ソフトウェア２
第３回
（2015/12/3）

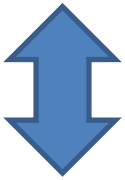
鶴岡 慶雅

今日の内容

- C言語入門
  - 動的メモリ確保
    - malloc, free
  - データ構造
    - 線形リスト
- アプリケーション
  - ペイントソフト

動的メモリ確保

- 配列
  - サイズを実行時に決められない
    - 注）最近のコンパイラ（C99対応）であれば可
  - スコープを抜けると解放される



- malloc による動的メモリ確保
  - サイズを実行時に指定
  - 明示的に解放されない限り、プログラムの実行が終わるまでメモリ領域が確保されている

malloc() 関数

- メモリの動的確保

```
int *ptr = malloc(10 * sizeof(int));
```

  - int 10個分のメモリ(40バイト)をヒープ領域に確保
  - 確保したメモリの先頭アドレスを返す
  - 確保に失敗した場合は NULL を返す

- メモリの解放
  - そのメモリが不要になったら free 関数で解放する

```
free(ptr);
```

  - 解放し忘れ　→ メモリリーク

malloc() 関数

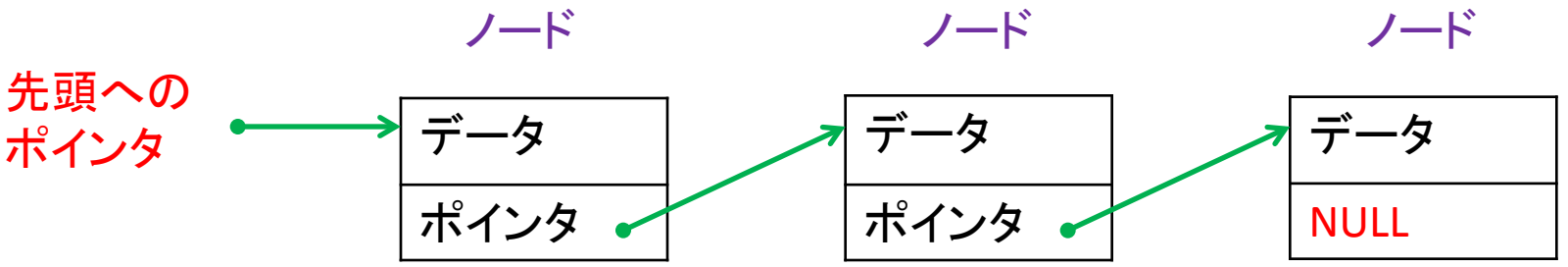
- 使い方の例

```
int *ptr = malloc(10 * sizeof(int));

if (ptr == NULL) {
  exit(1); // メモリ確保に失敗
}

ptr[0] = 123;
ptr[1] = 555;

free(ptr);
```

  - ポインタを利用して配列と似たように使える

線形リスト（linear list）

- 多数のデータを格納するためのデータ構造のひとつ
- 格納する各要素をポインタで連結

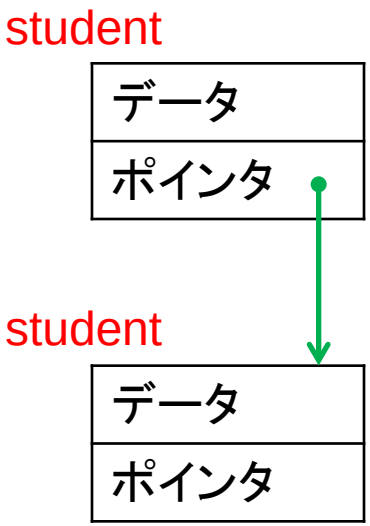


- 要素の削除、挿入のコストが小さい
  - 前後のポインタをつなぎかえるだけ
- 要素数の制限がない
  - ひとつ増えるたびに malloc すればよい
- ランダムアクセスのコストは高い

