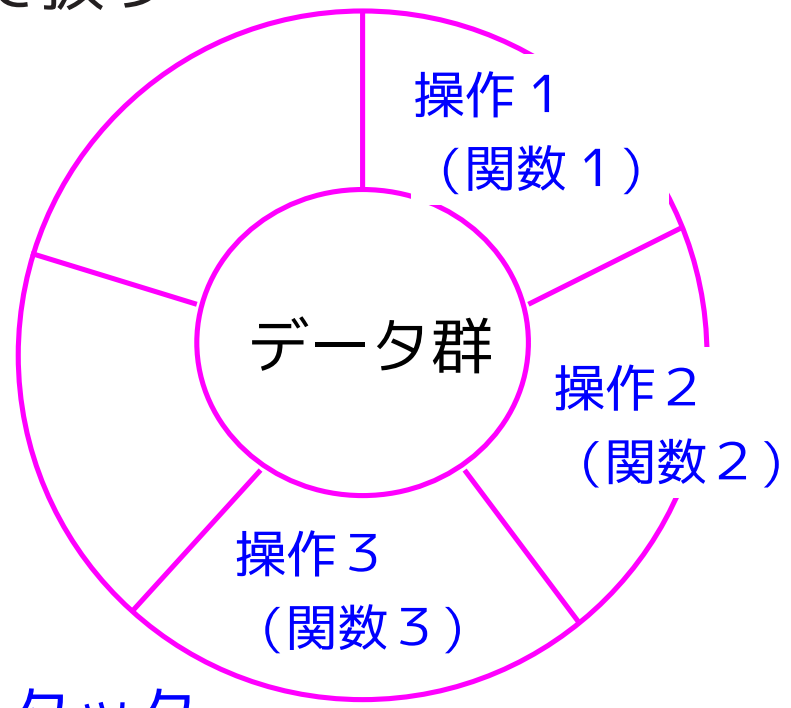
3 C言語構造体の考えの拡張、クラス

3–1 C言語の下でのpush-downスタックの実現

オブジェクト指向においては

> 関連するデータ群と操作(関数)を1つにまとめてカプセル化し、
> ソフトウェア部品(オブジェクト)として扱う

ということが基本



⇨ まず、試しに C言語の下で

char型データが最大100個入るスタック

をソフトウェア部品として提供することを考える。

実装例3. 1 (char型データが最大100個入るスタック,部品提供の実装案1

一般に、C言語では、
外部変数や関数にstatic修飾子を付けると、
それらの名前の有効範囲が同一ソースファイル内に限定される。

⇨ ①1つのソースファイルの中に関連するデータ群と関数を入れ、
②隠蔽したいデータと関数にstatic修飾子を付ければ、
このソースファイルはカプセル化され情報隠蔽された
ソフトウェア部品として機能する。

以上の考えに基づいて
「char型データが最大100個入るスタック」部品を
Cソースファイルの形で表した例を次に示す。

```
[motoki@x205a]$  cat -n stackA_char100.c
     1 /*************************************************
     2 /* char型データが最大100個入るスタックを実装したモジュ...
     3 /*-------------------------------------
     4 /*  外部へのサービスを行うために、次の４つの関数がこの
```

```
 5 /*  モジュールの中に用意されている。
 6 /*  (1)スタックを空に初期化する関数 initializeStackA,
 7 /*  (2)スタックが空かどうかを調べる関数 isStackAEmpty,
 8 /*  (3)スタックに要素を1つ push-down する関数 pushdownInt
 9 /*  (4)スタックから要素を1つ pop-up する関数 popupFromSta
10 /*********************************************************
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 typedef  int  Boolean;
15
16 static char element[100]; /*モジュール外からは見えない*/
17 static int  top;          /*モジュール外からは見えない*/
18
19 /* 外部へのサービスを行うための関数群 */
20 void initializeStackA(void)
21 {
```

```
22   top = -1;
23 }
24
25 Boolean isStackAEmpty(void)
26 {
27   return (top < 0);
28 }
29
30 void pushdownIntoStackA(char c)
31 {
32   if (++top >= 100) {
33     printf("stack overflow\n");
34     exit(1);
35   }
36   element[top] = c;
37 }
38
```

```
39 char popupFromStackA(void)
40 {
41   if (top < 0) {
42     printf("popup from empty stack\n");
43     exit(1);
44   }
45   return element[top--];
46 }
[motoki@x205a]$
```

これに関して、

- このソフトウェア部品を利用した例

```
[motoki@x205a]$  cat -n reverseWordThroughStack1.c
     1 /* 文字列を1個読み込み、
     2 /* それをスタックを用いて反転した後出力するCプログラム */
     3 /* (スタックを1個のソースファイルで実装する版)          */
     4
     5 #include <stdio.h>
```

```
 6 typedef  int  Boolean;
 7
 8 void initializeStackA(void);
 9 Boolean isStackAEmpty(void);
10 void pushdownIntoStackA(char c);
11 char popupFromStackA(void);
12
13 int main(void)
14 {
15   char s[100];
16   int  i;
17
18   printf("Input a string:  ");
19   scanf("%s", s);
20   initializeStackA();
21   for (i=0; s[i]!='\0'; ++i)
22     pushdownIntoStackA(s[i]);
```

```
23    for (i=0; !isStackAEmpty(); ++i)
24      s[i] = popupFromStackA();
25    printf("Reversed string: %s\n", s);
26    return 0;
27  }
[motoki@x205a]$  gcc reverseWordThroughStack1.c
                                        stackA_char100.c
[motoki@x205a]$  ./a.out
Input a string:   abc123
Reversed string: 321cba
[motoki@x205a]$
```

