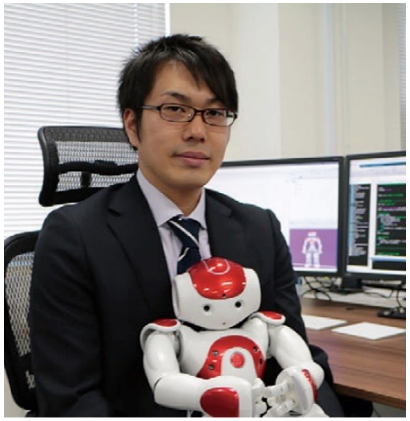
担当：
知能情報学分野
増山 直輝



HP: https://masuyama-lab.github.io/index_ja.html

情報工学演習I

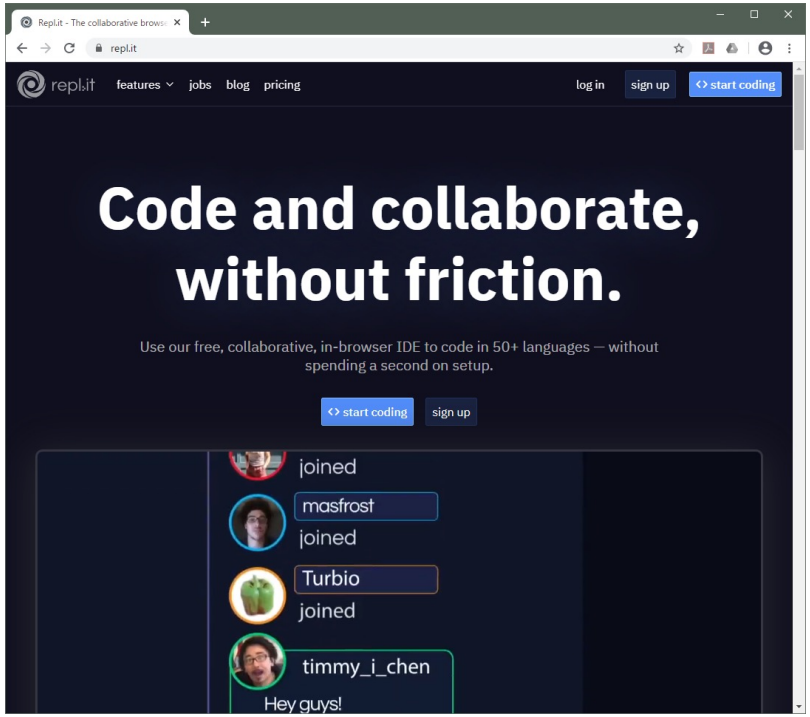
C++の演習1（Hello C++）

開発環境

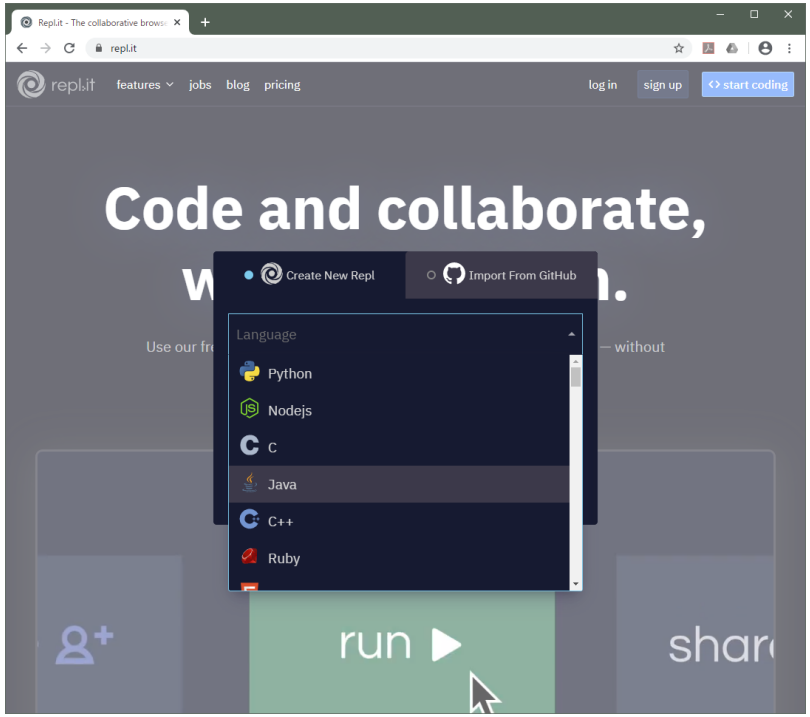
- **各自のPCで導入が困難な可能性があり，また，各自で環境設定が異なると採点時に問題が生じるため，以下のWebサービスを開発環境として指定します.**
- repl.it
  https://repl.it/

