# プログラム言語C

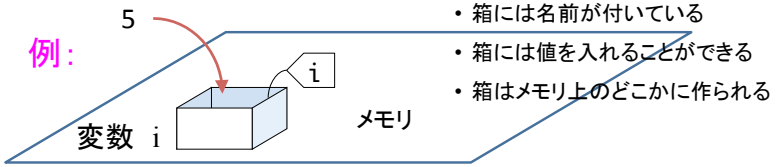
算術式の計算
条件判定
繰り返し

# 今日の内容

- 変数、代入
- 関数
  - ✓ システム関数
  - ✓ 標準出力関数 printf()
  - ✓ 標準入力関数 scanf()
- 制御構造
- if - else 文
  - ✓ else - if による多分岐
  - ✓ if 文の入れ子
- 論理演算
- switch 文
- 定回反復と必要性
  - ✓ for 文
  - ✓ while 文
  - ✓ do-while 文
  - ✓ ループの途中での実行の制御

# 変数

- 変数
  - データ(数値や文字)を入れるもの(箱)
- 変数名
  - 英数字かアンダーバー(_)で作られる
  - 数字は最初の文字に使えない
  - 大文字と小文字を区別する




# 数の表現

1ビット(bit) = 0か1かという2つの状態を表す情報量
= 0か1かが入る箱
1バイト(byte) = 8ビット

(アルファベット1文字は1バイトで表現可能)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

$= 2^2 + 2^0 = 5$

$2^7$ $2^6$ $2^5$ $2^4$ $2^3$ $2^2$ $2^1$ $2^0$

8ビットで表すことの出来る数字は？

$0 \sim 2^8 - 1 = 255$

# もっと大きな数は？

- ビットをたくさん並べると大きな数を表せるようになる
- 負の数も表さなければならない

1バイト（8ビット） → $-128 \sim 127$

2バイト（16ビット） → $-32{,}768 \sim 32{,}767$

4バイト（32ビット） → $-2{,}147{,}483{,}648 \sim 2{,}147{,}483{,}647$

機械で扱うので，あらかじめビット数を
決めておく必要がある．

# 変数宣言

- 宣言
  - コンパイラにユーザーが作った名前をどのように解釈するのかを伝えるもの
  - 変数を使う前に必ず宣言しなければならない
  - 値の種類を教える必要がある（型）

箱の大きさが異なる

| 型名 | 種類 | バイト数 |
|---|---|---|
| char | 文字 | 1バイト |
| short | 整数 | 2バイト |
| int | 整数 | 2または4バイト |
| long | 整数 | 4または8バイト |
| unsigned | 符号なし整数（組合せ） | |
| float | 単精度浮動小数点数 | 4バイト |
| double | 倍精度浮動少数点数 | 8バイト |

変数の上限下限はヘッダファイル `limit.h` に記述されている

# 整数変数

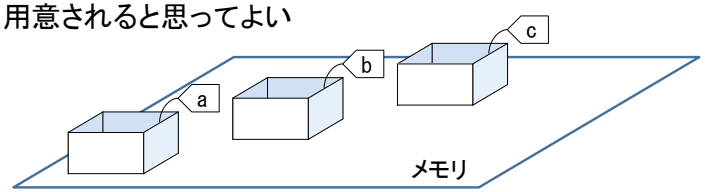
変数宣言　`int a, b, c;`

コンピュータの中で int 型の変数とは，
整数 (..., -2, -1, 0, 1, 2, ...) を蓄えておく32ビットの箱
下の範囲の整数を1つだけ箱の中に入れることができる

$-2^{31} \sim 2^{31}-1$

$-2{,}147{,}483{,}648 \sim 2{,}147{,}483{,}647$

上の変数宣言をすることで，上の図のような3つの箱が
用意されると思ってよい




# 代入

- 変数に数値を格納すること
- C言語ではイコール(=)を使用するが，左辺と右辺が等しいという意味ではない
- 変数の型と同じ型のデータを格納する

変数 = 式;