- このソフトウェア部品提供方式の利点・欠点は

  (利点) 「後入れ先出し」(**LIFO**) の原則が保証される。

  (欠点) 同種の部品が複数必要な場合にうまく対処できない。
  (必要な分だけ類似のソースファイルを用意する必要)

実装例3. 2 (char型データが最大100個入るスタック,部品提供の実装案2

先の実装例3.1の欠点を解消し、
必要なだけ簡単にスタックをソフトウェア部品として生成する
ための C言語の方法としては 、構造体の利用が考えられる。

この方向での実装案として、
①スタック領域とtopの番号をメンバにもつ構造体の枠組みの定義と、
②関連する操作(関数)群
を1つのファイルにまとめた例を次に示す。

```
[motoki@x205a]$  cat -n struct_stackChar100.h
     1 /* 「char型データが最大100個入るスタック」を表す構造体
                                                の枠組みと */
     2 /* これらの構造体を操作するための関数群
     3
     4 #include <stdio.h>
     5 #include <stdlib.h>
     6 typedef int Boolean;
```

```
 7
 8 struct stackChar100 {
 9   char element[100];
10   int  top;
11 };
12
13 /* StackChar100型構造体を操作するための関数群 */
14 void initialize(struct stackChar100 *stack)
15 {
16   stack->top = -1;
17 }
18
19 Boolean isEmpty(struct stackChar100 *stack)
20 {
21   return (stack->top < 0);
22 }
23
```

```
24 void pushdown(struct stackChar100 *stack, char c)
25 {
26   if (++stack->top >= 100) {
27     printf("stack overflow\n");
28     exit(1);
29   }
30   stack->element[stack->top] = c;
31 }
32
33 char popup(struct stackChar100 *stack)
34 {
35   if (stack->top < 0) {
36     printf("popup from empty stack\n");
37     exit(1);
38   }
39   return stack->element[stack->top--];
40 }
```

[motoki@x205a]$

これに関して、

- このソフトウェア部品提供の枠組みを利用した例

[motoki@x205a]$ cat -n reverseWordThroughStack2.c

```
 1 /* 文字列を1個読み込み、
 2 /* それをスタックを用いて反転した後出力するCプログ... */
 3 /* (スタックを構造体で実装する版)
 4
 5 #include <stdio.h>
 6 #include <stdlib.h>
 7 #include "struct_stackChar100.h"
 8
 9 int main(void)
10 {
11   struct stackChar100 stack;
12   char s[100];
13   int  i;
```

```
14
15    printf("Input a string:  ");
16    scanf("%99s", s);
17    initialize(&stack);
18    for (i=0; s[i]!='\0'; ++i)
19      pushdown(&stack, s[i]);
20    for (i=0; !isEmpty(&stack); ++i)
21      s[i] = popup(&stack);
22    printf("Reversed string: %s\n", s);
23    return 0;
24 }
[motoki@x205a]$  gcc reverseWordThroughStack2.c
[motoki@x205a]$  ./a.out
Input a string:   abc123
Reversed string: 321cba
[motoki@x205a]$
```

- このソフトウェア部品提供方式の利点・欠点は

(利点) 同種の部品が複数必要な場合にもうまく対処できる。

(欠点) スタック (構造体) 内のデータは全て直接の操作が可能

⇨ 「後入れ先出し」(**LIFO**) の原則は保証されない。

3–2 C言語構造体の考えの拡張、クラス

先の実装例3.1～ 3.2から、
もし
① 構造体の中にデータを操作する関数を入れることができ、また
② 必要に応じて構造体メンバの隠蔽もできる
様になっていれば 、

> 生成されたそれぞれの構造体は
> カプセル化され情報隠蔽されたソフトウェア部品として機能する、

と考えられる。

この様な考えの下、実際に
C++言語では 構造体に関する上記①～②の拡張が行われている。

実装例3. 3 (char型データが最大100個入るスタック,部品提供の実装案3

C++言語の下で、

上記①~②の拡張に沿って実装例3.2 の構造体定義を書き直した例

[motoki@x205a]$ cat -n StackChar100_ver0struct.h

```
 1 /* 「char型データが最大100個入るスタック」を表す構造体... */
 2
 3 #include <iostream>
 4 #include <cstdlib>
 5                 汎用性を保つため「using namespace std;」なし
 6 struct StackChar100 {
 7 private:
 8   char element[100];                                  非公開
 9   int  top;                                          非公開
10 public: //StackChar100型構造体を操作するための関数群
11   void initialize() { top = -1; }          暗黙に「inline」
12   bool isEmpty() { return (top < 0); }
13   void pushdown(char c);                本体はstruct構文の外で
```

```
14    char popup();
15 };
16                          ↓                          スコープ解決演算子
17 void StackChar100::pushdown(char c)                 非「inline」
18 {
19    if (++top >= 100) {
20      std::cout << "stack overflow" << std::endl;
21      exit(1);
22    }
23    element[top] = c;
24 }
25
26 char StackChar100::popup()                          非「inline」
27 {
28    if (top < 0) {
29      std::cout << "popup from empty stack" << std::endl;
30      exit(1);
```

```
31    }
32    return element[top--];
33 }
[motoki@x205a]$
```

