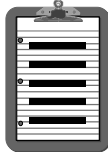


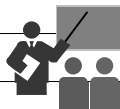
List Objects

- **So far, individual data objects**
 - Each object holds a single value
- **Sometimes we need more**
 - A group of objects
 - Stored together
 - Still accessible individually



Handwritten notes area with horizontal lines.

List Problem



- **Manage a set of grades**
 - Each grade is an integer value
 - Only four students in the class
- **Store grade for each student**
 - Examine grade
 - Modify grade

Handwritten notes area with horizontal lines.

Grade List Definition

```
// Declare global variables:
int grade_0; // First grade
int grade_1; // Second grade
int grade_2; // Third grade
int grade_3; // Fourth grade
```

In C++, we number (index) things starting with zero!

Handwritten notes area with horizontal lines.

Examining a Grade

```

int lookAt (int index)
{
  int g;
  switch (index)
  {
    case 0: g = grade_0; break;
    case 1: g = grade_1; break;
    case 2: g = grade_2; break;
    case 3: g = grade_3; break;
  }
  return g;      Return grade value
                  corresponding to index.
}

```

Changing a Grade

```

void change (int index, int gr)
{
  switch (index)
  {
    case 0: grade_0 = gr; break;
    case 1: grade_1 = gr; break;
    case 2: grade_2 = gr; break;
    case 3: grade_3 = gr; break;
  }
}
      Set new value for grade
      corresponding to index.

```

Adjusting All Grades

```

// Increase all grades by
// fifteen points:
for (int j = 0; j < 4; j++)
{
  change (j, lookAt(j) + 15);
}

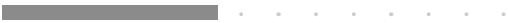
```

Change to new value Get current grade

⋮

List Advantage

- **Why go to this trouble?**
- **Can handle entire list at once**
 - Use loop to iterate through list elements
- **Can be extended to any size**
 - Though functions grow, too

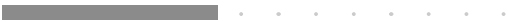


Handwritten notes on lined paper for the List Advantage section.

⋮

A Disadvantage

- **Two functions needed**
 - Examine: lookAt
 - Returns value
 - Modify: change
 - Modifies object
- **Can we combine them?**

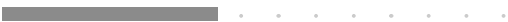


Handwritten notes on lined paper for the A Disadvantage section.

⋮

An Alternative

- **One list access function**
- **Returns reference to grade**
 - Not simply the value
- **Reference**
 - Similar to pass-by-reference (lvalue)
 - Reference is alias for actual object



Handwritten notes on lined paper for the An Alternative section.

Reference Implementation

```

Return reference to grade value.
int& grade (int index)
{
  switch (index)
  {
    case 0: return grade_0;
    case 1: return grade_1;
    case 2: return grade_2;
    case 3: return grade_3;
  }
}

```

Adjusting Grades - Reference

```

// Increase all grades by
// fifteen points:
for (int j = 0; j < 4; j++)
{
  grade (j) = grade (j) + 15;
}

```

Set new value Get current grade

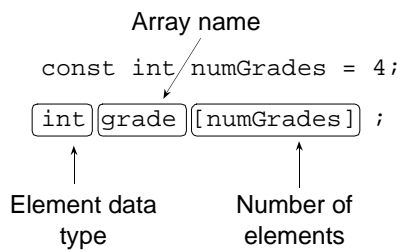
Summary So Far

- **List implementation works**
 - Reference version is simpler
- **But, serious limitations**
 - Only one list
 - Doesn't scale well to large lists
 - switch statements grow linearly

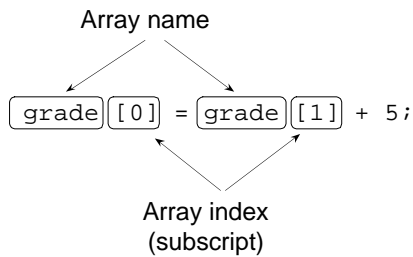
Array Objects

- **C++ aggregate type**
 - Collection of individual objects
- **Group managed together**
 - One name for collection
- **Individual items accessible**
 - Using an index value

Defining an Array



Accessing Array Elements



Adjusting Grades - Array

```

// Increase all grades by
// fifteen points:
for (int j = 0; j < 4; j++)
{
  grade [j] = grade [j] + 15;
}

```

Set new value

Get current grade

Subscript Range

- If N elements in an array
- Subscript range: 0 .. N-1
- Programmer must guarantee
 - Subscript in legal range
 - Not negative or greater than N-1
 - Array must be large enough

Defining Arrays

```

// Uninitialized arrays:
int grade [20];
char letters [26];
double rainfall [365];

// Initialize array values
int nums[3] = { 27, -45, 16 };

// Automatic array sizing:
double xa[] = { 1.7, 2.3, 4 };

```

Arrays as Function Arguments

- **Always** passed by reference
 - Automatically, no "&" needed
- **Formal argument**
 - Use [] to specify array
- **Actual argument**
 - Array name only

Array Argument Example

```

void main ()
{
  const int asize = 4;
  int xx[asize] = {1,5,2,7};
  cout << Sum (xx, asize);
}
int Sum (const int ar[], int n)
{
  int sum = 0;
  for (int j=0; j<n; j++)
    sum += ar[j];
  return sum;
}

```

← const means no modification