- 左辺は変数を1つだけ書く
- 値を格納する変数

# 代入式

変数 = 式;　右辺はリテラル，演算式を記入できる

式を計算した結果（値）を左辺の変数に代入する

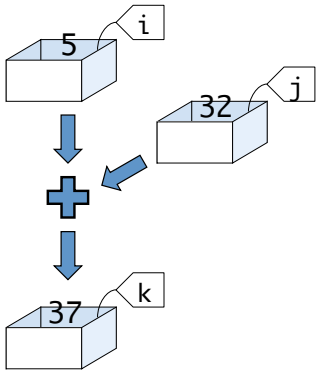
リテラルの代入

```
i = 5;
```
← 値5が代入される

```
j = 0x20;
```
← 値$32_{(10)}$が代入される

演算結果の代入

```
k = i + j;
```
← 足し算を計算した値が代入される




# リテラル

- "Hello, world"（文字列）や 5（整数）などの具体的なデータをリテラルと言う

| | | |
|---|---|---|
| 文字リテラル | 'A'　'a' | … |
| 文字列リテラル | "Hello, world" | … |
| 整数リテラル | 5 | （10進数） |
| | 074 | （8進数） |
| | 0x2f | （16進数）… |
| 浮動小数点数リテラル | 5.0　5.　0.2　.2 | … |

ダブルクォート

浮動小数点数リテラルは指定のない限りdouble型となる

```
3.141592　（double型）
3.1415f　（float型）
```

# 代入式の文

- 代入式をプログラム中に書くときには，その後ろにセミコロンをつけて文にする

代入式;　代入式の文

例

```
i = 5;
j = 0x20;
```
リテラルの代入

```
k = i + j;
```
演算式を計算した値を変数に代入
この場合は 5 + 32 で 37 を変数に代入する

```
f0 = (float)(j / i);
f1 = (float) j / (float)i;
```
キャスト
直後（右側）にある式，変数，リテラルの型を変更する

# 関数（ライブラリ関数）

- 数学的な関数はライブラリ関数として用意されている
  - ヘッダファイル math.h を include すると使用可
  - 関数　sin(), cos(), tan(), ln() など
  - 括弧の中に引数を書ける

関数名

使用例：

```
x = r * cos(theta);
```

引数：
- 関数に与えるパラメータ
- リテラル，変数，式などを書くことができる
- 変数の値が関数に伝わる
- 式を計算した結果が関数に伝わる

# 標準出力関数 printf()

- 変数の値をコンソールに出力する際には 標準出力関数(printf 関数)を使用する
- プログラム中では下のように記述

  printf(第１引数,第２引数,第３引数,…);

- printf では第1引数(出力書式文字列)を使って引数の数とその型が決められる
- 2つ目以降の引数として変数(もしくは式)を書く

# 書式付出力

出力変換仕様
%文字幅，精度，変換指定子

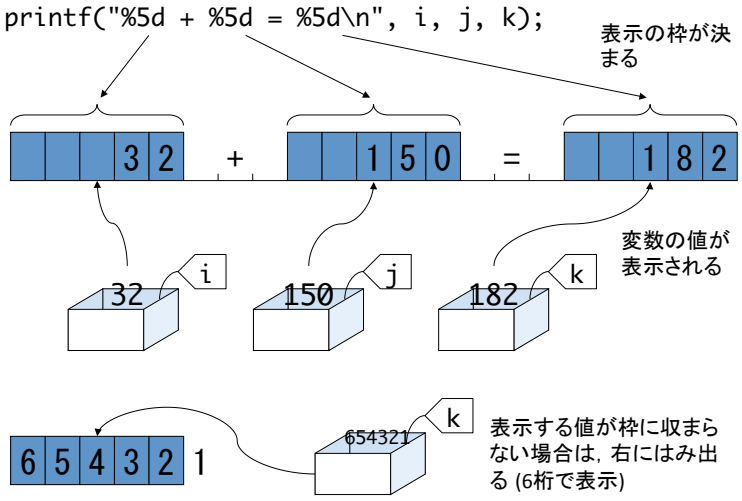
- 変数が3つある
- すべて整数である
- 表示幅は5桁

```
printf("%5d + %5d = %5d\n", i, j, k);
```

```
printf("(float)(%5d/%5d) = %7.3f\n", i, j, f0);
```

