new演算子とdelete演算子

- ヒープ領域を使って、インスタンス用のメモリを
  プログラマの使いたいタイミングで確保したり、
  いらなくなったらメモリを解放することができる

- new演算子
  必要なときに必要なぶんメモリを確保する

- delete演算子
  不要になったメモリを解放する

メモリ領域

- **プログラム領域**
  プログラム自体(機械語)を格納するメモリ領域
- **静的領域**
  グローバル変数を格納するメモリ領域
- **ヒープ領域**
  プログラム実行中、動的に確保されるメモリ領域
- **スタック領域**
  ローカル変数を格納するメモリ領域

new演算子とdelete演算子

- 教科書P144~145 **Sample402**

- Sample401フォルダから**Sample402**フォルダを作成

```
robocopy Sample401 Sample402
cd Sample402
```

- main.cppを編集

new演算子とdelete演算子

- main.cpp (Sample402)

```
int main() {
    Car* pkuruma = nullptr;
    pkuruma = new Car();
    pkuruma->setSpeed(40);
    pkuruma->drive(1.5);
    pkuruma->setSpeed(60);
    pkuruma->drive(2.0);
    cout << “総走行距離:” << pkuruma->getMigration()
         << “km” << endl;
    delete pkuruma;
    cout << “インスタンスの消去完了” << endl;
    return 0;
}
```

new演算子とdelete演算子

- コンパイルの仕方

  コマンドプロンプトで次のコマンドを入力する

```
cl␣/EHsc␣main.cpp␣car.cpp
```

  成功したら、main.exeを実行して結果を確認

new演算子とd

Carクラスのポインタ変数pkuruma
を宣言して nullptr で初期化
(NULLでも可。中身がない(NULL:0)な
ポインタという意味でnullptrを使う)

- main.cpp (Sa

```
int main() {
    Car* pkuruma = nullptr;
    pkuruma = new Car();
    pkuruma->setSpeed(40);
    pkuruma->drive(1.5);
    pkuruma->setSpeed(60);
    pkuruma->drive(2.0);
    cout << “総走行距離:” << pkuruma->getMigration()
         << “km” << endl;
    delete pkuruma;
    cout << “インスタンスの消去完了” << endl;
    return 0;
}
```

new演算子とdelete演算子

- main.cpp (Sample4

```
int main() {
    Car* pkuruma = nullpt
    pkuruma = new Car();
    pkuruma->setSpeed(40);
    pkuruma->drive(1.5);
    pkuruma->setSpeed(60);
    pkuruma->drive(2.0);
    cout << “総走行距離:” << pkuruma->getMigration()
         << “km” << endl;
    delete pkuruma;
    cout << “インスタンスの消去完了” << endl;
    return 0;  }
```

new演算子を使って、コンストラクタを実行して、インスタンスを生成
ポインタ変数 pkuruma に
インスタンスのアドレスを代入

ここで指定するのはコンストラクタ名でクラス名でないことに注意！

new演算子とdelete演算子

- main.cpp (Sample402)

```
int main() {
    Car* pkuruma = nullptr;
    pkuruma = new Car();
    pkuruma->setSpeed(40);
    pkuruma->drive(1.5);
    pkuruma->setSpeed(60);
    pkuruma->drive(2.0);
    cout << “総走行距離:” << pkuruma->getMigration()
         << “km” << endl;
    delete pkuruma;
    cout << “インスタンスの消去完了” << endl;
    return 0;  }
```

ポインタを用いる場合は「.」でなく「->（アロー）」を使う

new演算子とdelete演算子

- main.cpp (Sample402)

```
int main() {
    Car* pkuruma = nullptr;
    pkuruma = new Car();
    pkuruma->setSpeed(40);
    pkuruma->drive(1.5);
    pkuruma->setSpeed(60);
    pkuruma->drive(2.0);
    cout << “総走行距離:” << pkuruma->getMigration()
         << “km” << e
    delete pkuruma;
    cout << “インスタン
    return 0;  }
```

