8. Sequence Data II

11. Sequences

Static Arrays (1)

```
dataType arrayName[size];
dataType arrayName[size] = {value1, value2, … };
dataType arrayName[] = {value1, value2, … };

sizeof

arrayName[index] // [0, size-1]

returnType functionName(datatype arrayName[], const int size)
```

Static Arrays (2)

```
void print(int a[], int n) {
   for (int i = 0; i < n; i++)
      std::cout << a[i] << " ";
   std::cout << '\n';
}

int main() {
   int list[] = { 2, 4, 6, 8 };
   print(list, 4);     // sizeof(list)/sizeof(int)
   std::cout << sum(list, 4) << '\n';
   for (int i = 0; i < 4; i++)
      list[i] = 0;

   print(list, 4);
}
```


Static Arrays (3)

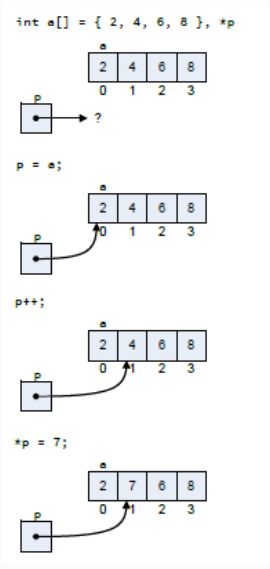
```
void print(int a[], int n) {
   for (int i = 0; i < n; i++)
      std::cout << a[i] << " ";
   std::cout << '\n';
}
void clear(int a[], int n)  // void clear(int* a, int n)
{
   for (int i = 0; i < n; i++) a[i] = 0;
}
int main() {
   int list[] = { 2, 4, 6, 8 };
   print(list, 4); // print(&list[0], 4);
   clear(list, 4);
   print(list, 4);
}
```

Pointers and Arrays (1)

```
#include <iostream>
int main() {
   int a[] = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 }, *p;
   p = &a[0]; // p points to first element of array a

   for (int i = 0; i < 10; i++) {
      std::cout << *p << ' ';
      p++;  // +1 ?
      // std::cout << *p++ << ' ';    // a[i], i=i+1
      // std::cout << (*p)++ << ' '; // a[0]++
   }

   std::cout << '\n';
}
```


Pointers and Arrays (2)

```
#include <iostream>
int main() {
   int a[] = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 },
      *begin, *end, *cursor;
   begin = a;
   end = a + 10;

   cursor = begin;
   while (cursor != end) {
      std::cout << *cursor << ' ';
      cursor++;
   }
   std::cout << '\n';
}
```



Pointers and Arrays (3)

```
#include <iostream>
void iterative_print(const int *a, int n) {
   for (int i = 0; i < n; i++)
      std::cout << a[i] << ' ';
}
void recursive_print(const int *a, int n) {
   if (n > 0) {
      std::cout << *a << ' ';
      recursive_print(a + 1, n - 1);
   }
}
int main() {
   int list[] = { 23, -3, 4, 215, 0, -3, 2, 23, 100, 88, -10 };
   iterative_print(list, 11);
   recursive_print(list, 11);
}
```


Pointers and Arrays (4)

```
void print(int *begin, int *end)
// end points just past the end of the array
{
   for (int *elem = begin; elem != end; elem++)
      std::cout << *elem << ' ';
   std::cout << '\n';
}
```

Pointers and Arrays (5)

```
#include <iostream>
int main() {
   int arr[] = {10, 20, 30, 40, 50};
   int *p = arr;
   std::cout << *p << '\n';   // 10
   std::cout << p[0] << '\n'; // 10
   std::cout << p[1] << '\n'; // 20
   std::cout << *p << '\n';   // 10
   p++; // Advances p to the next element
   std::cout << *p << '\n';   // 20
   p += 2; // Advance p two places
   std::cout << *p << '\n';   // 40
   std::cout << p[0] << '\n'; // 40
   std::cout << p[1] << '\n'; // 50
   p--;
   std::cout << *p << '\n';
}
```


Dynamic Arrays (1)

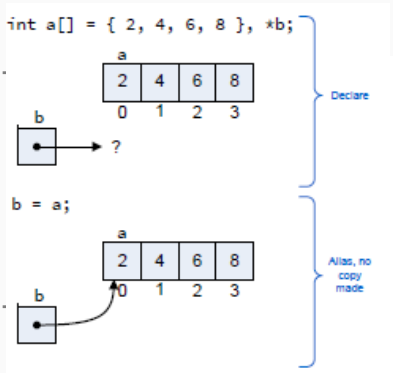
```
#include <iostream>
const int MAX_NUMBER_OF_ENTRIES = 1000000;
double numbers[MAX_NUMBER_OF_ENTRIES];
int main() {
   int size;
   std::cin >> size;
   if (size > 0) {
      for (int i = 0; i < size; i++)
         std::cin >> numbers[i];

      for (int i = 0; i < size; i++)
         std::cout << numbers[i] << '\n';
   }
}
```

