C++プログラミングI

- 第8回：テキスト入力処理
- 担当：二瓶芙巳雄

本日の内容

- テキスト入力処理
- テキストを処理するためのテクニックを学習
  - 単純に `std::cin` を使う（おさらい）
  - `std::cin` をちょっと便利に使う
  - 文字列を `std::cin` かのように使う
  - などなど
- **文字**と**文字列**の違いを意識しましょう
  - 文字: シングルクオーテーション `'` で囲われた，長さが1の値
    - 例）`'a'` `'Z'` `'7'` `'+'` `'\n'`
  - 文字列: ダブルクオーテーション `"` で囲われた，長さが0以上の値
    - 例）`"a"` `"16"` `""` `"c++ programming"` `"hello\n"`

`cin` の基本動作：おさらい

- `cin >> x` ではホワイトスペースを読み飛ばす
  - ホワイトスペース：スペース `' '` ・タブ `'\t'` ・改行 `'\n'` の各文字
- ※このプログラムの入力の対象が **文字 (char)** であることに注意

```cpp
#include <iostream>

int main(int argc, char *argv[]) {
    char ch {};
    while (std::cin >> ch) std::cout << ch;
}
```

```
$ ./a.out
this is a test.
thisisatest.
(Ctrl+D入力でプログラム終了)
```

※ `while( std::cin >> x )`, `for( int x; std::cin >> x; )` のようなパターンについては第4回資料参照

1文字ずつの処理

- `std::noskipws`：ホワイトスペース (ws) をスキップしない指示

```cpp
#include <iostream>

int main(int argc, char *argv[]) {
    char ch {};
    while (std::cin >> std::noskipws >> ch) std::cout << ch;
}
```

```
$ ./a.out
this is a test.
this is a test.
(Ctrl+D入力でプログラム終了)
```

※ 上記プログラムは，ファイルの内容をコピーするプログラムともいえる

入力文字のカウント

- `cin`を行った回数が入力文字数となる
- ※ホワイトスペースを含んだ文字数のカウントが可能

```cpp
int main() {
  int n {0};  // 入力文字数の合計
  for (char ch{}; std::cin >> std::noskipws >> ch;) ++ n;
  std::cout << n << "\n";
}
```

```
$ ./a.out
this is a test.
(Ctrl+Dを入力)
15
```

入力行のカウント

- 改行文字 `'\n'` を数えれば行数が分かる
- 改行文字 `'\n'` の役割は区切ること．`'\n'` もホワイトスペースの一種．

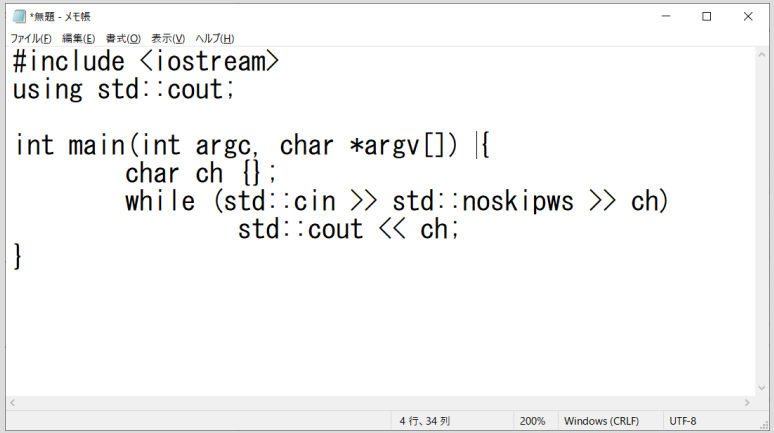
```cpp
int main() {
  int n {0};  // 入力文字数の合計
  for (char ch{}; std::cin >> std::noskipws >> ch;)
    if (ch == '\n') ++ n;
  std::cout << n << "\n";
}
```

```
$ ./a.out
tokyo
musashino
kichijoji-kitamachi
(Ctrl+Dを入力)
3
```