- 3つ目の変数は小数である
- 3つ目は小数点以下3桁で表示
- 3つ目は小数点を含めて全体で7桁で表示

# 書式指定出力の例




# 変換指定子，文字幅，精度

| 変換指定子 | 変換の文字列の形式 | 引数のデータ型 |
|---|---|---|
| c | 文字 | char |
| s | 文字列 | 先頭文字のアドレス |
| d | 符号付10進整数 | int |
| u | 符号なし10進整数 | int |
| o | 符号なし8進整数 | int |
| x | 符号なし16進整数 | int |
| f | 浮動小数点数 | float/double |
| e | 浮動小数点数(指数表現) | float/double |

%5d　符号付10進整数(少なくとも5文字幅)
%8f　浮動小数点数(少なくとも8文字幅)
%8.3f　浮動小数点数(少なくとも8文字幅，小数点以下は3桁)

# 書式付出力での出力例

| 出力する形式 | 出力変換仕様 | 表示例 |
|---|---|---|
| 整数 | %d | 321<br>12987 |
| 桁指定の整数(6桁) | %6d | 321<br>12987<br>12345678 |
| 桁指定の整数(6桁)<br>(頭に0を付ける) | %06d | 000321<br>012987<br>12345678 |
| 小数 | %f | 123.456789 |
| 桁指定の小数<br>(全体8桁，小数部3桁)<br>(小数点は1桁になる) | %8.3f | 1.235<br>123.457<br>98765.432 |

値の桁数のほうが大きい場合は揃わない

整数部が5桁なので，小数点が揃わない

# 例題（1）

どのような出力になるか確認しなさい

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
  int i, j, k;    /* 変数宣言 */
  float f0, f1;
  i = 5;          /* 代入 */
  j = 3;
  k = i + j;      /* 算術演算 */
  printf("%5d + %5d = %5d\n", i, j, k);
  printf("%5d - %5d = %5d\n", i, j, i - j);
  return 0;
}
```

# 例題（2）

どのような出力になるか確認しなさい

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
  int i, j, k;    /* 変数宣言 */
  float f0, f1;
  i = 5;          /* 代入 */
  j = 3;
  f0 = (float)(i / j);      /* キャスト */
  f1 = (float)i / (float)j;
  printf("(float)(%d / %d) = %6.3f\n", i, j, f0);
  printf("(float)%d / (float)%d = %6.3f\n", i, j, f1);
  return 0;
}
```

# 標準入力関数 scanf()

- 前もって，stdio.hを読み込むこと
- プログラム中では下のように記述

```
scanf(第1引数，第2引数，第3引数，…);
```

```
char c;
int i, j, k;
float f;
double df;

scanf("%c", &c);          /* 変数cに文字を入力 */
scanf("%d", &i);          /* 変数iに整数を入力 */
scanf("%d %d %f %lf", &j, &k, &f, &df);
```

float型の変数の場合

double型の変数の場合

入力変換仕様
%変換指定子

& は変数のメモリアドレスを計算する演算子

# scanf関数で使用可能な変換文字

| 変換指定子 | 入力文字列の形式 | 変数の型 |
|---|---|---|
| c | 文字 | char |
| s | 文字列 | 先頭文字のアドレス |
| d | 10進整数 | int |
| ld | 10進整数 | long int |
| o | 8進整数 | int |
| x | 16進整数 | int |
| f | 浮動小数点数 | float |
| lf | 浮動小数点数 | double |

scanfでは，第2引数以降に与えた変数の型を考慮しないので，注意が必要である
変換指定子で示された型の値だとして，変数に代入する

```
scanf("%d", &f);
```
← 変数 f の型が何であっても10進整数の値を代入する

# 例題（3）

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
  int i, j, sum, dif;           // 変数宣言

  printf("Enter two integer:");
  scanf("%d %d", &i, &j);       // 入力
  sum = i + j;                  // 算術演算
  dif = i - j;
  printf("%5d + %5d = %5d\n", i, j, sum);
  printf("%5d - %5d = %5d\n", i, j, dif);
  return 0;
}
```

# 例題（4）

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
  float x, y, f0, f1;           // 変数宣言

  printf("Enter two float:");
  scanf("%f %f", &x, &y);       // 入力
  f0 = x + y;                   // 算術演算
  f1 = x * y;
  printf("%8.3f + %8.3f = %8.3f\n", x, y, f0);
  printf("%8.3f * %8.3f = %8.3f\n", x, y, f1);
  return 0;
}
```

