NWEN 241
Systems Programming

Sue Chard

suechard@ecs.vuw.ac.nz

Content

- New eBook in the library.

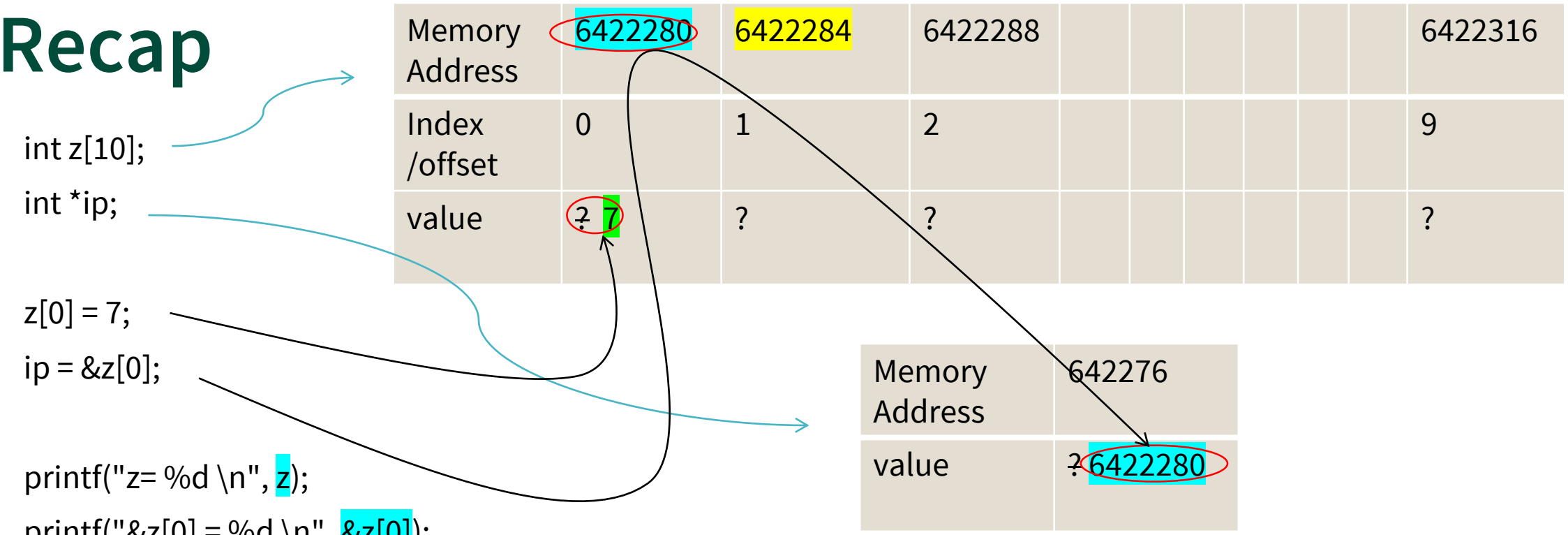
  S. Malik C++ Programming Program Design including data structures 8th Edition 2018

- Recap and Start Slide 28 Pointers

- More on pointers in C

- Pointers in C++

Recap




```
int z[10];
int *ip;

z[0] = 7;
ip = &z[0];

printf("z= %d \n", z);
printf("&z[0] = %d \n", &z[0]);
printf("ip = %d \n", ip);
printf("z[0] = %d \n", z[0]);      //value store in array element offset by zero
printf("*z = %d \n", *z);          // value stored when the pointer with the address of the first element in
                                   // the array is de-referenced (follow the pointer)
printf("*ip = %d\n", *ip);         // value stored when ip is dereferenced  (follow the pointer)
printf("ip++ = %d\n", (ip+1));     // address when the size of 1 array element is added to the address stored in ip
```

Reference / variable parameters

A function can return a pointer value

```
float *findMax(float A[], int N) {
  int I;
  float *theMax = &(A[0]);
  for (I = 1; I < N; I++)
    if (A[I] > *theMax) theMax = &(A[I]);
  return theMax;
}

void main() {
  float A[5] = {0.0, 3.0, 1.5, 2.0, 4.1};
  float *maxA;
  maxA = findMax(A,5);
  *maxA = *maxA + 1.0; //
  printf("maxA %.1f    A[4]%.1f\n",*maxA,A[4]);
}
```

The output is: maxA 5.1 A[4] 5.1

Function prototype, return float, parameters are an array of float and integer N, with number of elements in the array. Note the array will be a pointer to the start of the array.