これに関して、

- このソフトウェア部品提供の枠組みを利用した例

```
[motoki@x205a]$  cat -n reverseWordThroughStack3.cpp
     1 /* 文字列を1個読み込み、
     2 /* それをスタックを用いて反転した後出力するC++プロ... */
     3 /* (スタックをC++構造体で実装する版)
     4
     5 #include <iostream>
     6 #include <string>
     7 #include "StackChar100_ver0struct.h"
     8 using namespace std;
     9
    10 int main()
```

```
11 {
12   StackChar100 stack;
13   string s;
14
15   cout << "Input a string:  ";
16   cin >> s;
17   stack.initialize();
18   for (int i=0; i < s.length(); ++i)
19     stack.pushdown(s[i]);
20   for (int i=0; !stack.isEmpty(); ++i)
21     s[i] = stack.popup();
22   cout << "Reversed string: " << s << endl;
23 }
```

「struct」不要

```
[motoki@x205a]$  g++ reverseWordThroughStack3.cpp
[motoki@x205a]$  ./a.out
Input a string:   abc123
Reversed string: 321cba
```

```
[motoki@x205a]$
```

- このソフトウェア部品提供方式の利点・欠点は

  (利点) 同種の部品が複数必要な場合にもうまく対処できる。

  (利点) 「後入れ先出し」(**LIFO**) の原則が保証される。

実装例3. 4 (char型データが最大100個入るスタック,部品提供の実装案4

先の実装例3.3では、

目的とするソフトウェア部品(オブジェクト)の設計図

をstruct構文を用いて定義できることを示した。

同等の定義は class構文を用いて行うこともできる。

[motoki@x205a]$ cat -n StackChar100_ver1.h

```
 1 /* 「char型データが最大100個入るスタック」を表すオブジェ... *
 2
 3 #include <iostream>
 4 #include <cstdlib>
 5
 6 class StackChar100 {                    暗黙に「private:」
 7   char element[100];                          非公開
 8   int  top;                                   非公開
 9 public: //StackChar100型構造体を操作するための関数群
10   void initialize() { top = -1; }
```

```
11    bool isEmpty() { return (top < 0); }
12    void pushdown(char c);
13    char popup();
14 };
15
16 void StackChar100::pushdown(char c)
17 {
18    if (++top >= 100) {
19       std::cout << "stack overflow" << std::endl;
20       exit(1);
21    }
22    element[top] = c;
23 }
24
25 char StackChar100::popup()
26 {
27    if (top < 0) {
```

```
    28      std::cout << "popup from empty stack" << std::endl;
    29      exit(1);
    30    }
    31    return element[top--];
    32 }
[motoki@x205a]$
```

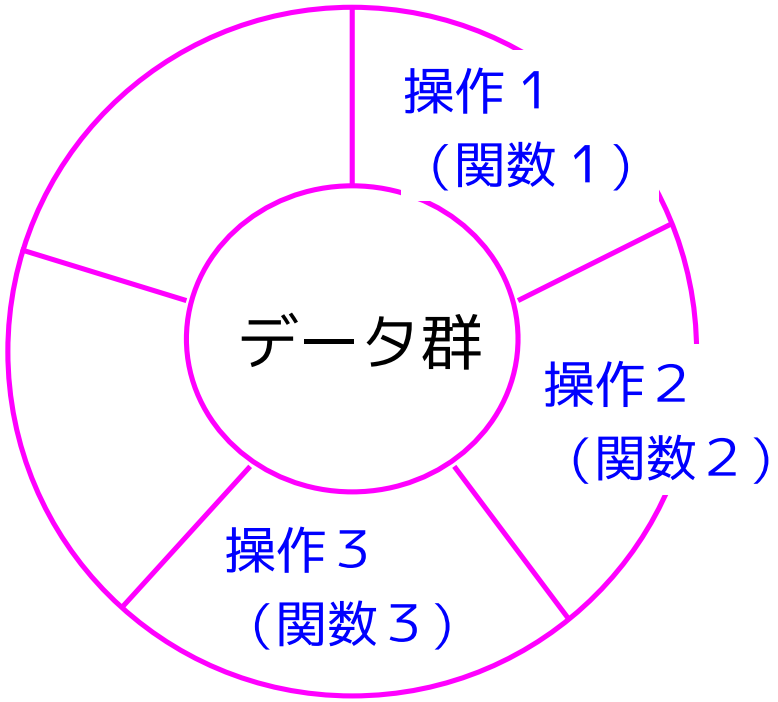
補足: structとclassの効果の違いは、
オブジェクト外からのアクセスに関する
暗黙の設定 (公開／非公開) がどうなるか
ということだけ。