# 制御構造

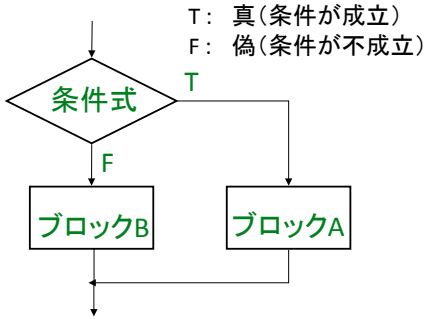
- 文は置かれた順に実行される
  - 上から下
  - 同じ行内であれば，左から右
- 制御構造によって，条件分岐や反復を行える

条件分岐
　if - else 文，switch 文

反復
　for 文，while 文

# if - else 文

```
if ( 条件式 ) {
    ブロックA
}
else {
    ブロックB
}
```




- 「条件式」が成り立てばブロックAを，成り立たなければブロックBを実行
- ブロックは単文でも複文でも可能
- else以下は省略可能

# if - else 文の使用例

```
#include <stdio.h>
#include <math.h>
int main(int argc, const char * argv[]) {
  double x;
  double y;
  printf("x = ");
  scanf("%lf", &x);
  if ( x < 0 ) {
    printf("負なので計算できません\n");
  }
  else {
    y = sqrt(x);
    printf("sqrt(%f) = %f\n", x, y);
  }
  return 0;
}
```

sqrt関数を使うために必要

条件式

条件が成り立つ場合に実行される部分

条件が成り立たない場合に実行される部分

# 比較演算

- 条件式の中には，比較演算や論理演算を書く

| 演算子 | 意味 |
|---|---|
| < | 左辺が右辺より小さい |
| <= | 左辺が右辺以下 |
| > | 左辺が右辺より大きい |
| >= | 左辺が右辺以上 |
| == | 左辺が右辺と等しい |
| != | 左辺が右辺と等しくない |

等号を2つ続ける

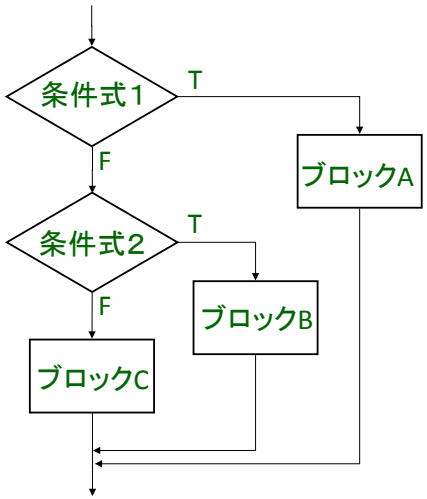
# 比較演算の例

「左辺が右辺以上」の意味

```
if  (age >= 20 ){
  printf("You may drink alcoholic beverage.");
}
else{
   printf("You must not drink alcoholic beverage.");
}
```

# else - if を用いた多分岐

```
if ( 条件式１ ) {
    ブロックA
}else if( 条件式２ ) {
    ブロックB
}
else {
    ブロックC
}
```




ブロックA，ブロックB，ブロックCのうち，いずれか1つが実行される．

# 多分岐の例

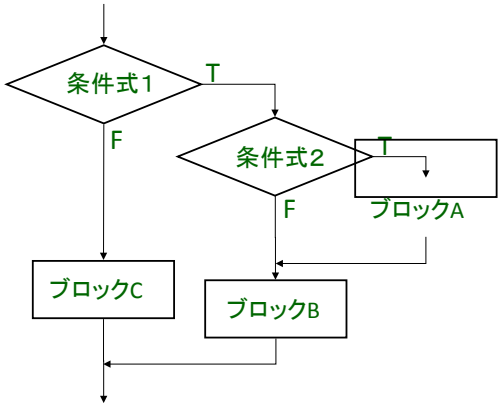
```
#include <stdio.h>
int main(int argc, const char * argv[]) {
  int a;
  printf("年齢を入力してください：");
  scanf("%d", &a);
  if ( a >= 18 ) {
    printf("大人料金1,000円\n");          A
  }
  else if( a >= 12 ) {
    printf("中・高生料金800円\n");        B
  }
  else {
    printf("子供料金500円\n");            C
  }
  return 0;
}
```

