Advanced design of constructor

Various types of statements

```
Circle circ(10, 20, 5);
Circle circ = Circle(10, 20, 5);
Circle *circ = new Circle(10, 20, 5);
```

Omit the 3-rd parameters： rad is set by a default value

```
Circle circ(10, 20);
Circle circ = Circle(10, 20);
Circle *circ = new Circle(10, 20);
```

Converting constructor

- Constructor that takes only one parameter
- Two calling types exist (explicit or implicit)

If class C has a constructor of C::C (int x);

Explicit calling：

```
C obj(10);
```

Implicit calling：

```
C obj = 10;
```

Converting constructor for Circle

Constructor of Circle class can be a converting constructor by adding default parameters as

```
Circle (int cx, int cy = 0, int r = 10) {
        x = cx; y = cy; rad = r;
}
Explicit calling：
            Circle circ(10); // y = 0, rad = 10
Implicit calling：
            Circle circ = 10; // y = 0, rad = 10
```

Constructor for primitive variables

In C++, primitive variables, such as int, float, char, can be regarded as a class which has a constructor as

All statements below generates integer variables of 100

```
int n=100;
int n(100);
int *p = new int(100);

// Notice the difference against the array
definition such as int *p = new int [100];
```

Usage of constructors for primitives

```
class Circle {
private:
    int x, y;
    int rad;
public:
    Circle (int cx, int cy, int r = 10) {
        x = cx; y = cy; rad = r;
  }
```

```
Circle (int cx, int cy, int r = 10) : x(cx), y(cy), rad(r) { }
```

Array of class object

Review of array in C language

A hundred of struct Health is allocated as array data

```
struct Health data[100];
```

After allocating 100 of pointers to struct Health, assign their instances in run time

```
struct Health *data[100];
data[0] = (struct Health *) malloc(sizeof(struct Health));
data[1] = (struct Health *) malloc(sizeof(struct Health));
// continue…
```

Dynamically allocating size of pointers in run time

```
size = 50;
struct Health **data = (struct Health **) malloc(size * sizeof(struct Health*));
data[0] = (struct Health *) malloc(sizeof(struct Health));
// continue...
```

Array for class instance

Allocating 100 instances of Health by calling a constructor of no parameter

```
Health data[100];
```

After allocating 100 pointers to Health class, assign their instances in run time

```
Health *data[100];
data[0] = new Health (“taro”, 1.7, 60);
data[1] = new Health (“hanako”, 1.6, 50);
// continue…
```

Dynamically allocating size of pointers in run time

```
size = 50;
Health **data = new Health* [size];
data[0] = new Health(“taro”, 1.7, 60);
// continue…
```

Usage of class array

```
class HealthGroupManager {
private:
    Health *data[100]; // 100 Health data can be assigned at maximum
    int numStudents = 0; // number of assigned data
public:
    void setStudentData (char *n, float h, float w);
    float getAverageBMI (); // averaged BMI for all data
};
```

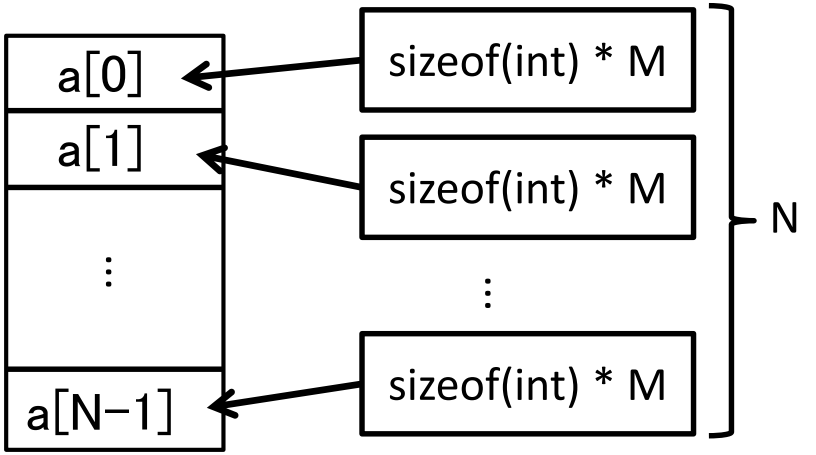
Example of member functions

```
// add (register) a single data to this class
void HealthGroupManager::setStudentData (char *n, float h, float w) {
    data[numStudents++]  = new Health (n, h, w);
}

// calculate the average for all registered students
float HealthGroupManager::getAverageBMI () {
    float sumBMI = 0.0;
    for (int i = 0; i < numStudents; i++)
        sumBMI += data[i]->getBMI();
    return sumBMI / (float) numStudents;
}
```

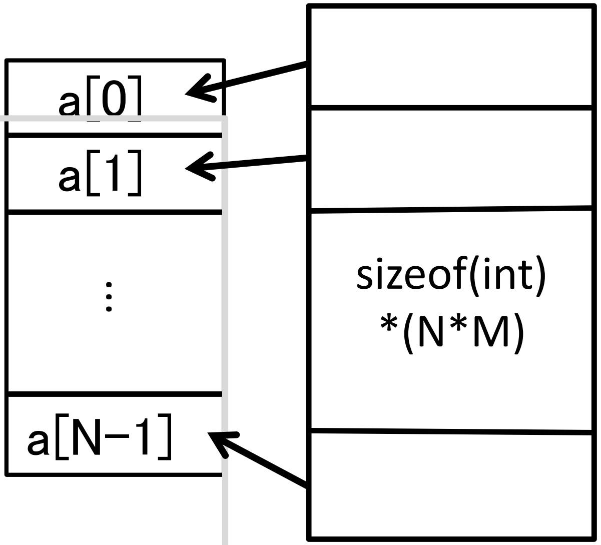
Allocation of 2D array

```
int **a = new int*[N];
for (int i = 0; i < N; i++)
     a[i] = new int[M];
```



For allocating continuum region of memory：

```
int **a = new int*[N];
// allocate a whole region
a[0] = new int[N * M];
for (int i = 1; i < N; i++)
     a[i] = a[0] + i * M;// assign each addresses
```



I/O functions in C++

I/O functions

- C＋＋ has own I/O functions
  - standard functions have the prefix of std::
- Stream is used for input/output of data
  - std::istream class for input
  - std::ostream class for output
  - These classes are defined in iostream.h
  - Standard i/o is implemented as std::cin or std::cout object
- >> and << represents the flow of input and output, respectively

Usage of stream I/O

```
int main () {
    int n;
    char str[100];
    std::cout << ″Input Integer value¥n″;
    std::cin >> n; // Input a integer with keyboard
    std::cout << ″Integer ″ << n << ″ is inputted¥n″;
    std::cout << ″Input string value¥n″;
    std::cin >> str; // Input a string with keyboard
    std::cout << ″String ″ << str << ″ is inputted¥n″;
}
```

Backslash is input by pushing “¥” on a keyboard,
or option-” ¥ ” depending on your system environment

Change of usage for stream

Old type → warning in compilation

#include <iostream.h>

New type → Declare standard libraries

#include <iostream> // omit ″.h″

rename as cin ⟶ std::cin , cout ⟶ std::cout

where the prefix of std:: can be omitted by declaring namespace of standard library as

using namespace std;

In this exercise, we recommend the usage of new type!