C++プログラミングI

第2回

成蹊大学理工学部

bool 型と論理式

bool 型の基本

- bool 型の値は true と false
- C 言語との互換性のために整数も扱う
  - 0 が false, 0 以外が true
  - cout の出力, 初期化に注意

```
bool x{true}, y{}, t{x};
std::cout << x <<" "<< y <<"\n";
// 「1 0」と表示される

x = y;  y = t;
std::cout << std::boolalpha
<< x <<" "<< y <<"\n";
// 「false true」と表示される

bool a{1}, b{0};
std::cout << a <<" "<< b <<"\n";
// 「true false」と表示される

// bool c{2}; // error
```

比較演算子

- 2 つの値を比べる演算子
  - 二項演算子：オペランド（被演算子）が二つ
- 演算結果は `bool` 型の値

演算子	意味
<	小なり
<=	小なりイコール
>	大なり
>=	大なりイコール
==	等しい
!=	等しくない

bool 変数と比較演算

- 比較の結果は bool 型変数の初期化や代入に利用できる
- 実数値に対する==, !=の使用は注意が必要
- char 型でも大小比較が可能 (ASCII コード順, 辞書順)

```
int i{3}, j{4};
bool x{3 < 4}, y{};    // x の初期値
y = i > j;             // y の値
std::cout << std::boolalpha
          << x <<" "<< y <<"\n";

double z = 0.1 + 0.2;
std::cout << (z==0.3) <<"\n"; // false!

char a{'m'}, b{'n'};
std::cout <<(a == b)<<" "<<(a <= b)<<"\n";
```

論理演算子

- 論理積：&&, 論理和：||, 論理否定：!
- 複数の条件を組み合わせるのに利用する

A	B	A && B	A || B	!A
true	true	**true**	true	false
true	false	false	true	false
false	true	false	true	false
false	false	false	**false**	true

- &&と||は比較演算子よりも優先度が低い

```
int x {3};
std::cout << std::boolalpha << x <<":"
          << (1 <= x    && x <= 3) <<" "
          << (x <= -50 || x >= 100) <<"\n";
```

短絡評価

- &&と||の特別ルール
- 一つ目のオペランドで演算結果が決まる
- 二つ目のオペランドは評価（計算）しない
- 二つ目の演算が副作用を持つ場合に重要
  - 入力や代入も結果を持つ式であることに注意

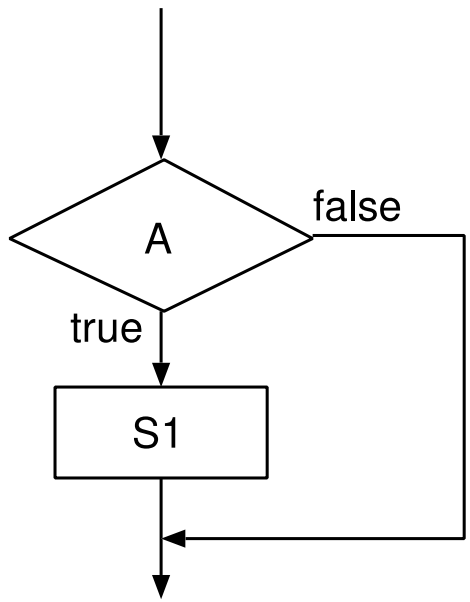
```
x >= 0 || std::cout <<"の入力エラー x\n";
y >  0 || std::cout <<"の入力エラー y\n";
y != 0 && std::cout <<"x%y = "<< x%y <<"\n";

std::cin >> x || std::cout<<"error\n";
std::cin >> y || (y = 10);
```

条件文

if 文

- A が true ならば S1 を行う
  - 条件 A は bool 型の式
  - 文 S1 は一つの実行文
- 注意事項
  - 中括弧を使った複文の利用
  - if 文を 1 行に書いても良い
  - インデントの位置決めも任意