Eclipse を導入して開発を行っても構いませんが,
最終的な動作テストは repl.it で行うこと！
※採点は repl.it 上で行われるため

Repl.it による開発



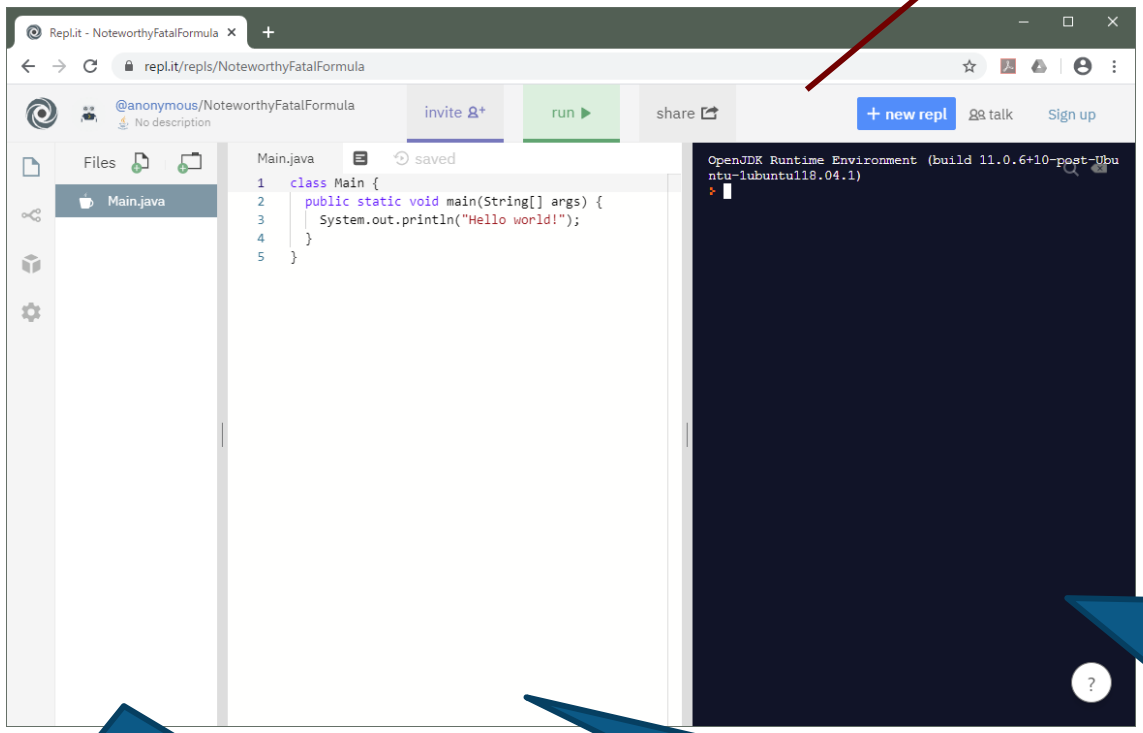
ブラウザで
Repl.it (https://repl.it/)
にアクセスする.



サイト右上の「start coding」をクリックし，Language で C++ を選択し，「Create New Repl」をクリックする.

Repl.it による開発

C++開発画面に移る.



無料版の Repl.it では
ランダムに生成された
URL にアクセスすると
誰でもコードを閲覧できる
設定になっているので注意！
開発終了の際はファイルを
消去しておくことを
推奨します.

実行画面：
ページ上部の「run」ボタンを
押すと，「main.cpp」が
コンパイルされ実行される.

ファイル操作欄：
ファイルの追加,
アップロード,
ダウンロードが可能

ファイル編集画面：
デフォルトで
Hello world の
コードが入っている.