Set the initial value of the pointer themax to the address of the first element in the array
Iterate checking each value, setting themax to the address of the higher values
Return the value currently stored in themax
follow the pointer and add 1 to the value
Dereference maxA and output it,
Use index of 4 to output the value stored in this element in the array

const and pointer parameters

The qualifier const can be added to the left of the parameter to declare that the code using the variable will not change the variable.

```
int foo(const int * a, const int * b){
     *a = 5;
     return *a + *b;
}
```

This will not compile …

const and pointer parameters

Const pointer: cannot be reassigned to point to a different location from the one it is initially assigned, but it can be used to modify the location that it points to

```
Datatype * const p;
```

Pointer to a const location: can be reassigned to point to a different location, but it cannot be used to modify any location that it points to

```
const Datatype * p;
```

Const pointer to a const location: can neither be reassigned nor used to modify the location that it points to

```
const Datatype * const p;
```

Pointer to pointers

A pointer can also be made to point to a pointer variable (but the pointer must be of a type that allows it to point to a pointer)

Example:

```
int V = 101;
int *P = &V;                    /* P points to int V */
int **Q = &P;                    /* Q points to int pointer P */

printf("%d %d %d\n", V, *P, **Q); /* prints 101 3 times */
```

Pointer Types and Casting

Pointers are generally of the same size (enough bytes to represent all possible memory addresses), but it is inappropriate to assign an address of one type of variable to a different type of pointer

Doing the following will generally result in a warning, not an error, because C will allow you to do this and it is appropriate in some situations

Example:

```
int V = 101;
float * P = &V; // Generally results in a Warning not an error
```

When assigning a memory address of a variable of one type to a pointer that points to another type, it is best to use the cast operator to indicate the cast is intentional (this will remove the warning), but it is still unsafe to do this !!!

Example:

```
int V = 101;
float * P = (float *) &V;   // Casts int address to float
```

Pointer operators

```
int * p, * q;
p =q            //assign q to p
p == q          //evaluates to true if both pointers point to the
                //same address
p!= q           //evaluates to true if  p and q point to different addresses
p++ or p = p + 1    // increments pointer by sizeof variable to point to
                    // the next element
P-- or p = p - 1    // decrements pointer by sizeof variable to point
                    // to the previous element
```

General (void) pointers

A **`void *`** is considered to be a general pointer

No cast is needed to assign an address to a **`void *`** or from a **`void *`** to another pointer type

Example:

```
int V = 101;
void * G = &V; //* No warning
float * P = G; //* No warning, still unsafe
```

Some library functions return **`void *`** results

C++

- Everything that C can do plus some
- Declaration of pointers * symbol

  `int * j  or  int* j  or  int *j`
- Address of operator &
- Defererencing / indirection operator * (follow the pointer)
- Initialising pointer to NULL

  `int * j = NULL  or  int * j = 0  or  int * j = nullptr` (only C++ 11 standard)

C++ dynamic variables

- C++ has dynamic variables, used to allocate memory space during a programs execution
- new and delete reserved words are used to create and destroy dynamic variables
- The new keyword allocates memory for the variable and returns a pointer to it. The following syntax allocates memory for a single variable and an array of variables.

    new *datatype*;

    new *datatype* [an expression that evaluates to an integer];

C++ dynamic memory leaks

- new and delete reserved words are used to create and destroy dynamic variables. They must be used in a pair to prevent memory leaks…
- The delete keyword is used to return / release the memory that was allocated using the new keyword. The following syntax deallocates memory for a single dynamic variable and a dynamic array variable

```
delete pointervariable;

delete [] pointervariable;
```