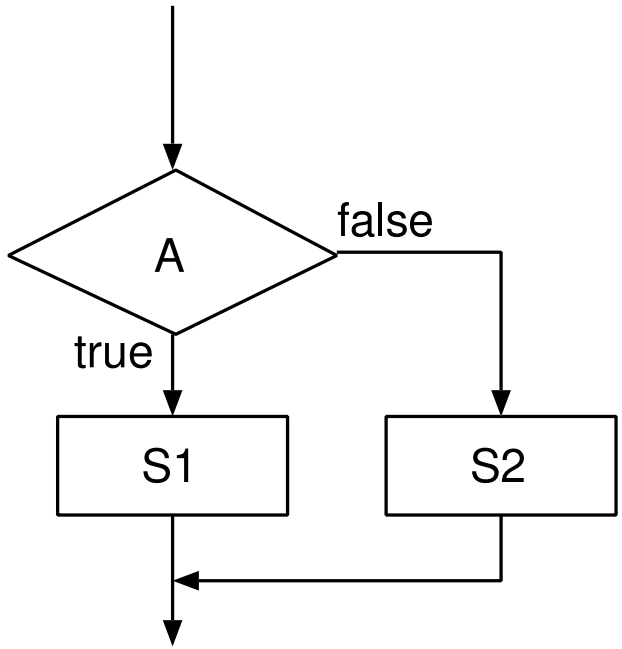
```
if (条件 A)
  文 S1
```



if-else 文

- if 文に加えて条件が false の場合の処理がある
- S1 と S2 はどちらも複文で指定できる

```
if (条件 A)
  文 S1
else
  文 S2
```



if 文 if-else 文の例

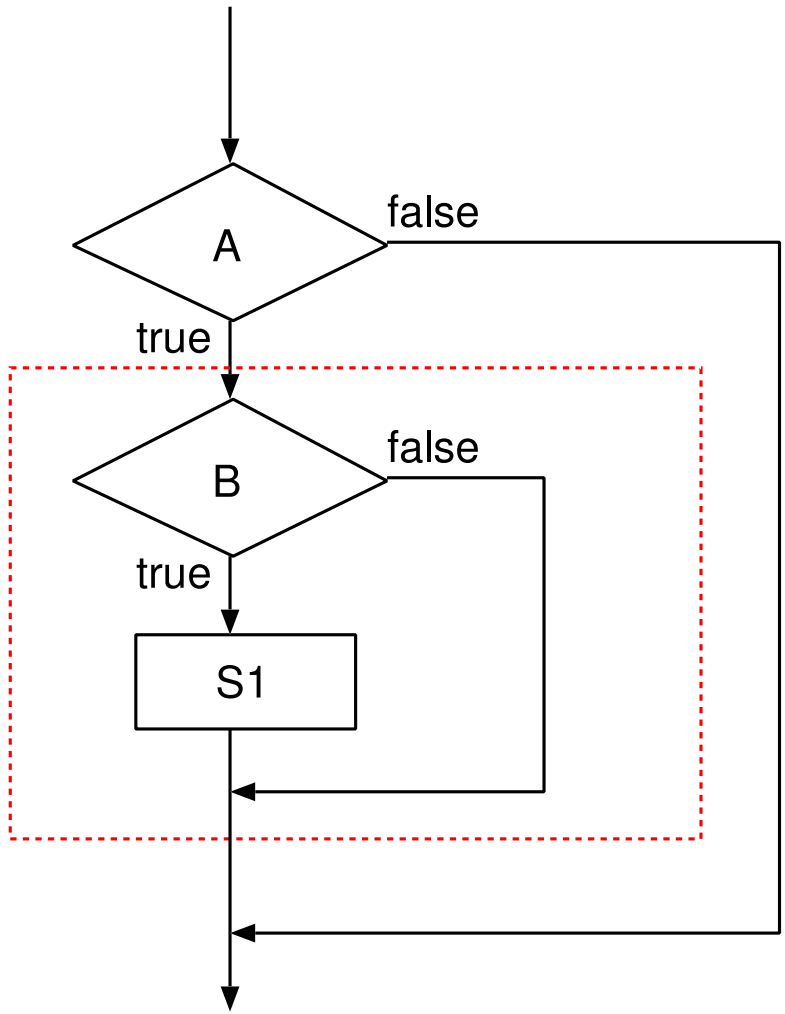
- 複文を使うかどうかを検討する
- 全体が短ければ 1 行に書く方法もある

```
double x {};
std::cin >> x;

if (x < 0.0) x = -x;

if (x < 1.0) x += 10.0;
else         x *= 10.0;

if (x <= 10.0) {
   std::cout << "input error";
   x = 10.0;
} else {
   std::cout << "Result is "<< x <<"\n";
}
```