今日の内容

- Hello C++
- newとdelete
  （配列の動的メモリ確保）
- 関数のオーバロード
- デフォルト引数
- インライン関数

Hello C++

C++

- C言語の拡張
  - クラス、仮想関数、多重定義、多重継承、テンプレート、例外処理などの導入
  - C++の当初の名前は「C with Classes」だった

- C++は、基本的にC言語の上位互換
  - ほとんどC言語で、一部だけC++のソースコードも書ける
  - C++で導入された拡張をうまく使えば、より便利にわかりやすく！
    - より直感的に
    - バグを減らす工夫
    - ソースコードの再利用可能性

C++のお約束

- ソースコードのファイル名
  - .ccまたは.cppで終わる
- コンパイル
  - gccの代わりにg++ を使う

- 例：g++ ex0_hello_c++.cpp

Hello world of C++

ex0_hello_c++.cpp

```
#include <iostream> // Cのstdio.hに相当
using namespace std; // 名前空間stdを使う。とりあえずお約束

int main() {
  cout << "Hello C++!" << endl; // 出力の構文。 coutは標準出力

  int n; // 宣言は、使う前ならプログラムのどこにあってもよい
  cout << "n = ";
  cin >> n; // 入力の構文。 cinは標準入力
  int s = 0;
  for (int i=1; i<=n; i++) s+=i;
  cout << "sum(1.." << n << ") = " << s << endl;
  return 0;
}
```

C++の流儀１

- ヘッダのinclude
  - C++特有の関数を使いたければ、C++用のヘッダをincludeする

  例：#include <iostream>
  - Cの関数を使う場合は、C用のヘッダもincludeする

  例：#include <iostream>
  #include <stdio.h>

- 変数の宣言
  - C言語の場合：関数やブロックの最初でないと宣言できなかった
  - C++の場合：どこでも可能

C++の流儀２

最初はusing namespace std;
をお約束としてつけておこう

- 名前空間（Namespace）
  - 同名の変数や関数を区別する工夫
  - 標準関数は、名前空間stdに属している
    - coutの正式名称はstd::cout
    - いちいちstd::を付けるのが面倒な場合は、
      using namespace std;
      を付ければ省略可能

2つの京橋

東京の京橋

大阪の京橋

「これから話すことは全部東京のことだから」と宣言すれば、いちいち「東京の京橋と言わなくても一意に意味が通る」

- コメント
  - //が使える
  - Cスタイルの/* */も使える

C++の流儀３

- C++の入出力
  - 出力
    - 標準出力：cout
    - 標準エラー出力：cerr
    - （改行：endl）

    例：cout << “hello” << endl;
  - 入力
    - 標準入力：cin

    例：int n; cin >> n;

標準出力と標準エラー出力の違いは、すぐに出力されるかどうか
（標準エラー出力はすぐに出力されるが、標準出力はバッファが一杯になるまで出力されない）

endlの代わりに¥nを使うことも出来る

coutやcinは変数の型を指定しなくてもいい

Cスタイルのprintf、scanfなども使える

C++の流儀３（続き）

サンプルファイルには入っていない

- coutでのフォーマット指定
  - 出力する文字幅を指定する

```
#include <iomanip>
// 値 value を 10 文字分の表示スペースに出力する
cout << setw(10) << value;
```

  - 表示位置を指定する

```
// 以降の数値出力を、表示幅に対して右寄せで出力する
cout.setf(ios::right, ios::adjustfield);
// 以降の数値出力を、表示幅に対して左寄せで出力する
cout.setf(ios::left, ios::adjustfield);
```

参考：http://program.station.ez-net.jp/special/handbook/cpp/stream/format.asp

newとdelete
（配列の動的メモリ確保）

配列の動的メモリ確保・解放

- C言語の場合
  - mallocでメモリの確保
  - freeでメモリの解放

メモリの容量をsizeof演算子で計算が必要，かつキャストも必要。

ex1_malloc.c

```
#include <stdlib.h> /* mallocを使うときのお約束 */

int main() {
  int *n;
  n = (int *)malloc(sizeof(int)*10); /* int型の変数10個分のメモリ確保 */
  free(n); /* 確保したメモリの開放 */
  return(0);
}
```