【重要!!】
new演算子を使って生成したインスタンスはdelete演算子を使って破棄しなければならない！（破棄しないとメモリリークが発生）

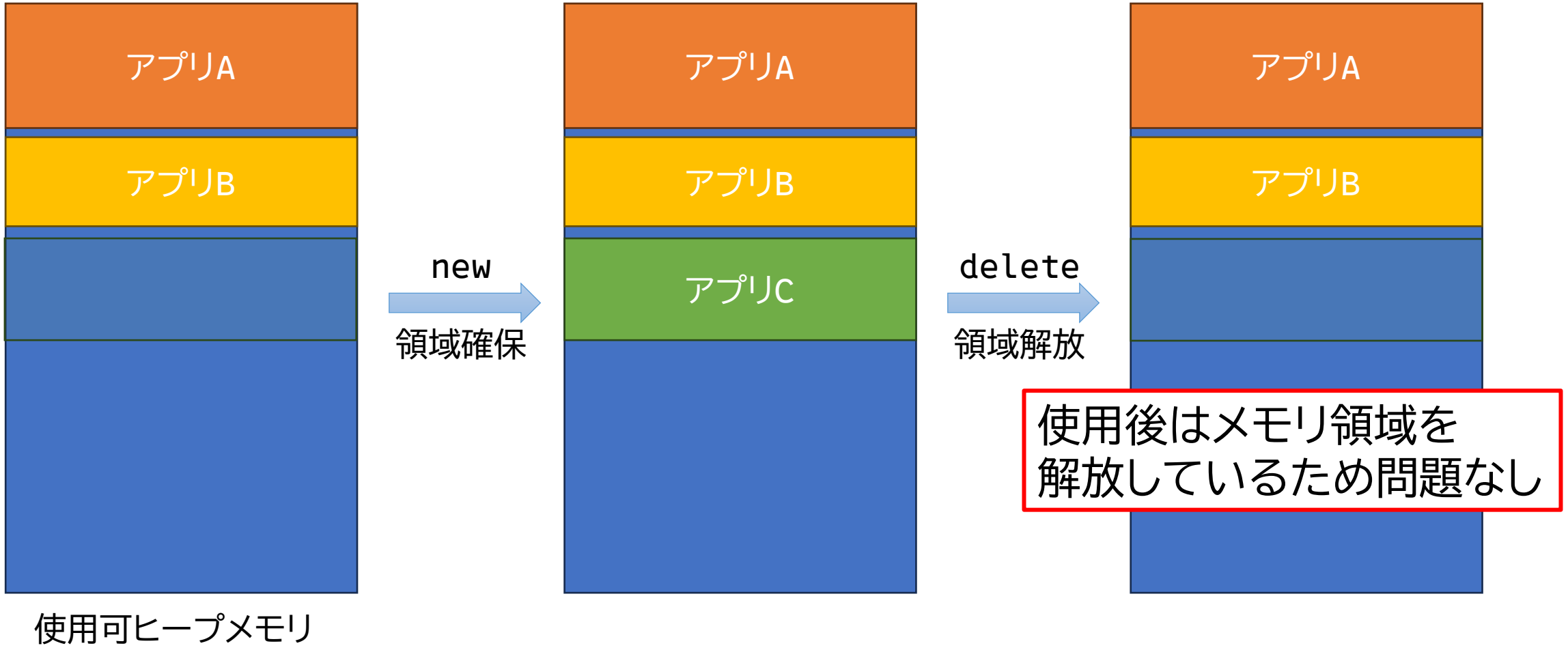
new演算子とdelete演算子

- メモリリーク
  プログラム中で動的に確保したメモリ領域を使用後に解放せずに放置することで、使用できるメモリ領域が次第に減っていくこと

  最終的には、システムが利用するためのメモリ領域もなくなるため、システムがハングアップしてしまうという危険性がある

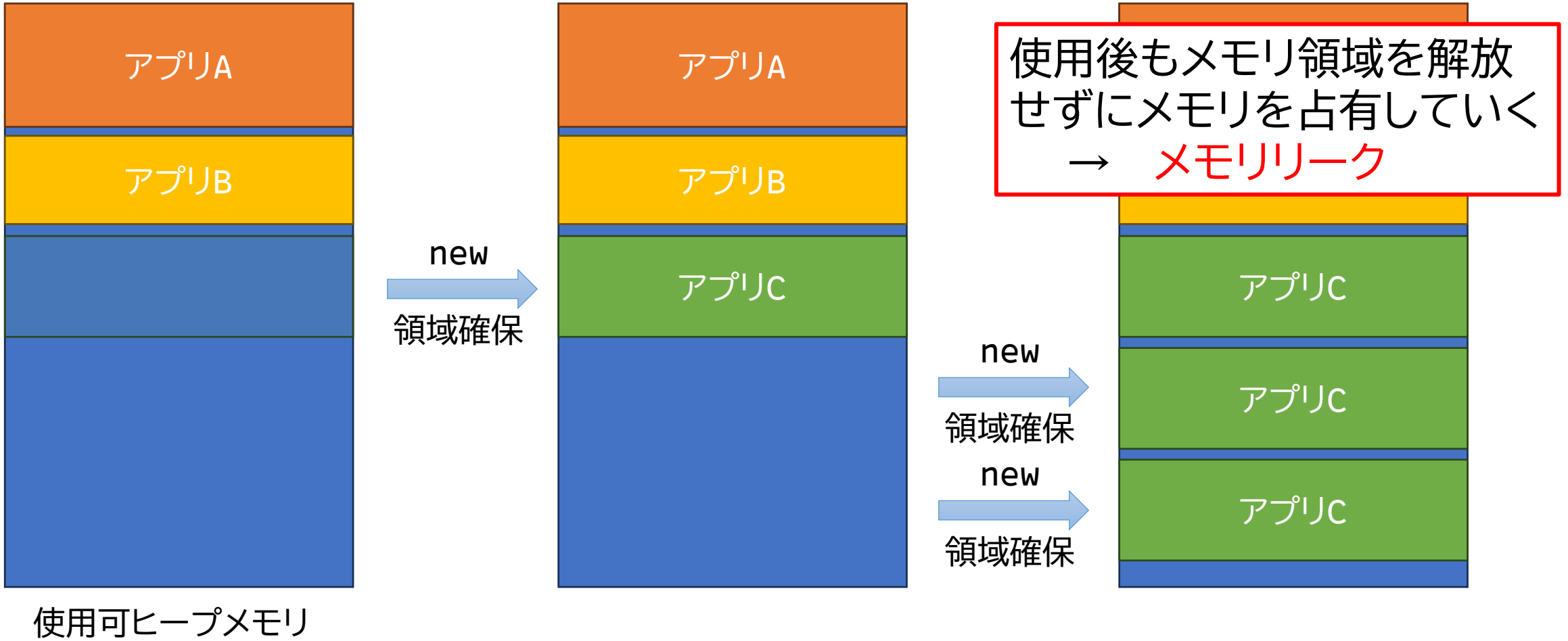
new演算子とdelete演算子

- メモリリーク



new演算子とdelete演算子

- メモリリーク



new演算子とdelete演算子

- スマートポインタを使用したmain.cpp

【参考プログラム】
今は覚えなくてよい

```
int main() {
    unique_ptr<Car> pkuruma(new Car());
    pkuruma->setSpeed(40);
    pkuruma->drive(1.5);
    pkuruma->setSpeed(60);
    pkuruma->drive(2.0);
    cout << “総走行距離:” << pkuruma->getMigration()
         << “km” << endl;
    //delete不要
    cout << “インスタンスの消去完了” << endl;
    return 0;
}
```

new

unique_ptr<型名> ポインタ変数名(初期値)