入れ子の if 文

- `if` 文の中に別の `if` 文がある
- 条件 `A` と `B` の組み合わせは論理積と同じ

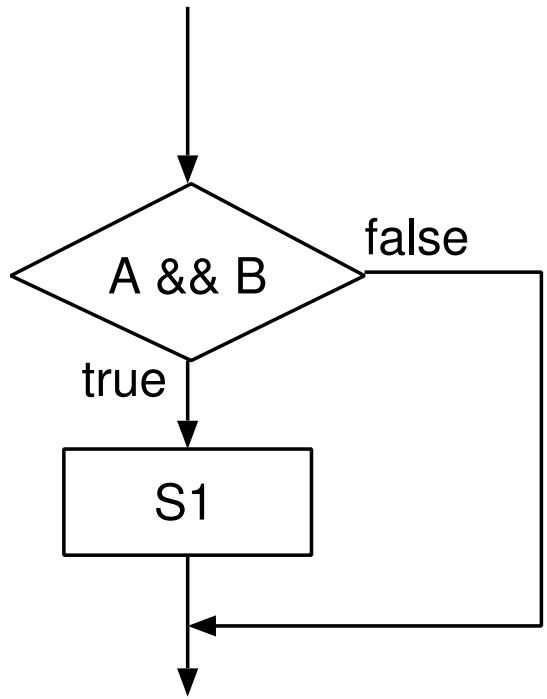
```
if (条件 A) {
  if (条件 B) {
    文 S1
  }
}
```