線形リストの実装法

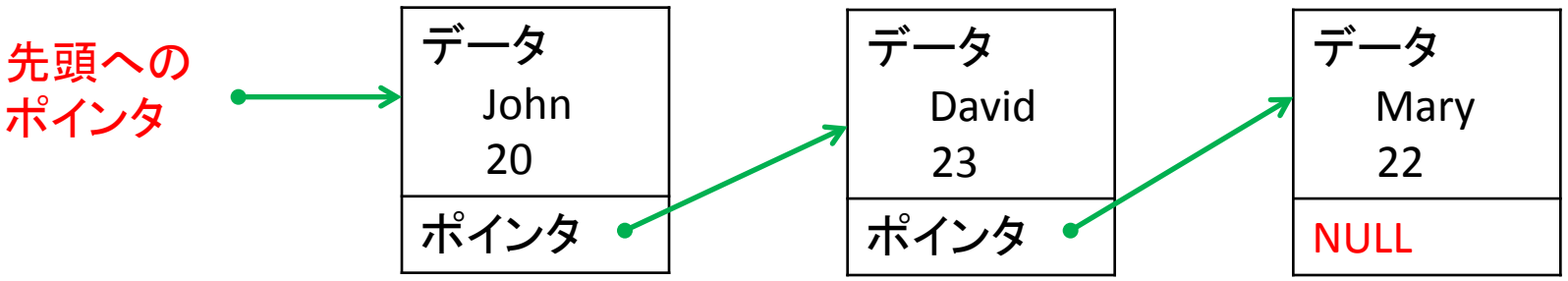
- 自己参照構造体を使う
  - 自己参照構造体：自分と同じ構造体を指すポインタをメンバに持つ構造体

```
struct student
{
        char name[20];
        int age;
        struct student *next;
};
```