⇨ 結局は構造体の枠組み／設計図が定義される。

一般に、

クラス ... C++言語のstruct構文やclass構文等で定義される
ソフトウェア部品(オブジェクト)の枠組み／設計図／種類

インスタンス ... クラスに属するソフトウェア部品



3–3 コンストラクタとデストラクタ, 静的メンバ

一般にソフトウェア部品は使う前に然るべき初期化を行うべき

⇨ C++言語では 、

- クラス設計図を基にソフトウェア部品(オブジェクト)を生成する際に
  初期化のために自動的に呼び出される関数を用意できる。
  (コンストラクタ)

- ブロック終了等に伴うオブジェクト消滅の際に
  後始末のために自動的に呼び出される関数も用意できる。
  (e.g. 領域開放) (デストラクタ)

- コンストラクタの名前... クラス名
  デストラクタの名前 ... ~クラス名

—

実装例3. 5 (char型データが入るスタック,部品提供の実装案5)

実装例3.4 のクラス定義にコンストラクタとデストラクタを加えて
スタックの容量をオブジェクト生成時に設定できる様にした例

[motoki@x205a]$ cat -n StackChar_ver1.h

```
 1 /* 「char型データが入るスタック」を表すオブジェクトの枠組... *
 2
 3 #include <iostream>
 4 #include <cstdlib>
 5
 6 class StackChar {
 7   char* element;
 8   int  size;
 9   int  top;
10 public:↓
11   explicit StackChar(int size = 100);
12   ~StackChar() { delete[] element; }
13   //StackChar型オブジェクトを操作するための関数群
```

データを入れる領域,ヒープ領域から確保

確保したデータ領域の容量

「型変換(引数の型→オブジェクトの型)に使わない」

戻り値型指定なし

戻り値型指定なし

```
14    bool isEmpty() const { return (top < 0); }
15    void pushdown(char c);
16    char popup();
17 };
18
19 StackChar::StackChar(int size): size(size), top(-1)
20 {
21    element = new char[size];
22 }
23
24 void StackChar::pushdown(char c)
25 {
26    if (++top >= size) {
27       std::cout << "stack overflow" << std::endl;
28       exit(1);
29    }
30    element[top] = c;
```

コンストラクタ引数による初期値指定

初期値指定

↓ ↓

↑ ↑

メンバ名 メンバ名

初期化子リスト

```
31 }
32
33 char StackChar::popup()
34 {
35   if (top < 0) {
36     std::cout << "popup from empty stack" << std::endl;
37     exit(1);
38   }
39   return element[top--];
40 }
[motoki@x205a]$
```

これに関して、

- プログラム 19行目 の「: size(size), top(-1)」の部分は初期化子リストと呼ばれるもので、メンバの初期化をどう行うかを指示

> 補足: オブジェクトメンバへの初期設定は 次の形で書ける。
>
> メンバ名 (コンストラクタへの実引数列)

- プログラム 19~22行目 は次の様に書くのと同じ

```
StackChar::StackChar(int size)
{   ↓                                  自オブジェクトへのポインタ
  this->size = size;
  element = new char[size];
  this->top = -1;
}
```

> 補足: オブジェクトメンバへの初期設定は 次の形で書ける。
>
> メンバ名 = クラス名 (コンストラクタへの実引数列);
>
> しかし、 この書き方では オブジェクトが一時的に作られ
> それが変数 メンバ名 に代入されるので、処理が非効率

- このソフトウェア部品提供の枠組みを利用した例

```
[motoki@x205a]$  cat -n reverseWordThroughStack5.cpp
     1 /* 文字列を1個読み込み、
     2 /* それをスタックを用いて反転した後出力するC++プログ.. *
     3 /* (スタックをコンストラクタ付きC++クラスのインスタン...
     4
     5 #include <iostream>
     6 #include <string>
     7 #include "StackChar_ver1.h"
     8 using namespace std;
     9
    10 int main()
    11 {                              ↓          コンストラクタへの引数
    12   StackChar stack(100);                    コンストラクタ呼出し
```

補足: この場合は次の書き方も可能

```
StackChar stack = StackChar(100);
```

しかし、これだと 一時的オブジェクトが作られるので処理が非効率

```
13    string s;
14
15    cout << "Input a string:  ";
16    cin >> s;
17    for (int i=0; i < s.length(); ++i)
18      stack.pushdown(s[i]);
19    for (int i=0; !stack.isEmpty(); ++i)
20      s[i] = stack.popup();
21    cout << "Reversed string: " << s << endl;
22 }
[motoki@x205a]$  g++ reverseWordThroughStack5.cpp
[motoki@x205a]$  ./a.out
Input a string:   abc123
Reversed string: 321cba
[motoki@x205a]$
```

実装例3. 6 (char型データが入るスタック,部品提供の実装案6)