論理積の利用

- 入れ子の `if` 文は `&&`演算子の短絡評価とマッチする
- 簡潔に記述できる

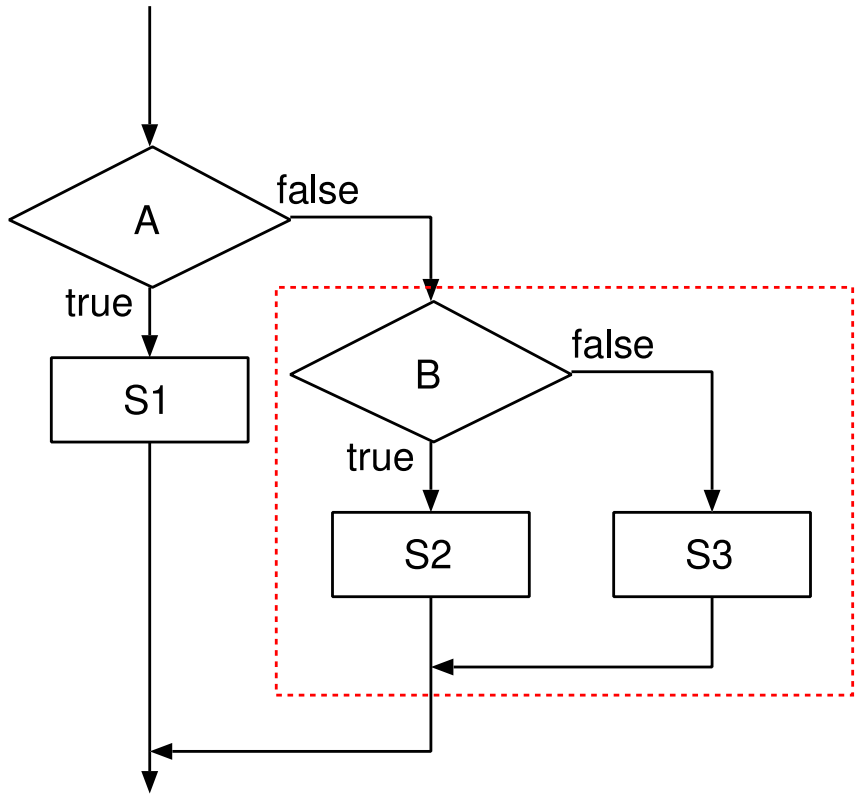
```
if (条件 A && 条件 B) {
    文 S1
}
```



else 部の if-else 文

- インデントが深くなる

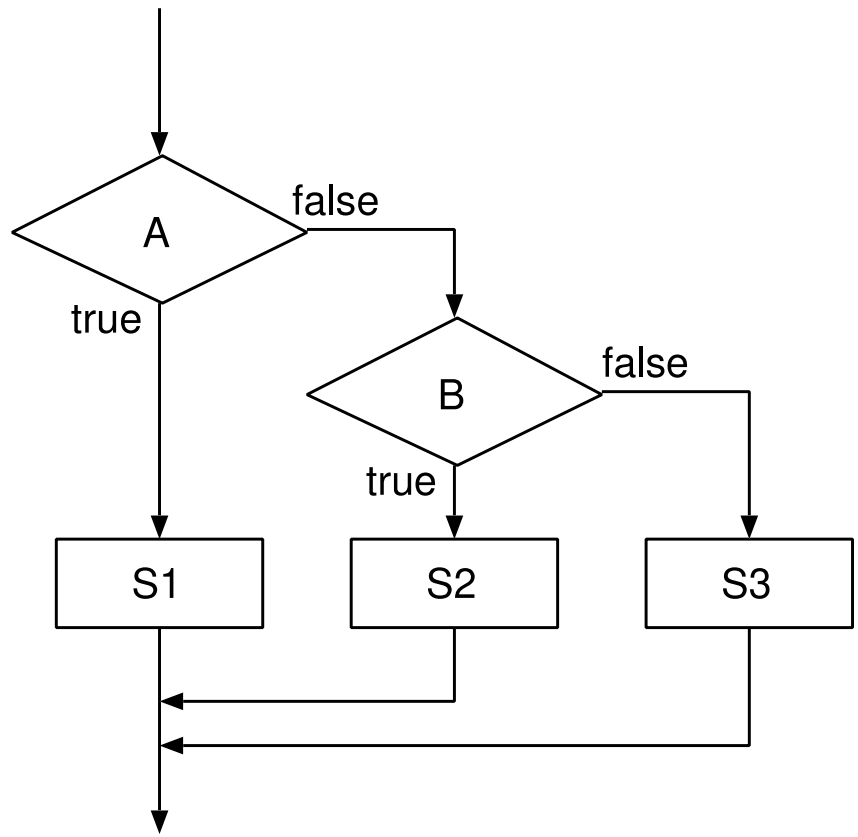
```
if (条件 A) {
    文 S1
} else {
    if (条件 B) {
        文 S2
    } else {
        文 S3
    }
}
```



3個以上の実行文の選択

- 条件に応じた3個以上の実行文の選択
- else部の中括弧を書かずにif-elseを続ける
- else-if文と呼ぶ人もいる
  - 非公式な呼び名なので注意する

```
if (条件A) {
    文S1
} else if (条件B) {
    文S2
} else {
    文S3
}
```



その他

bool 型の定数比較

- bool 型の変数を true/false と比較しない
- 変数そのものが true/false の値である

```
bool x{true};

if (x == true) // 冗長
  std::cout << "redundant";

if (x)         // 簡潔
  std::cout << "simple";
```

ぶら下がり else 問題

- else 部はもっとも近い if に対応する
- インデントで混乱しないように気をつける
- 不安ならば中括弧を使い複文とする
  - ただし行数が増えることがある

```
// a) どこか混乱している
if (x <= 10)
    if (x == 10) std::cout << "Equal to 10";
else
    std::cout << "Larger than 10?";

// b) これと a) は同じではない。
if (x <= 10) {
    if (x == 10) std::cout << "Equal to 10\n";
} else
    std::cout << "Larger than 10\n";
```

三項演算子

- 条件に応じて二択の値を選ぶ
- 式 1 と式 2 の結果は同じ型

条件 A ? 式 1 : 式 2

```
x = (a < 0) ? (-a * 10) : (a * 20);または



int x {(a < 0) ? (-a * 10) : (a * 20)};
```