このうちの1つが実行される

# if 文の入れ子

- if - else 文のブロックの中に if - else 文を記述しても構わない

```
if ( 条件式１ ) {
  if ( 条件式２ ) {
    ブロックA
  }
  ブロックB
}
else {
  ブロックC
}
```




# 論理演算

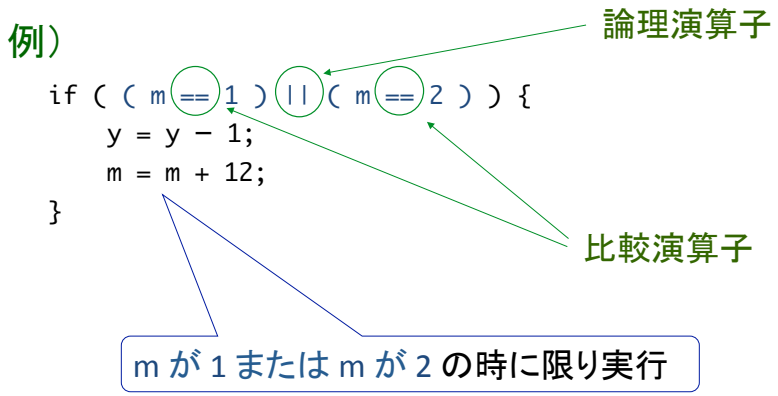
- 真，偽に関する論理的な演算を行う．

| 演算子 | 意味 |
|---|---|
| A && B | A かつ B |
| A \|\| B | A または B |
| !A | Aでない |

AやBは条件式である

# 比較演算と論理演算の組み合わせ

例）




# 月の最後の日を表示するプログラム

```
#include <stdio.h>
/* 入力した月の日数を表示する */
int main(int argc, const char * argv[]) {
  int month, days;
  printf("日数を知りたい月は？ ");
  scanf("%d", &month);
  if (month == 2){  /* 2月のとき */
    days = 28;
  }
  else if((month==4) || (month==6) || (month==9) || (month==11)){
    /* 4,6,9,11月のとき */
    days = 30;
  }
  else{ /* それ以外の月のとき */
    days = 31;
  }
  printf("%d月は%2d日です\n", month, days);
  return 0;
}
```

# switch 文

```
switch ( 条件式 ) {
  case 定数式１ : ブロックA
  case 定数式２ : ブロックB
        :

  default : ブロックC
}
```

- 条件式が定数式と一致する場合，その後のブロックを break または「}」が現れるまで実行
- ブロックは単文でも複文でも可能
- どの case にも当てはまらない場合は，default 以下を実行（省略も可能）

# switch 文の使用例

```
#include <stdio.h>
int main(int argc, const char * argv[]) {
  int a;
  printf("選択してください\n");
  printf("\t大人＝1\n\t中・高生＝2\n\t子供＝3\n\t\t：");
  scanf("%d", &a);
  switch (a) {
    case 1:
      printf("大人料金1,000円\n");
      break;
    case 2:
      printf("中・高生料金800円\n");
      break;
    default:
      printf("子供料金500円\n");
  }
  return 0;
}
```

# 月の最後の日を表示するプログラム（2）

```
#include <stdio.h>
/* 入力した月の日数を表示する */
int main() {
    int month, days;
    printf("日数を知りたい月は？");
    scanf("%d", &month);
    switch(month){
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            days = 31;
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            days = 30;
            break;
        case 2:
            days = 28;
            break;
        default:;
    }
    printf("%d月は%2d日です.",
            month, days);
}
```