クラス定義は将来の使用も念頭に置いて行うべき

⇨ 実装例3.5のクラス定義に有用そうなメンバを加えてクラスの拡充

```
[motoki@x205a]$  cat -n StackChar_ver2.h
     1 /* 「char型データが入るスタック」を表すオブジェクトの枠組... *
     2
     3 #include <iostream>
     4 #include <string>
     5 #include <cstdlib>     // for exit()
     6 #include <cstring>     // for memcpy()
     7
     8 class StackChar {
           ↓                          「クラス全体で共有するメンバ」の意
     9   static int numOfInstances;   生成されたインスタンス数
    10   const int id;    インスタンスに固有のid番号,値変更不可
    11   char* element;
```

```
12    int  size;
13    int  top;
14  public:
15    explicit StackChar(int size = 100);
                                  //デフォルトコンストラクタを含む
16    StackChar(const StackChar& stack);
                                        // コピーコンストラクタ
17    StackChar(const std::string& str);
18    StackChar(int size, const std::string& str);
19    ~StackChar() { delete[] element; }
20    // オブジェクト(やStackCharクラス全体)の情報を提供
                                        //するための関数群
21    static int getNumOfInstances() {
                                    return numOfInstances; }
22    int getId() const { return id; }
23    int getSize() const { return size; }
24    int getNumOfElements() const { return top+1; }
```

```
25    void showContents() const;
26    // StackChar型オブジェクトを操作するための関数群
27    void reset() { top = -1; }
28    void resize(int size);
29    bool isEmpty() const { return (top < 0); }
30    void pushdown(char c);
31    char popup();
32 };
33
34 // static変数の初期化 ------------------------------
35 int StackChar::numOfInstances = 0;
36
37 // 各種コンストラクタ ------------------------------
```

スタック内部の状況を表示

スタックを空にリセット

スタック容量を変更

拡張,満杯の場合は容量変更

クラス定義の外でstatic修飾子も付けずに行う

```
38 StackChar::StackChar(int size)
                                  // デフォルトコンストラクタを含む
39             : id(numOfInstances++), size(size), top(-1)
40 {
41   element = new char[size];
42 }
43
44 StackChar::StackChar(const StackChar& stack)
                                          // コピーコンストラクタ
45             : id(numOfInstances++), size(stack.size),
                                                top(stack.top)
46 {
47   element = new char[stack.size];
48   memcpy(element, stack.element, stack.size);
49 }                                  バイト列を別配列にコピー
50
```

```
51 StackChar::StackChar(const std::string& str)
52           : id(numOfInstances++), size(str.length()),
                                     top(str.length()-1)
53 {
54   element = new char[size];
55   for (int i=0; i<str.length(); i++)
56     element[i] = str[i];
57 }
58
59 StackChar::StackChar(int size, const std::string& str)
60           : id(numOfInstances++), size(size),
                                     top(str.length()-1)
61 {
62   element = new char[size];
63   for (int i=0; i<str.length(); i++)
64     element[i] = str[i];
65 }
```

```
66
67 // オブジェクトの情報を提供するための関数群 ----------
68 void StackChar::showContents() const
                                        // スタックの内容を表示
69 {
70   std::cout << "stack_of_char(id=" << id << ",size=" <<
71             << top+1 << " elements as follows:" << std:
72   std::cout << "  [Bottom] ";
73   for (int i=0; i<=top; i++)
74     std::cout << element[i] << "  ";
75   std::cout << "<--" << std::endl;
76 }
77
```

```
78 // StackChar型オブジェクトを操作するための関数群 ---------
79 void StackChar::resize(int size)
80 {
81   if (top >= size) {
82     std::cout << "insufficient stack size"
                                          << std::endl;
83     exit(1);
84   }
85   char* temp = new char[size];
86   memcpy(temp, element, top+1);バイト列を別配列にコピー
87   delete[] element;
88   this->size = size;
89   element = temp;
90 }
91
```

```
 92 void StackChar::pushdown(char c)     // pushdown操作
 93 {
 94   if (++top >= size) {
 95     this->resize(size+10);
 96   }
 97   element[top] = c;
 98 }
 99
100 char StackChar::popup()               // popup操作
101 {
102   if (top < 0) {
103     std::cout << "popup from empty stack"
                                          << std::endl;
104     exit(1);
105   }
106   return element[top--];
```

```
  107 }
[motoki@x205a]$
```