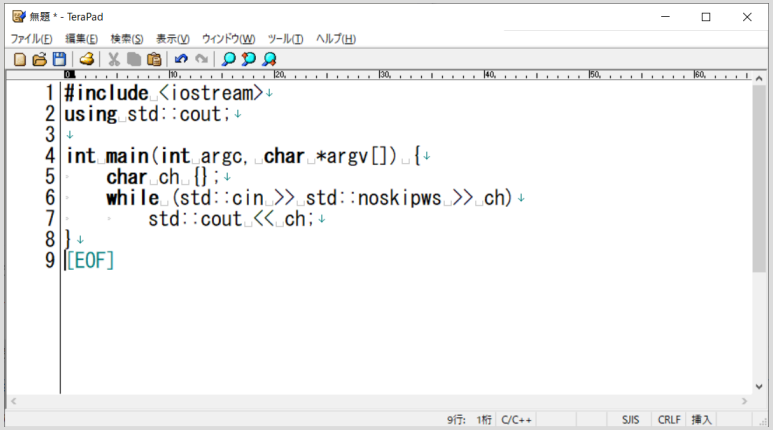
※ `'\n'`（文字型 `char`）`"\n"`（文字列型 `std::string`）に注意

ホワイトスペースの可視化

- テキストエディタによって，制御文字が見える・見えないがある
  - 制御文字が見えるほうが，プログラミング的には便利
- メモ帳



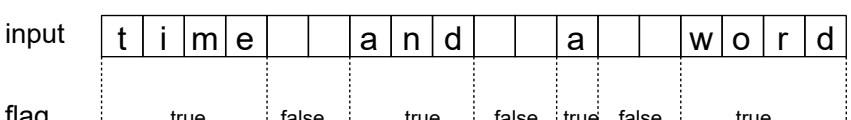
- メモ帳は早めに卒業しましょう
- terapad



- ↓：改行（`\n`），`>`：タブ文字（`\t`），␣：半角スペース
  - ホワイトスペースが別の記号として表現される

単語のカウント

- 英単語：ホワイトスペースで区切られた連続する文字
- `bool` 型の変数 `flag` を使って入力状態を把握
  - ホワイトスペースを読み込むと `false`
  - それ以外の文字を読み込むと `true`
  - ※ 変数名は `flag` 以外でもよい．状態を保存する `bool` 変数は `flag` と名付けられる傾向．
- `false` から `true` に変化したときが単語の先頭文字




```cpp
int main() {
  int n {0};          // 単語の数
  bool flag {false};  // 単語中の文字かどうか
  for (char ch{}; std::cin>> std::noskipws >>ch;) {
    if (ch == ' ' || ch == '\n' || ch == '\t')
      flag = false;
    else if (!flag) {
      flag = true;   // 単語の先頭が見つかった
      ++n;
    }
  }
  std::cout << n << "\n";
}
```

```
$ ./a.out
time  and  a  word
(Ctrl+Dを入力)
4
```

- `flag` 変数は単語の外を表す `false` から始める
- 入力が `ws` 以外の文字で `!flag` ならば単語の先頭
- ※ `std::string` 型を使えばより簡単に実装可能

ASCIIコード

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- 文字を数値に対応させた表（値は覚えなくても良い）．`'a'`から`'z'`と、`'A'`から`'Z'`が連続する数値．
- `'A'`：16進(`0x41`) 10進(`65`)．`'a'`：16進(`0x61`) 10進(`97`)．※ 大文字と小文字の距離はいつも`32`．

	+0x0	+0x1	+0x2	+0x3	+0x4	+0x5	+0x6	+0x7	+0x8	+0x9	+0xA	+0xB	+0xC	+0xD	+0xE	+0xF
0x00	'\0'							'\a'	'\b'	'\t'	'\n'	'\v'	'\f'	'\r'		
0x10																
0x20	' '	'!'	'"'	'#'	'$'	'%'	'&'	'''	'('	')'	'*'	'+'	','	'-'	'.'	'/'
0x30	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'	':'	';'	'<'	'='	'>'	'?'
0x40	'@'	'A'	'B'	'C'	'D'	'E'	'F'	'G'	'H'	'I'	'J'	'K'	'L'	'M'	'N'	'O'
0x50	'P'	'Q'	'R'	'S'	'T'	'U'	'V'	'W'	'X'	'Y'	'Z'	'['	'\'	']'	'^'	'_'
0x60	'`'	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'j'	'k'	'l'	'm'	'n'	'o'
0x70	'p'	'q'	'r'	's'	't'	'u'	'v'	'w'	'x'	'y'	'z'	'{'	'|'	'}'	'~'	