# 定回反復とその必要性

あらかじめ定められた回数がある．
その回数の中で同じことを繰り返すことを言う．

定回反復を表す命令(for 制御構造)の必要性

- Aを画面に3回書け．

```
printf("A"); printf("A"); printf("A");
```

または，`printf("AAA");`

- Aを画面に20回書け．

```
printf("AAAAAAAAAAAAAAAAAAAA");
```

- Aを画面に10,000回書け．

```
printf("AAAAAAAAAA ... AAA ");
```

10,000文字？？

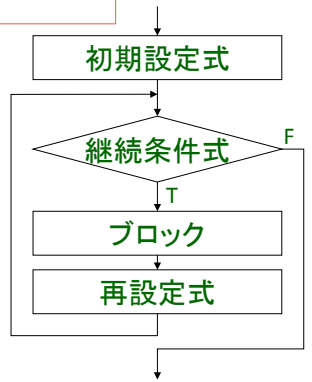
# 繰り返し

- C言語での繰り返しは次の3通り
  - for 文
  - while 文
  - do-uhile 文
- ループ変数 （ループカウンタ）
  - 繰り返しの回数を数える変数
  - インクリメント　値を1増やす（i++）
  - デクリメント　値を1減らす（i--）

# for 文

```
for（初期設定式；継続条件式；再設定式）{
   ブロック
}
```

- 初期設定式はループに先立って1回だけ実行される．
- 継続条件式が成り立っている間，ブロックを実行する．
- 条件が成り立たなければ，ループは終了する．
- ブロックの実行後，再設定式が評価される．



# 九九の表を表示するプログラム

```
#include <stdio.h>
/* 九九を表示する */
int main(int argc, const char * argv[]) {
    int i, j;
    for(i = 1; i <= 9; i++){
        for(j = 1; j <= 9; j++){
            printf("%3d", i * j);
        }
        printf("\n");    // 改行する
    }
    return 0;
}
```

# 不定回反復とその必要性

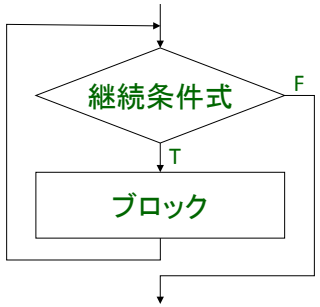
- 不定回反復の必要性
  - –繰り返しの回数がまえもって分からない場合は，for 文で記述するのは不適
  - –論理式が正しい間，繰り返す

```
while( 論理式 ){
    文1
    文2
    ⋮
}
```

```
do{
    文1
    文2
    ⋮
} while( 論理式 );
```

# while 文

```
while (継続条件式) {
    ブロック
}
```




- 継続条件式が成り立っている間，ブロックを実行する
- 条件が成り立たなければ，ループは終了する
- ブロックの実行前に条件を評価するので，ブロックが1回も実行されない場合もある

# while 文の使用例

```
#include <stdio.h>
/* 0が入力されるまで入力値を加算していく */
int main(int argc, const char * argv[]) {
    int sum, x;
    sum = 0;
    printf("data? ");
    scanf("%d", &x);
    while(x != 0){  /* 0が入力されるまで繰り返される */
        sum += x;
        printf("data?");
        scanf("%d", &x);
    }
    printf("sum = %d", sum);
    return 0;
}
```

sum = sum + x を意味する

# sum += i とは

□ は sum という変数の中身を表す

ループに入る前　sum = 0

□ = 0

iが1で実行　sum = sum + 1 ← $sum_1 = sum_0 + 1$

□ = □ + 1

iが2で実行　sum = sum + 2 ← $sum_2 = sum_1 + 2$

□ = □ + 2

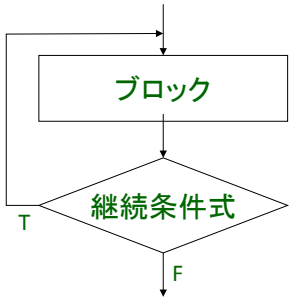
⋮

iがnで実行　sum = sum + n ← $sum_n = sum_{n-1} + n$

□ = □ + n

# do - while 文

```
do {
    ブロック
} while (継続条件式);
```




- 継続条件式が成り立っている間，ブロックを実行する．
- 条件が成り立たなければ，ループは終了する．
- ブロックを実行した後に条件を評価するので，ブロックを必ず1回は実行する．