これに関して、

- このソフトウェア部品提供の枠組みを利用した例

```
[motoki@x205a]$  cat -n useStackChar_ver2.cpp
     1 /* "StackChar_ver2.h"の利用例
     2 /* (1)文字列を1個読み込みそれをスタックを用いて反転した
                                                後出力 */
     3 /* (2)デフォルトコンストラクタの利用
     4 /* (3)コピーコンストラクタの利用
     5
     6 #include <iostream>
     7 #include <string>
     8 #include "StackChar_ver2.h"
     9 using namespace std;
    10
    11 int main(void)
```

```
12 {
13   StackChar stackA(100);
14   string s;
15
16   cout << "Input a string:  ";
17   cin >> s;
18   for (int i=0; i < s.length(); ++i)
19     stackA.pushdown(s[i]);
20   for (int i=0; !stackA.isEmpty(); ++i)
21     s[i] = stackA.popup();
22   cout << "Reversed string: " << s << endl;
23   cout << "---" << endl;
24
25   StackChar stackB;    //デフォルトコンストラクタの利用
26   StackChar stack[3]; //デフォルトコンストラクタの利用
27   for (int i=0; i<3; ++i) {
28     for (int k=0; k<=i; ++k)
```

```
29        stack[i].pushdown('*');
30    }
31    stackB.showContents();
32    for (int i=0; i<3; ++i)
33      stack[i].showContents();
34    cout << "---" << endl;
35
36    cout << "s = \"" << s << "\"" << endl;
37    StackChar stackC(s);
38    StackChar stackD(stackC);
                                  //コピーコンストラクタの利用
39    stackD.pushdown('*');
40    StackChar stackE = stackD;
                                  //コピーコンストラクタの利用？
41    stackE.popup();
42    stackE.pushdown('#');
43    stackC.showContents();
```

```
44    stackD.showContents();
45    stackE.showContents();
46    cout << "---" << endl;
47
48    cout << "生成されたStackCharインスタンス数 = "
49         << StackChar::getNumOfInstances() << endl;
50 }                          ↑                    「.」ではない
[motoki@x205a]$  g++ useStackChar_ver2.cpp
[motoki@x205a]$  ./a.out
Input a string:   abc123
Reversed string: 321cba
---
stack_of_char(id=1,size=100) has 0 elements as follows:
  [Bottom] <--
stack_of_char(id=2,size=100) has 1 elements as follows:
  [Bottom] *  <--
stack_of_char(id=3,size=100) has 2 elements as follows:
```

```
  [Bottom] *  *  <--
stack_of_char(id=4,size=100) has 3 elements as follows:
  [Bottom] *  *  *  <--
---
s = "321cba"
stack_of_char(id=5,size=6) has 6 elements as follows:
  [Bottom] 3  2  1  c  b  a  <--
stack_of_char(id=6,size=16) has 7 elements as follows:
  [Bottom] 3  2  1  c  b  a  *  <--
stack_of_char(id=7,size=16) has 7 elements as follows:
  [Bottom] 3  2  1  c  b  a  #  <--
---
生成されたStackCharインスタンス数 = 8
[motoki@x205a]$
```

3–4 ソースファイルの構成

クラスの仕様部と実装部の分離：

定義したクラスは将来再利用する可能性もある。
ただ、多くの場合、

- ◇ 構成メンバの情報や公開関数の仕様／使い方が分かれば十分、
- ◇ クラスの実装部 (e.g. メンバ関数の本体部) を見る必要がない

⇨ クラスを定義するファイルを構成する際は、

- クラスの仕様部と実装部をソースファイル上で分離すべき

具体的には、

- ○クラスの構成メンバを記述した

```
class クラス名 {
    (メンバの記述)
};
```

という部分1個と、

○inline宣言するメンバ関数の定義、および

○これらに関連した記述

だけから成るヘッダファイル クラス名.h を構成する。

その際、

◇ 将来の利用に備えて、十分なコメントを入れておく。

◇ 複数回インクルードした場合も「重複定義」のコンパイルエラーは避けたい。

⇨ インクルードガードという定石手法に従って、
例えば次の様に クラス名.h を構成する。

```
#ifndef ___Class_クラス名
#define ___Class_クラス名
(クラス定義等)
#endif
```

- 上記のヘッダファイル クラス名.h に含まれない
  - ◇ メンバ関数の詳細な定義
    (注意： inline関数は クラス名.h の方に入れる)、
  - ◇ 静的メンバ変数への初期設定、
  - ◇ #include クラス名.h

  等の記述から成るファイル クラス名.cpp を構成する。

  そして、将来の保守のために、コメントを入れておく。

この様なファイル構成のクラスを利用する際は、

- クラス利用のソースファイル上で 、

  ```
  #include クラス名.h
  ```

  とする。

- ソースファイル　クラス名.cpp　も指定してコンパイル

実装例3. 7 (char型データが入るスタック,部品提供の実装案7)