配列の動的メモリ確保・解放

- C++の場合
  - newでメモリの確保
    - 配列の大きさが１のときは、
      int *n=new int;
      としても良い
  - delete []でメモリの解放

sizeof演算子もキャストも不要

重要：配列のメモリ確保のときは、[]が必要

ex2_new_array.cpp

```
int main() {
  int *n;
  n = new int[10]; // int型の変数10個分のメモリ確保
  delete [] n; // 確保したメモリの開放
  return(0);
}
```

関数のオーバロード

関数のオーバーロード

- C++では、同じ名前を持つ関数を複数定義できるようになった

```
unsigned int hiku(unsigned int x, unsigned int y) {
```

```
int hiku(int x, int y) {
```

引数で区別

- 引数が異なる同名の関数を複数定義することを、**関数のオーバーロード**と呼ぶ
- C言語の場合は、同じ名前の関数を定義すると怒られる

関数のオーバーロード

ex3_overload.cpp

```
#include <iostream>
using namespace std;

unsigned int
hiku(unsigned int x,
unsigned int y) {
  return x-y;
}

int hiku(int x, int y) {
  return x-y;
}

int main() {
  unsigned int a=60;
  unsigned int b=200;

  cout << "In case of unsigned int: "
<< hiku(a, b) << endl;
  cout << "In case of int: " <<
hiku((int)a, (int)b) << endl;

  return 0;
}
```

unsigned int型

int型の定義

unsigned int型の呼び出し

int型の呼び出し

デフォルト引数

球の体積の計算

ex4_default_arg.cpp

デフォルト引数

```
#include <iostream>
#include <math.h> // πを使うため
using namespace std;

double sph_vol(double r=1.0) { // 球の体積
  return 4.0 / 3.0 * M_PI * r * r * r;
}

int main() {
  cout << "With argument (r=2.0): " << sph_vol(2.0) << endl;
  cout << "With argument (r=1.0): " << sph_vol(1.0) << endl;
  cout << "Without argument: " << sph_vol() << endl;

  return 0;
}
```

デフォルト引数

- 引数を指定しないときの引数の値が指定できる

```
double sph_vol(double r=1.0) { // 球の体積
```

引数を指定した場合

```
  cout << "With argument (r=2.0): " << sph_vol(2.0) << endl;
  cout << "With argument (r=1.0): " << sph_vol(1.0) << endl;
  cout << "Without argument: " << sph_vol() << endl;
```

デフォルト引数

- デフォルト引数の後に普通の引数は取れない

```
double sph_vol(double r=1.0, int n) { // 駄目な例
```

インライン関数

インライン関数

- 関数呼び出しで生じるオーバーヘッドを回避する方法
  - 関数呼び出しは処理のオーバーヘッドが生じる
    - 引数があれば余計にオーバーヘッドが大きくなる
  - インライン関数のコードは、呼び出し元のコードに埋め込まれる
  - C言語のマクロに似ている

- インライン関数とは処理の高速化で使用される手法の一つ

インライン関数

実際にインライン化されるかどうかはコンパイラ依存

- 宣言方法
  - 関数の宣言の前にinlineを入れるだけ

```
int tasu(int x, int y) { // 足し算する関数
  return x+y;
}

inline int tasu_inline(int x, int y) { // 足し算する関数(インライン版)
  return x+y;
}
```

インライン関数

ex5_inline1.cpp

```
#include <time.h> // 時間測定用
#include <iostream>
using namespace std;

int tasu(int x, int y) { // 足し算する関数
  return x+y;
}

inline int tasu_inline(int x, int y) {
// 足し算する関数(インライン版)
  return x+y;
}

int main() {
  int sum;
  clock_t start, end;
// 処理の開始時間と終了時間
  sum=0;
  start = clock(); // 開始時間測定
  for(int i=0; i<214748364; i++) {
    sum = tasu(sum,i);
  }
  end = clock(); // 終了時間測定
  cout << "Computation time without
inline: " << end-start << endl;

  sum=0;
  start = clock(); // 開始時間測定
  for(int i=0; i<214748364; i++) {
    sum = tasu_inline(sum,i);
  }
  end = clock(); // 終了時間測定
  cout << "Computation time with inline: "
<< end-start << endl;

  return 0;
}
```