線形リスト（linear list）

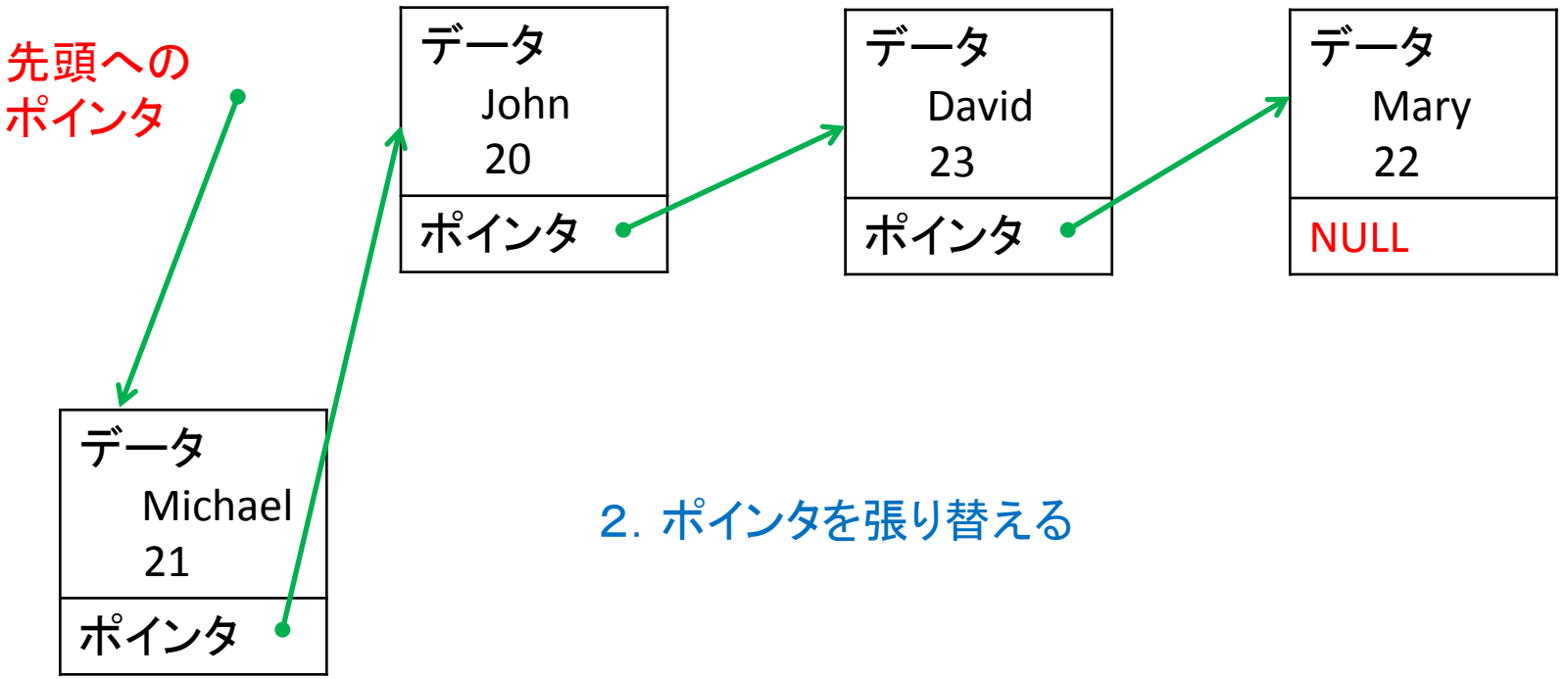
- 要素の挿入（先頭に挿入する場合）



1．挿入したいデータをヒープメモリにコピー

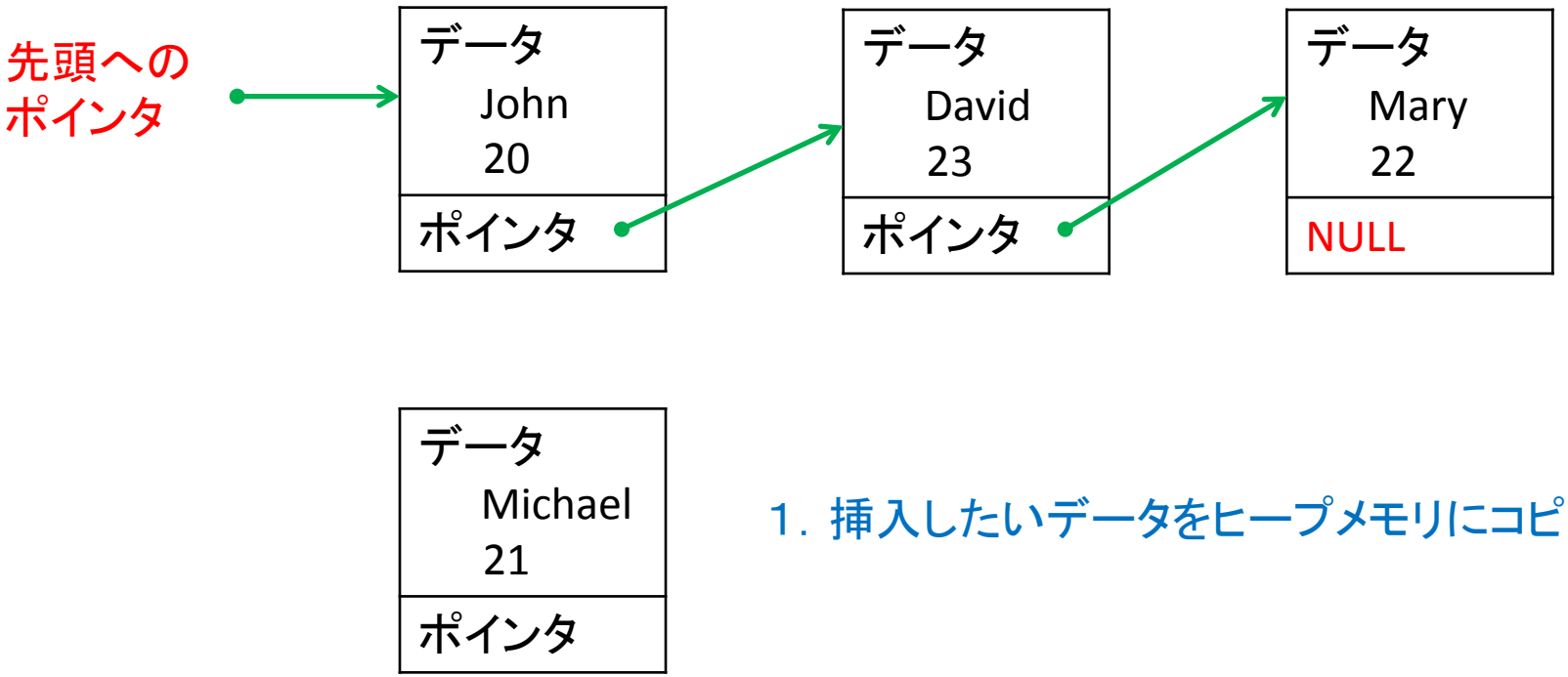
線形リスト（linear list）

- 要素の挿入（先頭に挿入する場合）



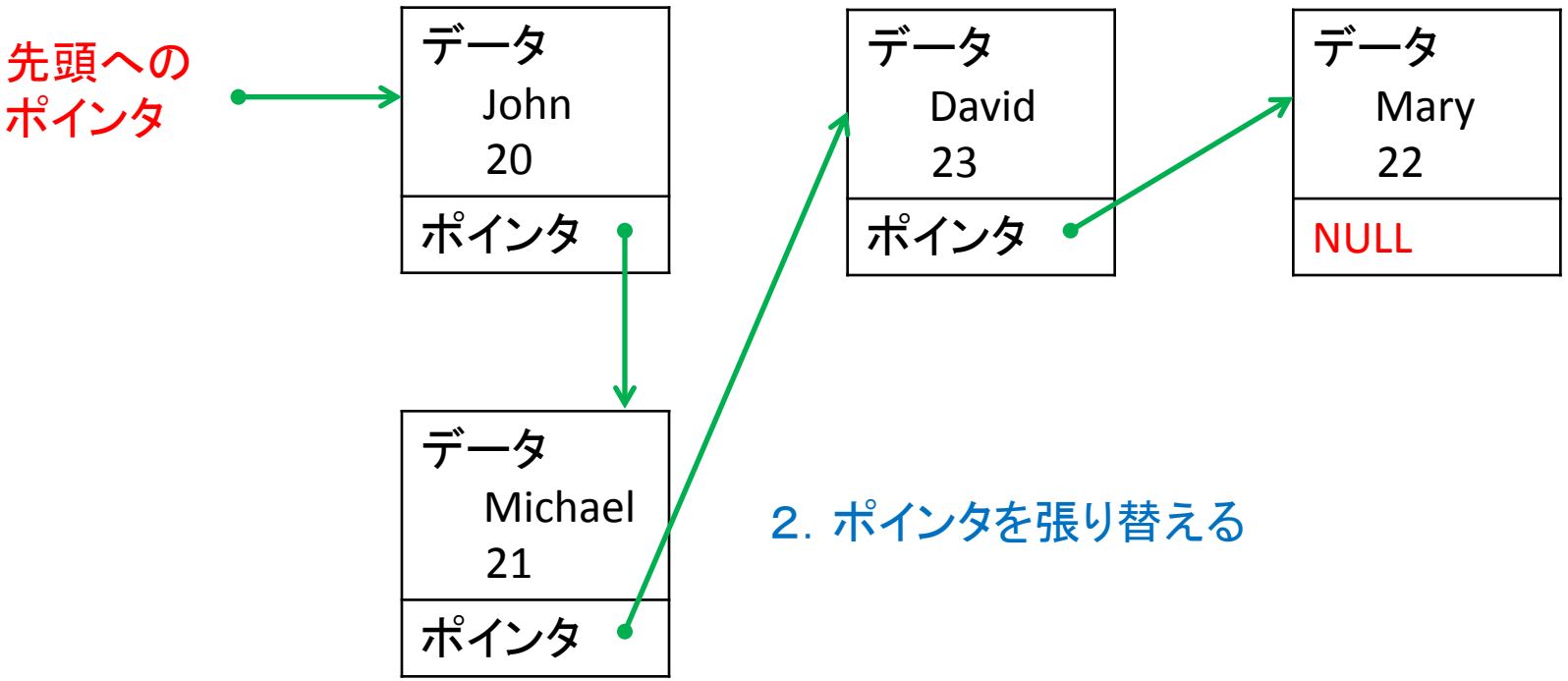
線形リスト（linear list）

- 要素の挿入（途中に挿入する場合）



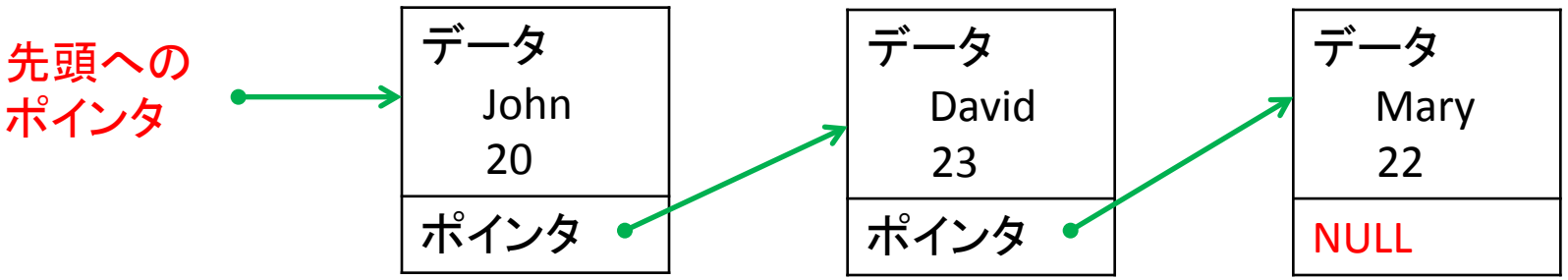
線形リスト（linear list）

- 要素の挿入（途中に挿入する場合）



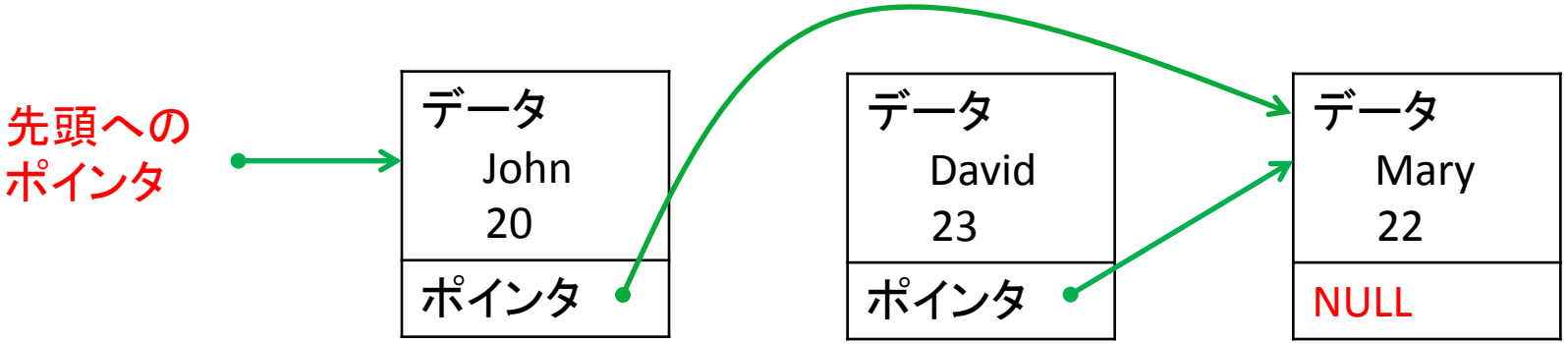
線形リスト（linear list）

- 要素の削除の例



線形リスト（linear list）

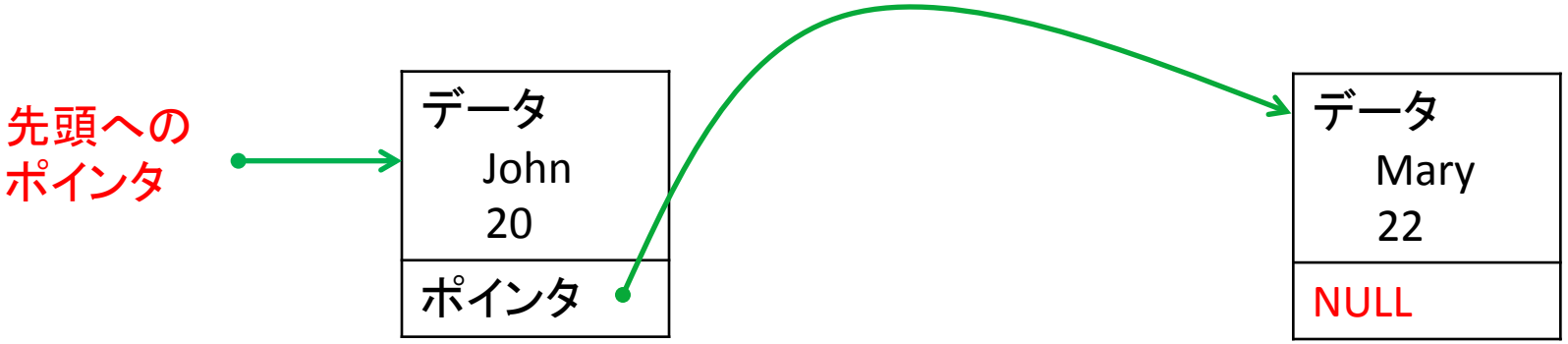