ASCIIコードを仮定した文字の変換

- 単語の先頭文字を大文字にする
- `ch - 'a' + 'A'` に注意
  - 次スライドで説明

```
$ ./a.out
time  and  a  word
Time  And  A  Word
seikei university
Seikei University
(Ctrl+D入力でプログラム終了)
```

```
1  // 各単語の先頭を大文字にするプログラム
2  int main() {
3    bool flag {false};  // 単語中の文字かどうか
4
5    for (char ch{}; std::cin>> std::noskipws >>ch;) {
6      if (ch == ' ' || ch == '\n' || ch == '\t')
7        flag = false;
8      else if (!flag) {
9         flag = true;   // 単語の先頭が見つかった
10        if ( ch>='a' && ch<='z' ) // 小文字ならば
11          ch = ch-'a'+'A';        // 大文字に変換
12     }
13     std::cout << ch;
14   }
15 }
```

文字の変換とは

```
ch = 'd';
ch = ch - 'a' + 'A';
```

- `ch = 'd';` とすると、`ch` は `100`
- `ch - 'a'` は `100 - 97 = 3`
- `'A'` から数えて `3` 番目の文字は `D` である
- `ch - 'a' + 'A'` は `'d'` を `'D'` に変換する

```cpp
int main() {
  std::cout << 'd' << " " << (int)'d' << "\n";
  std::cout << 'a' << " " << (int)'a' << "\n";
  std::cout << (int)('d' - 'a') << "\n";
  std::cout << "\n";

  std::cout << 'A' << " " << (int)('A') << "\n";
  std::cout << (int)('A' + 'd' - 'a') << "\n";
  std::cout << (char)('A' + 'd' - 'a') << "\n";
}
```

```
% ./a.out
d 100
a 97
3

A 65
68
D
```

ライブラリ関数の利用

- `isspace( c )`：引数の`char`型変数`c`がホワイトスペースであれば`true`，でなければ`false`．
- `islower( c )`：引数の`char`型変数`c`が英小文字であるか．小文字なら`true`，でなければ`false`．
- `toupper( c )`：引数の`char`型変数`c`を，大文字に変換したものを返却．

```cpp
// p10のプログラムを改めたもの
#include <cctype>

if main() {
  bool flag {false};
  for (char ch{}; std::cin>> std::noskipws >>ch;) {
    if ( std::isspace(ch) )
      flag = false;
    else if (!flag) {
      flag = true;
      if ( std::islower(ch) )
        ch = std::toupper(ch);
    }
    std::cout << ch;
  }
}
```

```cpp
// p10のプログラム


int main() {
  bool flag {false};
  for (char ch{}; std::cin>> std::noskipws >>ch;) {
    if (ch == ' ' || ch == '\n' || ch == '\t')
      flag = false;
    else if (!flag) {
       flag = true;
       if ( ch>='a' && ch<='z' )
         ch = ch-'a'+'A';
    }
    std::cout << ch;
  }
}
```

ホワイトスペースの明示的な読み飛ばし

```
1  int main() {
2    std::cin >> std::ws;    // ①ホワイトスペースを読み飛ばす
3    for (char ch{}; std::cin>>std::noskipws>>ch;){
4      std::cout << ch;
5      if (ch == '\n') // ②行末だったら...
6        std::cin >> std::ws; // ②行末から次の行頭の空白を読み飛ばす
7    }
8  }
```