# do - while 文の使用例

```
#include <stdio.h>
/* 0が入力されるまで入力値を加算していく */
int main(int argc, const char * argv[]) {
  int sum, x;
  sum = 0;
  do{
    printf("data?");
    scanf("%d", &x);
    sum += x;
  } while(x != 0);
  printf("sum = %d", sum);
  return 0;
}
```

# 1から10までの合計

- 3種類のループはどれを使っても構わない
- （こんなことをしてはいけない！）

for ループ

```
int sum = 0;
for (int k = 1; k <= 10; k++) {
    sum += k;
}
```

変数 k はループ変数（ループの回数を数えるための変数）

while ループ

```
int sum = 0;
int k = 1;
while (k <= 10) {
    sum += k;
    k++;
}
```

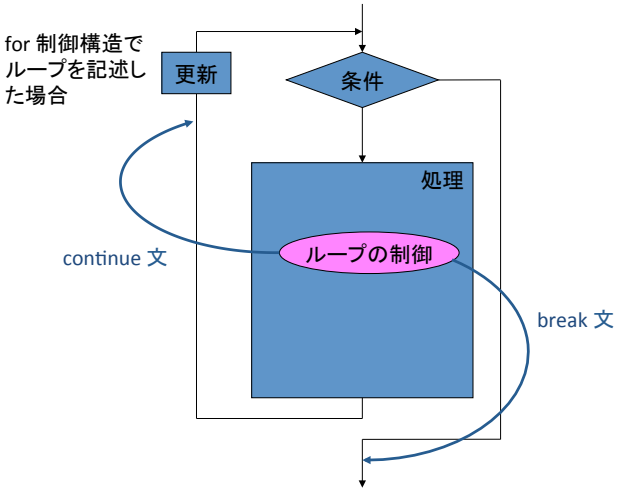
do-while ループ

```
int sum = 0;
int k = 1;
do {
    sum += k;
    k++;
} while (k <= 10)
```

# ループ途中での実行の制御

- ループの途中で，繰り返している処理を…
  - 終了させたい
    （ループから抜ける）
    - break 文
    - ループのブロック，switch 制御構造のブロックで使用可能
  - ループ内の残りの処理を省きたい
    （ループは続行）
    - continue 文
    - ループのブロックで使用可能

# 2つの制御文の流れ




# 素数の計算

- 2は素数だが、2の整数倍数は素数ではない。
- 3は素数だが、3の整数倍数も素数ではない。
- 1つでも割り切れたら素数ではない
- 3からの奇数だけで検討
- 求める数の平方根より大きい数字で割っても意味がない

- Nまでの数を入力して指定
- 1行に10個ずつを表示

# 素数のプログラム例

```c
#include <stdio.h>
#include <math.h>
#define WIDTH 10

int main(int argc, const char * argv[]) {

    int i, j, k, n, flag;

    printf("Plase input the number\n");
    scanf("%5d", &n);

    k = 1;
    printf("      2");

    for(i=3; i<=n; i+=2) {

        flag = 0;
        for(j=3; j<=sqrt(i) && flag == 0; j+=2) {
            if(i%j == 0) {
                flag=1;
                break;
            }
        }
        if(flag == 0) {
            printf("   %5d",  i);
            k++;
            if (k >= WIDTH) {printf("\n"); k = 0;}
        }
    }

    printf("\n");

    return 0;
}
```

# 課題

- 素数の計算をするプログラムを作成せよ。
- N個までのNとして、自分の学生番号の下3桁を使って実行した結果を貼付けて示せ。もし下3桁の数が50以下の場合には100を足して実行せよ。
- またプログラムファイルmain.cを添付せよ。

# アンケート調査

- WebCTに準備してあるアンケートに回答をしてください。よろしくお願いします。
- URLは次の通り
  http://rupus.i.kyushu-u.ac.jp/~suga/Ipclass/
- アカウントとパスワードは次の通りです。
  - ユーザ名 : zengaku
  - パスワード : jouhoushori