クラス構築子の
高度な設計

構築子の種々の記述方法

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

引数省略版：rad にはデフォルト値が代入される

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

変換コンストラクタ

- 引数をひとつだけとるコンストラクタ（構築子）は変換コンストラクタと呼ばれる
- 変換コンストラクタには、明示的と暗黙的の、２種類の呼び出し方が存在する

クラス C に、C::C (int x); というコンストラクタが定義されている場合、

明示的呼び出し：

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

暗黙的呼び出し：

```
C obj = 10;
```

変換コンストラクタ（Circle 版）

- クラス Circle も、デフォルト変数を増やして変換コンストラクタにした場合、以下の代入が可能

```
Circle (int cx, int cy = 0, int r = 10) {
        x = cx; y = cy; rad = r;
}
```

明示的呼び出し：

```
Circle circ(10); // y = 0, rad = 10
```

暗黙的呼び出し：

```
Circle circ = 10; // y = 0, rad = 10
```

プリミティブ変数に対する構築子

- C++では、int float char 等の基本（プリミティブ）変数をクラスとみなして構築子で生成できる。

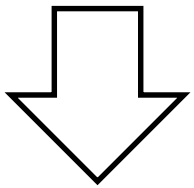
例：下記の命令は、どれも100の値の整数変数 n を生成している。

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

// 配列の生成 int *p = new int [100]; とは、
意味が異なる点に注意が必要！
```

構築子での用い方

```
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) { }
```

配列を用いた
クラスオブジェクトの操作法

配列の生成法（C言語の復習）

構造体 Health で100人分の配列生成

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

構造体へのポインタ変数を100人分確保して，実体は後から動的に生成

```
struct Health *data[100];
data[0] = (struct Health *) malloc(sizeof(struct Health));
data[1] = (struct Health *) malloc(sizeof(struct Health));
// 以下同様に続く…
```

上記と同じだが，ポインタ変数の数を動的に確保する場合

```
size = 50;
struct Health **data = (struct Health **) malloc(size * sizeof(struct Health*));
data[0] = (struct Health *) malloc(sizeof(struct Health));
//以下同様
```

配列の生成法

引数の無い構築子で100人分の静的な生成

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

クラスのポインタ変数を100人分確保して，実体は後から動的に生成

```
Health *data[100];
data[0] = new Health (“taro”, 1.7, 60);
data[1] = new Health (“hanako”, 1.6, 50);
// 以下同様に続く…
```

上記と同じだが，ポインタ変数の数を動的に確保する場合

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

クラス配列の使用例

```
class HealthGroupManager {
private:
    Health *data[100]; // 最大数 100人分の健康データ
    int numStudents = 0; // 登録済みの学生数

public:
    void setStudentData (char *n, float h, float w); // データ登録
    float getAverageBMI (); // BMI の平均値の計算
};
```

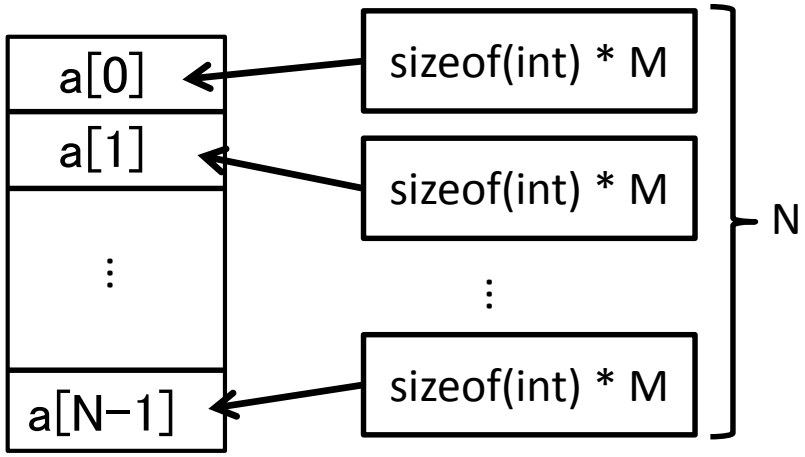
メンバ関数の実装例

```
// 1名分のデータの追加登録
void HealthGroupManager::setStudentData (char *n, float h, float w) {
    data[numStudents++]  = new Health (n, h, w);
}

// 全登録学生のBMIの平均値の取得
float HealthGroupManager::getAverageBMI () {
    float sumBMI = 0.0;
    for (int i = 0; i < numStudents; i++)
        sumBMI += data[i]->getBMI();
    return sumBMI / (float) numStudents;
}
```

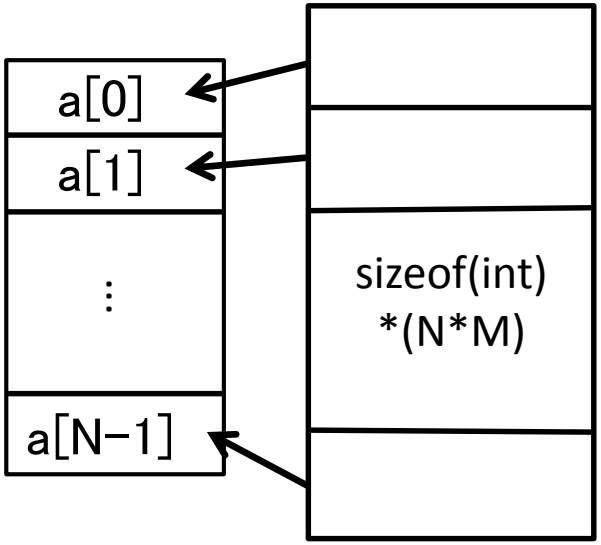
２重配列の生成法

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



メモリの連続領域に生成する場合：

```
int **a = new int*[N];
a[0] = new int[N * M];//全領域を一括確保
for (int i = 1; i < N; i++)
      a[i] = a[0] + i * M;//アドレス値の割当
```



Ｃ＋＋での
入出力関数

入出力関数

- C＋＋独自の標準入出力関数が存在する
  - 標準関数には std:: という prefix (接頭辞)を付ける
- データの入出力にはストリームが介在する
  - 入力には std::istream というクラスを用いる
  - 出力には std::ostream というクラスを用いる
  - これらのクラスは，iostream.h で提供される
  - 標準入出力は std::cin および std::cout というオブジェクトとして装備されている
- >> と << は，入力と出力データの流れを表す

ストリームの使用例

```
int main () {
    int n;
    char str[100];
    std::cout << ″整数値を入力して下さい¥n″;
    std::cin >> n; // キーボードから整数値を入力
    std::cout << ″整数値″ << n << ″が入力されました¥n″;
    std::cout << ″文字列を入力して下さい¥n″;
    std::cin >> str; // キーボードから文字列を入力
    std::cout << ″文字列″ << str << ″が入力されました¥n″;
}
```

注：バックスラッシュ「＼」がキーボードの ¥ では入力できない場合は、
option を押しながら ¥ を押す（日本語入力 [ことえり] の環境設定に依存する）

ストリーム使用法の新旧対比

旧方式 → コンパイルの際に警告が出力される

#include <iostream.h>

新方式→ 標準ライブラリの使用を明示する

#include <iostream> // 「.h」を付けない！

cin → std::cin , cout → std::cout と命名する

ただし、

using namespace std;

と、ファイル冒頭で標準ライブラリの名前空間を指定すると、std:: の記述は省略できる。

本演習では,「新方式」を推奨する！