実装例3.6で与えたStackChar_ver2.h を
仕様部 StackChar.h と実装部 StackChar.cpp の2つに分離した例

```
[motoki@x205a]$  cat -n StackChar.h
     1 /* 「char型データが入るスタック」オブジェクトのクラス */
     2 /*  StackChar の仕様部                                  *
     3
     4 #ifndef ___Class_StackChar
     5 #define ___Class_StackChar
     6
     7 #include <string>
     8
     9 class StackChar {
    10   static int numOfInstances;//これまでに生成したStackCha
    11   const int id;     // インスタンスに固有のid番号
    12   char* element;    // スタック領域へのポインタ
    13   int  size;        // スタックの容量
```

```
14    int  top;          // スタックのtop要素を保持する配列要...
15 public:
16    explicit StackChar(int size = 100); // スタック容量=s
17                                        // (デフォルトコ...
18    StackChar(const StackChar& stack);  // コピーコンスト.
19    StackChar(const std::string& str);//引数の文字列をス...
20    StackChar(int size,                 // 容量がsizeのス..
21              const std::string& str); //  そこに引数の文.
22    ~StackChar() { delete[] element; }
23    // オブジェクト(もしくはStackCharクラス全体)の情報を提...
24    static int getNumOfInstances() {
                                  return numOfInstances; }
25                          // これまでに生成したStackCharインス..
26    int getId() const { return id; }              // スタッ.
27    int getSize() const { return size; }            // ス.
28    int getNumOfElements() const { return top+1; }//スタ.
29    void showContents() const;                      // ス.
```

```
30   // StackChar型オブジェクトを操作するための関数群
31   void reset() { top = -1; }                          // スタッ
32   void resize(int size);                              // スタッ
33   bool isEmpty() const { return (top < 0); }          // スタッ
34   void pushdown(char c);                              // pushdo
35   char popup();                                       // popup
36 };
37
38 #endif
[motoki@x205a]$  cat -n StackChar.cpp
 1 /* 「char型データが入るスタック」オブジェクトのクラス */
 2 /*  StackChar の実装部                                  *
 3
 4 #include <iostream>
 5 #include <string>
 6 #include <cstdlib>    // for exit()
 7 #include <cstring>    // for memcpy()
```

```
 8 #include "StackChar.h"
 9 using namespace std;
10
11 // static変数の初期化 ----------------------------------
12 int StackChar::numOfInstances = 0;
13
14 // 各種コンストラクタ ----------------------------------
15 StackChar::StackChar(int size)       // スタック容量=siz
16                                      // (デフォルトコン...)
17           : id(numOfInstances++), size(size), top(-1)
18 {
19   element = new char[size];
20 }
21
22 StackChar::StackChar(const StackChar& stack)     // コ.
23           : id(numOfInstances++), size(stack.size),
                                          top(stack.top)
```

```
24 {
25   element = new char[stack.size];
26   memcpy(element, stack.element, stack.size);
27 }
28
29 StackChar::StackChar(const string& str) //引数文字列を...
30           : id(numOfInstances++), size(str.length()),
                                     top(str.length()-1)
31 {
32   element = new char[size];
33   for (int i=0; i<str.length(); i++)
34     element[i] = str[i];
35 }
36
37 StackChar::StackChar(int size,          //容量がsizeのス...
38                      const string& str)//そこに引数の文.
39           : id(numOfInstances++), size(size),
```

top(str.length()-1)

```
40 {
41   element = new char[size];
42   for (int i=0; i<str.length(); i++)
43     element[i] = str[i];
44 }
45
46 // オブジェクトの情報を提供するための関数群 ----------------
47 void StackChar::showContents() const                    // ス
48 {
49   cout << "stack_of_char(id=" << id << ",size="
                                      << size << ") has "
50        << top+1 << " elements as follows:" << endl;
51   cout << "  [Bottom] ";
52   for (int i=0; i<=top; i++)
53     cout << element[i] << "  ";
54   cout << "<--" << endl;
```

```
55 }
56
57 // StackChar型オブジェクトを操作するための関数群 ---------
58 void StackChar::resize(int size)                    // ス
59 {
60   if (top >= size) {
61     cout << "insufficient stack size" << endl;
62     exit(1);
63   }
64   char* temp = new char[size];
65   memcpy(temp, element, top+1);
66   delete[] element;
67   this->size = size;
68   element = temp;
69 }
70
71 void StackChar::pushdown(char c)                    // pu
```

```
72 {
73   if (++top >= size) {
74     this->resize(size+10);
75   }
76   element[top] = c;
77 }
78
79 char StackChar::popup()                                    // p
80 {
81   if (top < 0) {
82     cout << "popup from empty stack" << endl;
83     exit(1);
84   }
85   return element[top--];
86 }
[motoki@x205a]$
```