- ①：入力の先頭のホワイトスペースを除去
- ②：行末から行頭までのホワイトスペースを除去
  - 改行のみの行も削除される
  - 見えないが行末のスペース文字がなくなる
  - 空白行もなくなる
- 左にそろう

- 入力

```
   abc
bcd
      cde

   defg
   efghi
```

- 出力

```

abc
bcd
cde
defg
efghi
```

`string`による単語のカウント

- `string`型の入力では単語単位に読み取れる
- ホワイトスペースの読み飛ばしも起こる

```
1  int n {0}; // 入力単語数の合計
2  for (std::string s; std::cin>>s; )
3     ++n;
4  std::cout << n << "\n";
```

`string`による大文字変換

- 単語のすべての文字を大文字にする
- 範囲for文の`auto`は`char`
- `ch`はリファレンス`&`のため更新すれば`s`も変化
- ホワイトスペースの個数が無視できる時には有効

```
1  int main() {
2    for (std::string s; std::cin >> s; ) {
3       for (auto& ch : s) {
4        if (std::islower(ch))
5          ch = std::toupper(ch);
6       }
7       std::cout << s <<"\n";
8    }
9  }
```

```
% ./a.out
hello
HELLO
this is a pen
THIS
IS
A
PEN
(Ctrl+D入力でプログラム終了)
```

stringストリーム

- stringストリーム：文字列を `cin` / `cout` のように扱う手段
  - 用途：文字列から整数や実数を取り出す，整数や実数を文字列に変換する，etc…
- ヘッダファイルと型名
  - `<sstream>` をインクルードする
  - `istringstream` 型：文字列から取り出す
  - `ostringstream` 型：文字列に変換する
- 類似機能の関数：
  - `std::to_string( val )` ：引数の整数や実数 `val` を文字列に変換
  - `std::stoi( s )` ：引数の文字列 `s` を `int` に変換
  - `std::stod( s )` ：引数の文字列 `s` を `double` に変換

入力用のstringストリーム

- `std::istringstream` 型．初期値は `string` 変数または文字列リテラル．
  - `std::istringstream` の変数は `>>` でstringストリームからデータを読み取る
  - `std::cin` は `>>` で標準入力ストリームからデータを読み取る（キーボード入力を受け付ける）
  - ※初期値として与えた文字列がキーボードから入力されたものとしてみなす，`std::cin` のようなもの

```cpp
#include <iostream>
#include <sstream>
#include <string>

int main(int argc, char *argv[]) {
  // cin のような入力変換を行う文字列ストリーム
  std::string str = "1 2 3.4 2 4 6.8";
  std::istringstream iss { str };

  int x, y; double z;
  while (iss >> x >> y >> z)
    std::cout << 2*x << ", " << 2*y << ", " << 2*z << "\n";
}
```

```
% ./a.out
2, 4, 6.8
4, 8, 13.6
```

stringストリームを使わないと…

- `std::istringstream`を使う場合

```
1  string str = "1 2 3.4 2 4 6.8";
2  std::istringstream iss{str};
3  int x, y; double z;
4  while (iss >> x >> y >> z)
5    cout << 2*x << ", "
6         << 2*y << ", "
7         << 2*z << "\n";
```

- 便利なので使えるタイミングがあれば使いましょう

- `std::istringstream`を使わない場合

```
1  string str = "1 2 3.4 2 4 6.8";
2  int n_item {0}; string num {""};
3
4  for (size_t i = 0; i < str.size(); i++) {
5    char c { str[i] };
6    num += c;
7
8    if( c == ' ' || i == str.size()-1 ) {
9      if( n_item%3 == 2 )
10       cout << 2 * std::stod(num) << "\n";
11     else
12       cout << 2 * std::stoi(num) << ", ";
13
14     num = ""; n_item++;
15   }
16 }
```