C++ dynamic variables examples

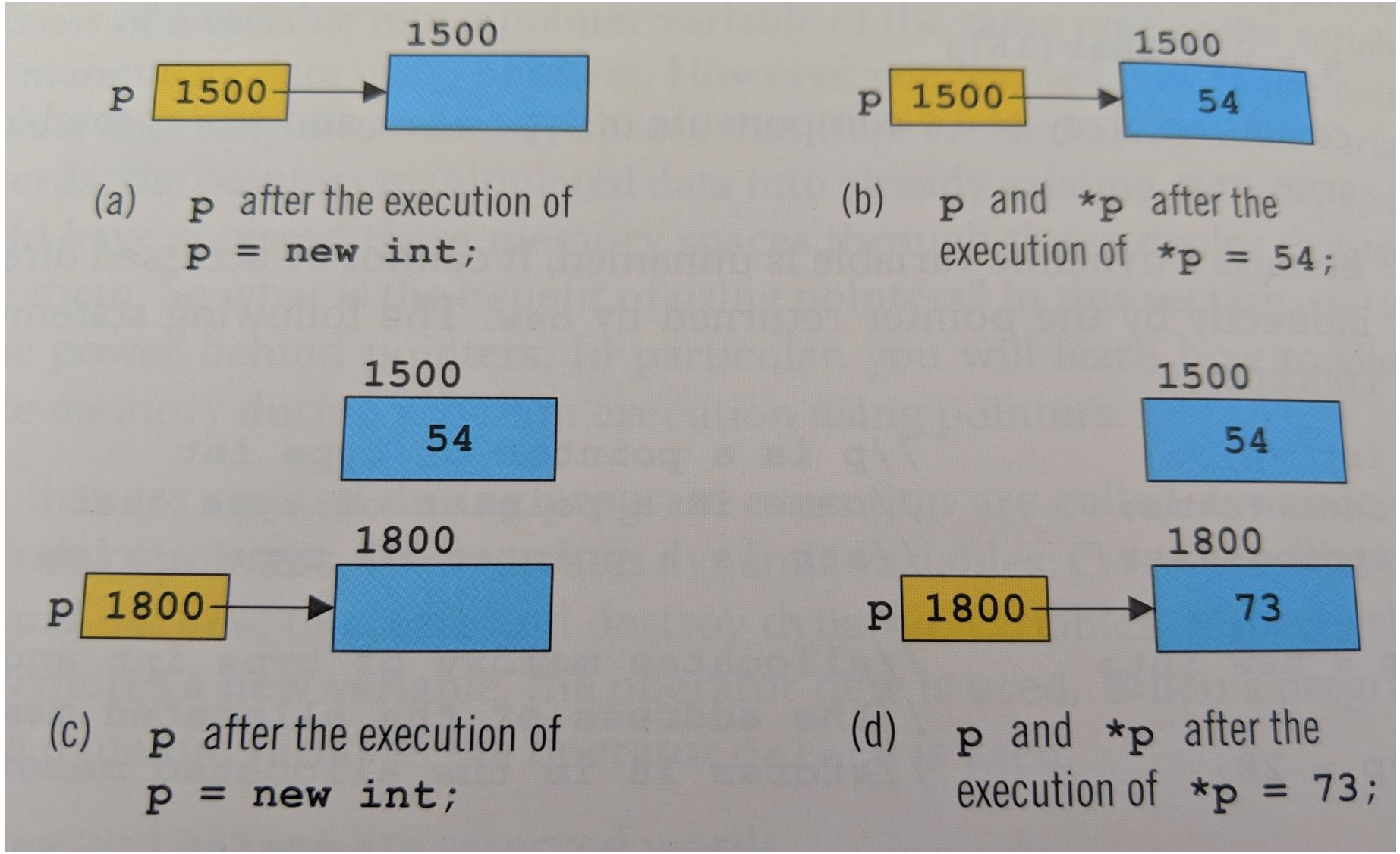
```
int * p;                    // p is a pointer to variable of type int
char * z;                   // z is a pointer to variable of type char
int x;                      // x is a variable of type int
p = &x;                     // p is assigned the address of x
p = new int;                                    // allocates memory of type int and
stores                                                  // the address of the allocated memory in p
*p = 28;                    // stores 28 in the allocated memory
z = new char[10];           // allocates memory for a char array of
10                                                      // elements and stores the memory address in z
strcpy(z,"tarakihi");       //copies tarakihi to z
delete p;                   //releases the memory allocated to store type int
delete [] z;                //releases the memory allocated to store char array
```

```
int * p;          //line 1

p = new int;  //line 2
*p = 54;          //line 3

p = new int;  //line 4
*p =73;           //line 5
```

Needs the code
delete p;
between line 3
and line 4
otherwise there
will be a memory
leak



A Useful quote to remember

*"Pointers have been lumped with the **goto** statement as a marvelous way to create impossible-to-understand programs. This is certainly true when they are used carelessly, and it is easy to create pointers that point somewhere unexpected. With discipline, however, pointers can also be used to achieve clarity and simplicity."*

– Kernighan and Ritchie, 1988.