Carクラスのスマートポインタpkurumaを宣言し、初期値としてCarクラスのコンストラクタを呼び出すことで、インスタンスのアドレスを格納

- スマ

【参考プログラム】
覚えなくてよい

```
int
    unique_ptr<Car> pkuruma(new Car());
    pkuruma->setSpeed(40);
    pkuruma->drive(1.5);
    pkuruma->setSpeed(60);
    pkuruma->drive(2.0);
    cout << “総走行距離:” << pkuruma->getMigration()
         << “km”
    //delete不要
    cout << “インス
    return 0;
}
```

スマートポインタの一番の利点は使用後に**自動的にメモリ領域を解放**してくれること

new演算子とdelete演算子

- 教科書P148 **Sample403**

- **Sample403**フォルダを作成
```
mkdir Sample403
cd Sample403
```

- main.cppを作成
```
copy nul main.cpp
```

new演算子とdelete演算子

- main.cpp (Sample403)

```
include<iostream>
using namespace std;
int main() {
    int* p = nullptr;
    p = new int();
    *p = 123;
    cout << *p << endl;
    delete p;
    return 0;
}
```

new演算子とdelete演算子

- main.cpp (Sample403)

```
include<iostream>
using namespace std;
int main() {
    int* p = nullptr;
    p = new int();
    *p = 123;
    cout << *p
    delete p;
    return 0;
}
```

int型のポインタ変数 p を宣言してnullptrで初期化
new int()で整数値を格納できるメモリ領域を確保

使用していたメモリ領域（アドレスはpに格納）を解放

new演算子とdelete演算子

- main.cpp(スマートポインタ版)

【参考プログラム】
今は覚えなくてよい

```
include<iostream>
using namespace std;
int main() {
    unique_ptr<int> p(new int());
    *p = 123;
    cout << *p << endl;
    return 0;
}
```

deleteは不要

new演算子とdelete演算子

- main.cpp(スマートポインタ版)

【参考プログラム】
今は覚えなくてよい

```
include<iostream>
using namespace std;
int main() {
    unique_ptr<int> p(new int(123));
    //スマートポインタを宣言した際に初期値を与える方法
    cout << *p << endl;
    return 0;
}
```

deleteは不要

new演算子とdelete演算子

- 教科書P149 **Sample404**

- **Sample404**フォルダを作成
```
mkdir Sample404
cd Sample404
```

- main.cppを作成
```
copy nul main.cpp
```

new演算子とdelete演算子

- main.cpp (Sample404)

```
include<iostream>
using namespace std;
int main() {
    int* p = nullptr;
    p = new int[10];
    for(int i = 0; i < 10; i++){
        p[i] = i;
        cout << p[i] << “ “;
    }
    cout << endl;
    delete[] p;
    return 0;
}
```

new演算子とdelete演算子

- main.cpp (Sample404)

```
include<iostream>
using namespace std;
int main() {
    int* p = nullptr;
    p = new int[10];
    for(int i = 0; i < 10; i++){
        p[i] = i;
        cout << p[i] << " ";
    }
    cout << endl;
    delete[] p;
    return 0;
}
```

int型のポインタ変数 p を宣言してnullptrで初期化
new int[10]で10個の整数値を格納できる配列用メモリ領域を確保

配列の場合はdelete[]としてメモリ領域を解放

new演算子とdelete演算子

- main.cpp(スマートポインタ版)

【参考プログラム】
今は覚えなくてよい

```
include<iostream>
using namespace std;
int main() {
    unique_ptr<int[]> p(new int[10]);
    for(int i = 0; i < 10; i++){
        p[i] = i;
        cout << p[i] << “ “;
    }
    cout << endl;
    return 0;
}
```

deleteは不要

new演算子とdelete演算子

- 通常のポインタ(生ポインタ)とスマートポインタではどちらを使うべき？
  - スマートポインタのほうがバグの少ないプログラムを記述可能(deleteのし忘れがない)
  - しかし、スマートポインタが新しいC++の機能のため、対応していない教科書も多い(このテキストも)
  - 授業ではテキストに沿って、生ポインタを使用する