出力用のstringストリーム

- `std::ostringstream`型．`string`へ出力．`.str()`で取り出す
  - `std::ostringstream`の変数は`<<`でstringストリームにデータを送る
  - `std::cout`は`<<`で標準出力ストリームにデータを送る（画面に出力する）

```
1   #include <iostream>
2   #include <sstream>
3   #include <string>
4   using std::string;
5
6   int main(int argc, char *argv[]) {
7      std::ostringstream oss;
8      oss << "abc: "<< 6 << " " << 1.5 << " "; // ストリームにデータを追加
9      string s1 = oss.str();                   // この時点の文字列
10
11     oss <<"xyz: "<< 7 <<" "<< 2.5;           // さらに追加
12     string s2 = oss.str();                   // この時点の文字列
13
14     std::cout << s1 << "\n";
15     std::cout << s2 << "\n";
16  }
```

```
% ./a.out
abc: 6 1.5
abc: 6 1.5 xyz: 7 2.5
```

3種のストリーム

- ストリーム型
  - 入出力：`istream`・`ostream`型(`cin`・`cout`)
  - ファイル：`ifstream`・`ofstream`型
  - `string`：`istringstream`・`ostringstream`型
- ストリーム (stream): データの流れを抽象化する概念
  - 入力ストリーム：プログラムに向かってデータが流れる通路，キーボード入力やファイルからデータを読み込むときに使用
  - 出力ストリーム：プログラムから外部にデータが流れる通路，標準出力（※黒い画面に表示）やファイル出力のときに使用
- 共通の機能
  - `>>` によるストリームからの取り出し
  - `<<` によるストリームへの出力
  - `getline()` 行単位の処理（以降のスライドで説明）

入力ストリーム比較

- 入力ストリーム：プログラムに向かってデータが流れる通路，キーボード入力やファイルからデータを読み込むときに使用
- `>>` によるストリームからの取り出し

```cpp
int main() {
  int x, y; double z;

  // 標準入力ストリーム istream
  std::cin >> x >> y >> z;
  std::cout << 2*x << " " << 2*y << " " << 2*z << "\n";

  // ファイル入力ストリーム ifstream
  std::ifstream fin{ "input.txt" }; // エラー処理省略
  fin >> x >> y >> z;
  std::cout << 2*x << " " << 2*y << " " << 2*z << "\n";

   // string入力ストリーム istringstream
  std::istringstream iss{ "7 8 9.9" };
  iss >> x >> y >> z;
  std::cout << 2*x << " " << 2*y << " " << 2*z << "\n";
}
```

```
% cat input.txt
4 5 6.6
% ./a.out
1 2 3.3     <- cinに対する手入力
2 4 6.6     <- cinで入力した値の2倍を表示
8 10 13.2   <- finで入力した値の2倍を表示
14 16 19.8  <- issで入力した値の2倍を表示
```

※ `fin` や `iss` は，`std::cin` と異なり，自分で宣言する変数なので，好きな名前にしてよい．例えば，`fin1`，`fin2`，`fin_file1`，`in_file`，`in_values`，etc…

出力ストリーム比較

- 出力ストリーム：プログラムから外部にデータが流れる通路，標準出力（※黒い画面に表示）やファイル出力のときに使用
- `<<` によるストリームへの出力

```cpp
int main() {
  int x {1}, y {2}; double z {3.3};

  // 標準出力ストリーム ostream
  std::cout << 2*x << " " << 2*y << " " << 2*z << "\n";

  // ファイル出力ストリーム ofstream
  std::ofstream fout{ "output.txt" }; // エラー処理省略
  std::fout << 3*x << " " << 3*y << " " << 3*z << "\n";

  // string出力ストリーム ostringstream
  std::ostringstream oss;
  oss << 5*x << " " << 5*y << " " << 5*z << "\n";
  std::cout << oss.str();
}
```

```
% ./a.out
2 4 6.6     <- coutの部分
5 10 16.5   <- ossの部分
% cat output.txt
3 6 9.9
```

※ `fout` や `oss` は，`std::cout` と異なり，自分で宣言する変数なので，好きな名前にしてよい．例えば，`fout1`，`fout2`，`fout_file1`，`out_file`，`out_values`，etc…

