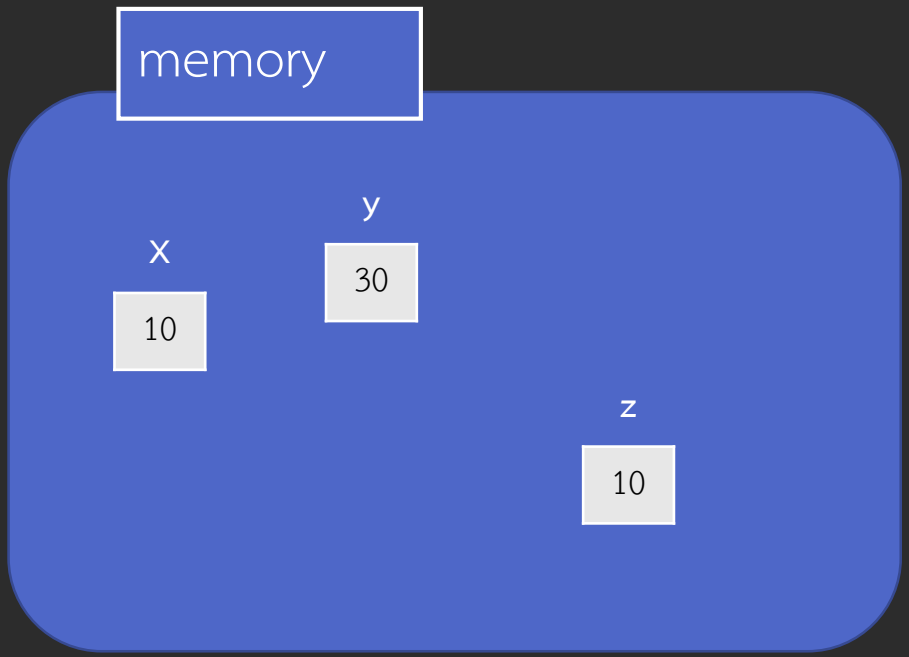
Type of Variable

Basic, Pointer, Reference

Variable

- Variable is a storage location
- We declare by giving type and name

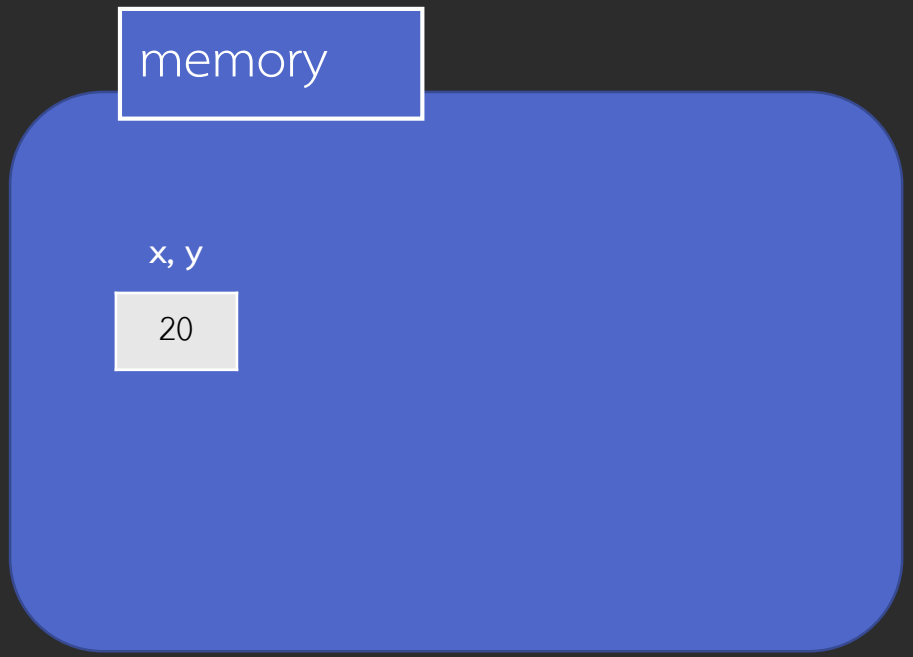
```
3   int x;
4   int y;
5   int z;
6   x = 10;
7   y = 30;
8   z = x;
```



Reference Variable

- Not an actual variable
  - But another name of other variable
- Cannot declare without initialize
- Can be used as output in function argument
- Declared with prefix **&** in front of the variable name

```
int x;         // normal variable
int &y = x // define a second name for x
x = 10      // set x to 10
y = 20      // set y (which is actually x) to 20
```



Reference Variable as “output” in function argument

- When a function argument is declared as a reference, calling that function must use a variable of the same type as an argument

```
1  void square(int &x) {
2    x = x * x
3  }
4
5  void half(int x) {
6    x = x / 2
7  }
8
9  int main() {
10   int y = 20;
11   square(y);
12   cout << y << endl;  // we will get 400
13   half(y);
14   cout << y << endl;  // we still have 400
15
16   square(30);   // fail to compile
17 }
18
```

Pointer

- Variable that store the address of another variable
- Declared with a prefix *
- Pointer is also typed
  - Pointer to an int is not the same as pointer to a string

```
4   int main() {
5
6       int x;        // x is int
7       int *p;       // p is a pointer to an int
8
9       x = 20;
10
11      // followings line not working
12      p = 30;
13      p = x;
14  }
```