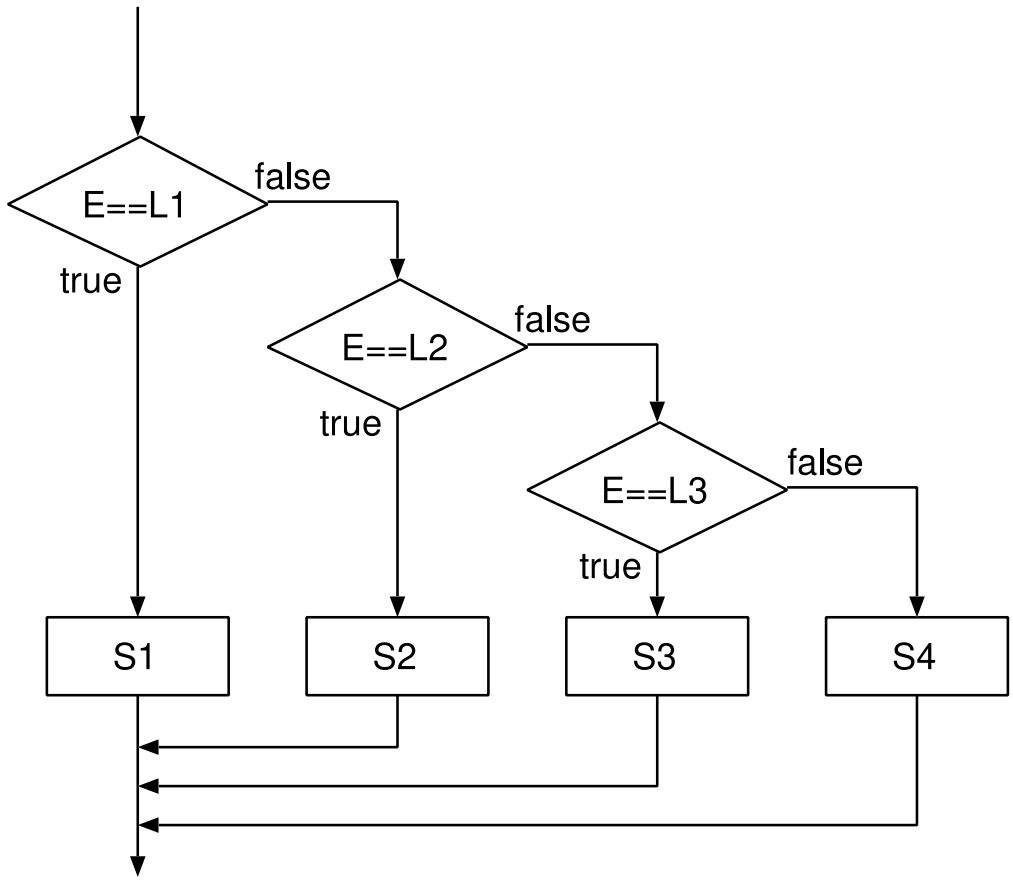
- 3 個以上の選択肢にも有用

```
ch = (a == 1) ? 'a':
     (a == 2) ? 'b':
     (a == 3) ? 'c':
     (a == 4) ? 'd': 'z';
```

switch 文

- 整数値に応じて処理を選ぶ

```
switch (条件式 E) {
case ラベル L1:
    文 S1;
    break;
case ラベル L2:
    文 S2;
    break;
case ラベル L3:
    文 S3;
    break;
default:
    文 S4;
}
```



switch 文の例

- case ラベルにはコンパイル時の定数を使う
- default ラベルは他のラベルにマッチしない場合
- break を忘れないようにする

```
switch (ch) {
case 'a':
    std::cout << "apple\n";
    ++ apple;
    break;
case 'b':
    std::cout << "banana\n";
    ++ banana;
    break;
case 'c':
    std::cout << "candy\n";
    ++ candy;
    break;
default:
    std::cout << "unknown\n";
}
```

break をあえて書かない例

- 条件となる値を並べるだけ
- 長い変数名も一度だけの指定で良い
- break 文と case ラベルの関係
  - switch 文を終わらせる役割
  - case ラベルは実行文を開始する場所を示すだけ

```
int longnamevariable {};
std::cin >> longnamevariable;
switch (longnamevariable) {
case 1: case 2: case 3:
case 5: case 7: case 11:
    std::cout << " あたりです\n";
    break;
default:
    std::cout << " はずれです\n";
}
```

switch 文の default

- default ラベルが最後ならば break は不要
- 逆に、break を付けるならば途中に指定しても良い

```
int n {};
std::cin >> n;
switch (n) {
case  1:
    std::cout << "を検出 1\n";
    break;
default:
    std::cout << " 不正な値\n";
    break;
case -1:
    std::cout << "を検出-1\n";
    break;
}
```

switch 文と変数宣言

- case ラベルの後で変数宣言が必要ならば複文とする

```
switch (a+b) {
case 3:
     {  // 複文にする
        int x {};  // 新たな変数宣言
        ....       // x を使った処理
        break;
     }
case 4:
...
```