1行ずつの処理

- 関数 `std::getline( stream, string )` の利用
  - 第1引数: ストリーム型の変数（リファレンス）
  - 第2引数: `string` 変数（リファレンス）
  - 戻り値: 第1引数に指定したストリームそのもの（ループとの相性が良い）
- 行末の `'\n'` は読み捨てられる

```
hello
this is a pen
seikei university
cpp programming
(Ctrl+Dを入力)
number of lines: 4
longest line:3
----->seikei university
```

```cpp
int main(int argc, char* argv[]) {
  int num{0};           // 読み込んだ行の数
  int maxline_num{0};   // もっとも長い行の行番号
  std::string maxline;  // もっとも長い行の内容

  // 1 行ずつ読み込んで処理する
  for( std::string line; std::getline(std::cin, line); ) {
    ++num;  // 行数のカウント
    if (maxline.size() < line.size()) {
      maxline = line;
      maxline_num = num;
    }
  }
  std::cout << "number of lines: " << num << "\n"
            << "longest line:" << maxline_num << "\n"
            << "----->" << maxline << "\n";
}
```

行の途中から行末までの入力

- `getline()` は入力位置から改行文字 `'\n'` まで読み込む
- 改行を取り除いて `string` 変数に設定
  - `in >> x >> y` で行の途中まで読み込む
  - `getline(in, s)` で行の残りを読み込む

```cpp
int main(int argc, char* argv[]) {
  std::ifstream in("input.txt");
  if (!in) { return 1; }

  int x, y; std::string s;
  while ( in >> x >> y && std::getline(in, s) )
    std::cout << x * y << s << "\n";
}
```

```
$ cat input.txt
32 300 White Chocolate
42 430 Orange Cookie
53 380 Lemon Macaroons
$ ./a.out
9600 White Chocolate
18060 Orange Cookie
20140 Lemon Macaroons
```

区切り文字ごとの処理

- `getline()` は `'\n'` を区切り文字にしている
- `getline()` の第3引数の指定で，区切り文字を別の文字に変更できる

```
1  int main(int argc, char* argv[]) {
2    std::string line{"aaa,bbb,ccc,ddd"};
3    std::istringstream iss(line);
4
5    for( std::string s; std::getline(iss, s, ','); ) // 第3引数が増えた
6      std::cout << s << "\n";
7  }
```

```
$ ./a.out
aaa
bbb
ccc
ddd
```

CSVファイルとは

- **C**haractor (or **C**omma) **S**eparated **V**alues
- 表計算ソフトやDBデータのテキスト保存形式
- ※ テキストファイルのうち，カンマ `,` などで構造化されたものを，csvファイルと呼ぶ
- デファクトスタンダード
- たくさんの変種
- 共通部分
  - レコードと呼ぶ関係するデータを 1 行ごとにまとめる
  - レコードが複数の値を持つならばカンマ文字で区切って値を並べる

```
White Chocolate,32,300
Orange Cookie,42,430
Lemon Macaroons,53,380
```

CSVファイルの処理

- 1行ずつ取り出し、カンマ区切りで取り出す
- `input.csv` は `, `（カンマ+スペース）区切り
- `getline()` は `','` 区切り
  - `v[0] = "White Chocolate"`
  - `v[1] = " 32"`
  - `v[2] = " 300"`

```
$ cat input.csv
White Chocolate, 32, 300
Orange Cookie, 42, 430
Lemon Macaroons, 53, 380
$ ./a.out
White Chocolate: 9600
Orange Cookie: 18060
Lemon Macaroons: 20140
```

```cpp
int main(int argc, char *argv[]) {
  std::ifstream ifs("input.csv");
  if (!ifs) { /*エラー処理*/ }

  for( string line; getline(ifs, line); ) {
    std::vector<string> v;
    std::istringstream iss(line);

    for( string s; getline(iss, s, ','); )
      v.push_back(s);

    // csvデータは行ごとに3個あるはず，例外のためのエラー処理
    if (v.size() < 3) {
      cout << "line error\n";
      continue;
    }

    int num {std::stoi(v[1])};
    int price {std::stoi(v[2])};
    cout << v[0] <<": "<< num*price << "\n";
  }
}
```