Address operator: getting the address

- Using & in front of a variable (or function) results in its address

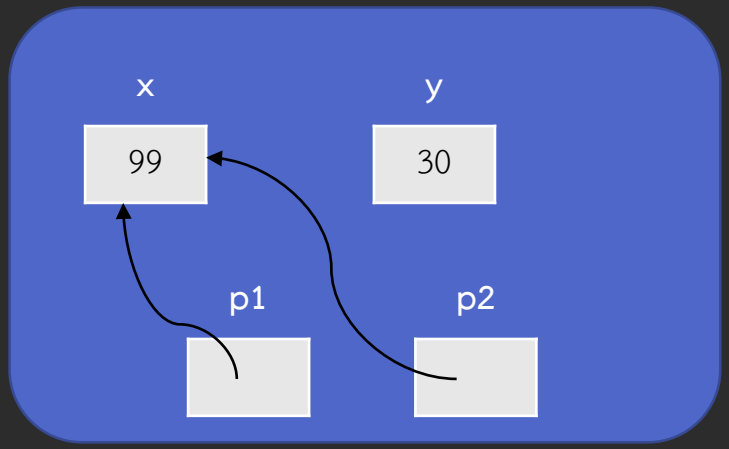
- Pointer variable must store address of a variable of the same type

```
6   void test(int x) {
7   }
8
9   int a,b;
10
11  int main() {
12      int x = 10;
13      int y = 20;
14      int z;
15
16      cout << x << endl;
17      cout << &x << endl;
18      cout << &y << endl;
19      cout << &z << endl;
20      cout << &a << endl;
21      cout << &b << endl;
22      cout << (long)&test << endl;
23  }
```

Dereferencing a pointer

- *x is the variable that x is pointing to
  - Can be used as both value and variable
- *x fails to be compiled if x is not a pointer

```
4   int main() {
5
6       int x,y;        // x, y are ints
7       int *p1,*p2;    // p1, p2 are int pointers
8
9       x = 20;
10      y = 30;
11
12      p1 = &x;
13      p2 = p1;
14
15      cout << "P1: " << p1 << endl;
16      cout << "P1: " << p2 << endl;
17      cout << "*P1: " << *p1 << endl;
18      cout << "*P2: " << *p2 << endl;
19      cout << "&P1: " << &p1 << endl;
20      cout << "&P2: " << &p2 << endl;
21
22      *p2 = 40;
23      cout << x << endl;
24      x = 99;
25      cout << *p1 << endl;
26  }
```



New operator

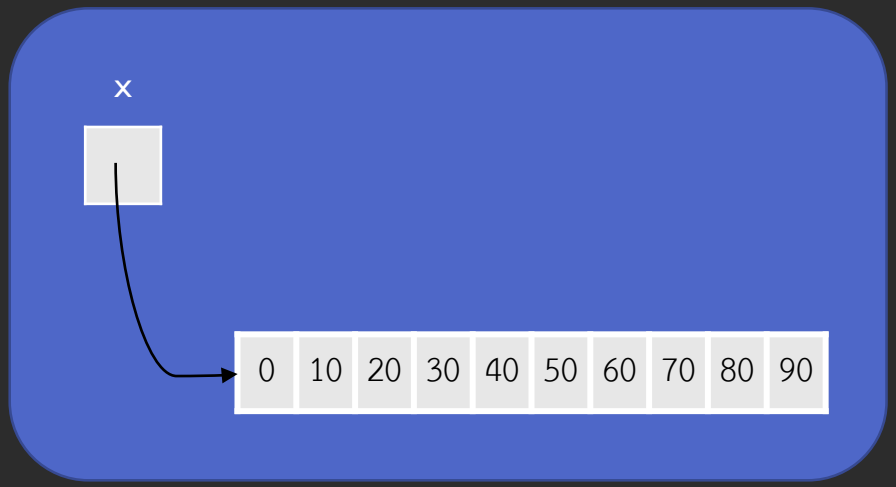
- `new` operator allocate a memory and return its address
- The allocated memory must be deleted by `delete` operator

```
3   int main() {
4     int *p,*q;
5     int x,y;
6
7     p = new int;
8     q = new int(20);
9
10    x = *q;
11    *p = 30;
12    *q = *p;
13
14    cout << *p << endl;
15    cout << *q << endl;
16    cout << x << endl;
17
18    delete p;
19    delete q;
20  }
```

Dynamic Array

- new operator can be used to allocate a block of memory for the same type of variable

```
2  int main() {
3      int *x;
4      x = new int[10];
5
6      for (int i = 0;i < 10;i++) {
7          x[i] = i * 10;
8      }
9
10     cout << "address" << endl;
11     cout << x << endl;  //address
12     cout << &x[0] << endl;
13     cout << &x[1] << endl;
14     cout << &x[2] << endl;
15     cout << (x+4) << endl;
16
17     cout << "value" << endl;
18     cout << *x << endl;
19     cout << x[1] << endl;
20     cout << x[2] << endl;
21     cout << *(x+4) << endl;
22 }
```



Summary

Modifier	Declaration prefix	As an operator
(none)	Normal variable	N/A
*	Pointer variable	De-reference
&	Reference variable	Addressing