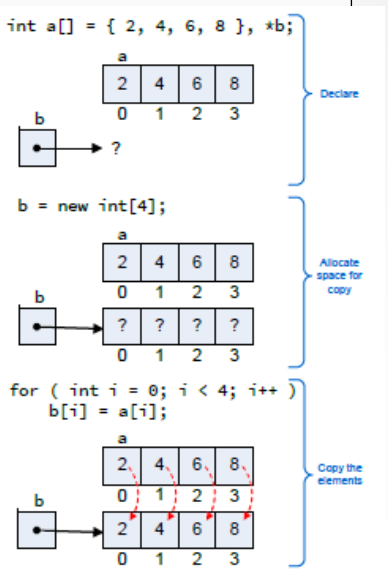
Dynamic Arrays (2)

```
#include <iostream>
int main() {
   double *numbers;
   int size;
   std::cin >> size;
   if (size > 0) {
      numbers = new double[size];     // numbers = new double;
      for (int i = 0; i < size; i++)
         std::cin >> numbers[i];

      for (int i = 0; i < size; i++)
         std::cout << numbers[i] << '\n';

      delete [] numbers;              // delete numbers;
   }
}

// local variables: stack, global(static) variables: data
// dynamic memory: heap, instruction: code
```


Copying an Array

```
int a[10], b[10];
for (int i = 0; i < 10; i++)
   a[i] = i;
b = a;
-----------------------------

int a[10], *b;
for (int i = 0; i < 10; i++)
   a[i] = i;
b = a;
-----------------------------

int a[10], *b;
b = new int[10];
for (int i = 0; i < 10; i++)
  b[i] = a[i];
```





Multidimensional Arrays (1)

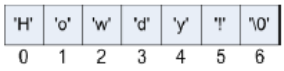
```
dataType array[size2][size1];
dataType array[size2][size1] = {{…}, {…}, …};
dataType array[][size1] = {{…}, {…}, …};
dataType array[size2][size1] = {…};

array[index2][index1]
array[index2] // ?

------------------------------------------------------------

int m[3][2] = {{1}, {21, 22}, {31, 32}};
```


Multidimensional Arrays (2)

```
#include <iostream>
#include <iomanip>
const int ROWS = 3, COLUMNS = 5;
using Matrix = double[ROWS][COLUMNS];
// typedef double Matrix[ROWS][COLUMNS];
// void print_matrix(const double m[ROWS][COLUMNS])
// const double m[][COLUMNS], const double (*m)[COLUMNS]
void print_matrix(const Matrix m){
   for (int row = 0; row < ROWS; row++) {
      for (int col = 0; col < COLUMNS; col++)
         std::cout << std::setw(5) << m[row][col];
      std::cout << '\n';
   }
}
int main() {
   double mat[ROWS][COLUMNS] = {{1, 2, 3, 4, 5}, {11, 12, 13, 14, 15},
      {21, 22, 23, 24, 25}}; // Matrix mat = {…};
   print_matrix(mat);
}
```

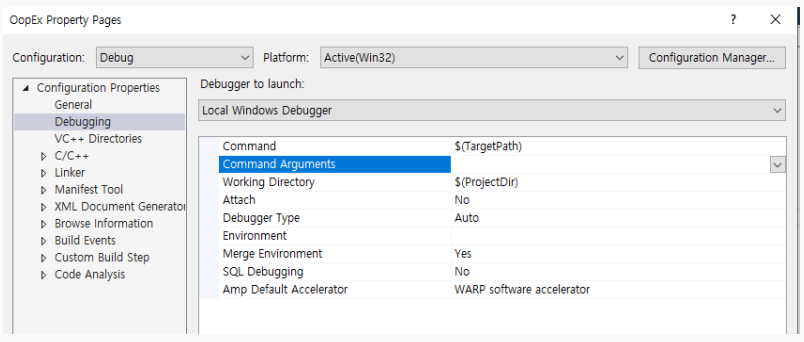
C String

```
char *word = "Howdy!";
std::cout << word << '\n';

char word[256];
std::cin >> word;

char word[10];
fgets(word, 10, stdin);        // #include <cstdio>
std::cout << word << '\n';

// delimiter, '\0'

// <cstring>
int strlen(const char *s);
char *strcpy(char *s, const char *t);
int strcmp(const char *s, const char *t);
```



Command-line Arguments

```
>> copy count.cpp count2.cpp

-------------------------------------------------------------------------

#include <iostream>
int main(int argc, char *argv[]) {
   for (int i = 0; i < argc; i++)
      std::cout << '[' << argv[i] << "]\n";
}
```



Vectors vs. Arrays (1)

```
#include <iostream>
#include <vector>
#include <array>
int main() {
   std::vector<int> v(10);
   std::cout << v[0] << std::endl;

   std::array<int, 10> a;
   std::cout << a[0] << std::endl;

   int arr[10];
   std::cout << arr[0] << std::endl;

   int x = int();
   std::cout << x << std::endl;
}
```


Vectors vs. Arrays (2)

```
std::vector<int> vec = {10, 20, 30};

std::cout << *vec.begin() << std::endl;
std::cout << *(vec.end()-1) << std::endl;

int *cursor = &vec[0];
int *end = &vec[0] + vec.size();
while (cursor != end) {
   std::cout << *cursor << ' ';
   cursor++;
}
```

Vectors vs. Arrays (3)

	Vector	Array
Memory	Occupy more memory than array	Memory-efficient
Length	Variable length	Fixed-size length
Usage	Frequent insertion and deletion	Frequent element access
Resize	Dynamic	Resizing arrays is expensive
Indexing	Non-index based	Zero-based indexing
Access	Time-consuming	Constant time