これに関して、

- このソフトウェア部品提供の枠組みを

実装例3.6の場合に倣って利用した例

```
[motoki@x205a]$  cat -n useStackChar.cpp
     1 /* "StackChar_ver3.h, StackChar_ver3.cpp"の利用例
     2 /* (1)文字列を1個読み込みそれをスタックを用いて反転した
                                                後出力 */
     3 /* (2)デフォルトコンストラクタの利用
     4 /* (3)コピーコンストラクタの利用
     5
     6 #include <iostream>
     7 #include <string>
     8 include "StackChar.h"
     9 using namespace std;
    10
    11 int main(void)
    12 {
    13   StackChar stackA(100);
```

```
14    string s;
15
16    cout << "Input a string:  ";
17    cin >> s;
18    for (int i=0; i < s.length(); ++i)
19      stackA.pushdown(s[i]);
20    for (int i=0; !stackA.isEmpty(); ++i)
21      s[i] = stackA.popup();
22    cout << "Reversed string: " << s << endl;
23    cout << "---" << endl;
24
25    StackChar stackB;    //デフォルトコンストラクタの利用
26    StackChar stack[3]; //デフォルトコンストラクタの利用
27    for (int i=0; i<3; ++i) {
28      for (int k=0; k<=i; ++k)
29        stack[i].pushdown('*');
30    }
```

```
31    stackB.showContents();
32    for (int i=0; i<3; ++i)
33      stack[i].showContents();
34    cout << "---" << endl;
35
36    cout << "s = ¨ << s << "¨ << endl;
37    StackChar stackC(s);
38    StackChar stackD(stackC);
                                  //コピーコンストラクタの利用
39    stackD.pushdown('*');
40    StackChar stackE = stackD;
                                  //コピーコンストラクタの利用？
41    stackE.popup();
42    stackE.pushdown('#');
43    stackC.showContents();
44    stackD.showContents();
45    stackE.showContents();
```

```
    46    cout << "---" << endl;
    47
    48    cout << "生成されたStackCharインスタンス数 = "
    49         << StackChar::getNumOfInstances() << endl;
    50 }
[motoki@x205a]$  g++ useStackChar.cpp StackChar.cpp
[motoki@x205a]$  ./a.out
Input a string:   abc123
Reversed string: 321cba
---
stack_of_char(id=1,size=100) has 0 elements as follows:
  [Bottom] <--
stack_of_char(id=2,size=100) has 1 elements as follows:
  [Bottom] *  <--
stack_of_char(id=3,size=100) has 2 elements as follows:
  [Bottom] *  *  <--
stack_of_char(id=4,size=100) has 3 elements as follows:
```

```
  [Bottom] *  *  *  <--
---
s = "321cba"
stack_of_char(id=5,size=6) has 6 elements as follows:
  [Bottom] 3  2  1  c  b  a  <--
stack_of_char(id=6,size=16) has 7 elements as follows:
  [Bottom] 3  2  1  c  b  a  *  <--
stack_of_char(id=7,size=16) has 7 elements as follows:
  [Bottom] 3  2  1  c  b  a  #  <--
---
生成されたStackCharインスタンス数 = 8
[motoki@x205a]$
```

3–5 文法的な諸注意(まとめ)

class宣言,struct宣言の大枠:

カプセル化されたソフトウェア部品(オブジェクト)を作り出すために、C++言語ではオブジェクトの枠組み／設計図を定めた次の形の記述

```
class クラス名 {
        (メンバの記述)
};
```

もしくは、

```
struct クラス名 {
        (メンバの記述)
};
```

ここで、

- 「メンバの記述」の部分は、作り出すオブジェクトの構成メンバ(e.g. データ,関数)の記述を並べたもの。　(C言語構造体の書式の拡張)

- メンバの公開／非公開に関しては 、
  - ◇ キーワードが`struct`の場合はメンバは原則公開。一方、
    キーワードが`class`の場合はメンバは原則非公開。
  - ◇ 「メンバの記述」部に
    `private:`という宣言 $\longrightarrow$ 「これ以降のメンバは非公開」。また、
    `public:`という宣言 $\longrightarrow$ 「これ以降のメンバは公開」。

- メンバの宣言に`static`修飾子が付いていれば 、
  そのメンバはクラス全体で共有される静的メンバとして扱われ、
  クラス外では　クラス名`::`メンバ名　という名前で表す。
  - ◇ 静的メンバ変数の初期化 は
    クラス定義の外で`static`修飾子も付けずに

    データ型 クラス名`::`静的メンバ変数名 `=` 初期値を表す式`;`

    という形で行う。

- 関数メンバの記述に関して 、

  ◇ 定義全体をメンバリスト{...} の中に書いた場合、
  暗黙に inline 宣言されたものとして処理される。

  ◇ プロトタイプだけをメンバリスト{...} の中に書いた場合、
  関数定義はメンバリスト{...} の外で行う。
  その際の関数名は

  クラス名::メンバ関数名

コンストラクタ：

オブジェクトの構成メンバとして
コンストラクタと呼ばれる 特殊な役割 の関数を用意できる。
(オブジェクトの初期化)

- オブジェクト生成の際に自動的に呼び出される。
- コンストラクタの関数名は クラス名 である。
- 関数定義の際に「戻り値の型」の部分には何も書かない。
- オブジェクト内の データメンバの初期化 は、
  関数頭部と関数処理本体の間に...

  : メンバ名 (初期値を表す式), ...

  但し、メンバがオブジェクトである場合 には次の形で指定。

  メンバ名 (コンストラクタへの実引数列)

- 引数なしのコンストラクタ ... デフォルトコンストラクタ
  初期設定の指定がない場合に暗黙に呼び出される

- 引数が1個のコンストラクタ
  デフォルトでは
  「引数の型→コンストラクタが扱うオブジェクトの型」
  という型変換に使われる。

  この 型変換を止めたい場合 ⟶ 関数宣言の前に`explicit`修飾子

- コピーコンストラクタ
  ... 同クラスの別インスタンスをコピーして初期化を行うもの

デストラクタ：

オブジェクトの構成メンバとして

デストラクタと呼ばれる 特殊な役割 の関数を用意できる。

(オブジェクト消滅の後始末,

e.g. ヒープ領域から確保した領域の開放)

- ブロック終了等の際に自動的に呼び出される。
- デストラクタの関数名は ~クラス名 である。
- 関数定義の際に「戻り値の型」の部分には何も書かない。