- 要素の削除の例



1. ポインタを張り替える

線形リスト（linear list）

- 要素の削除の例



２．削除するデータのメモリを解放

簡単な線形リストの実装例

```c
#include <stdio.h>
#include <stdlib.h>

struct student
{
  int id;
  struct student *next;
};

typedef struct student Student;

Student *push_front(Student *p, int id)
{
  Student *q = malloc(sizeof(Student));

  q->id = id;
  q->next = p;

  return q;
}
```

```c
int main()
{
  Student *begin = NULL;
  begin = push_front(begin, 1);
  begin = push_front(begin, 2);
  begin = push_front(begin, 3);

  const Student *p;
  for (p = begin; p != NULL; p = p->next){
    printf("%d¥n", p->id);
  }

  return 0;
}
```

実行結果

```
$ ./a.out
3
2
1
```

typedef について

- 型に別名をつけることができる
- たとえば、構造体

```
struct student
{
  int age;
  struct student *next;
};
```

に関して、毎回 struct student と書くのは面倒なので

```
typedef struct student Student;
```

型　　別名

と別名をつけると、以降は Student と書けば済む

typedef について

- 構造の宣言とまとめて以下のように書くこともできる

```
typedef struct student
{
  int age;
  struct student *next;
} Student;
```

- 自己参照のない構造体であれば構造体のタグ名も省略可

```
typedef struct
{
  int id;
  int age;
} Student;
```

サンプルプログラム list.c

- 処理内容
  - 標準入力から1行ずつ読み込み、文字列を線形リストに保存
  - 線形リストを先頭から順にたどり、文字列を標準出力に書きだす
  - 実行例

```
% ./a.out < list.c
```

  - 線形リストではなく配列で同じ処理を実装したらどうなるか？

- 線形リストの操作
  - begin: 先頭ノードへのポインタ
  - push_front() 関数: 先頭に要素を追加
  - push_back() 関数: 末尾に要素を追加
  - pop_front() 関数: 先頭の要素を削除

list.c 冒頭

- 自己参照構造体の宣言

```
struct node
{
  char *str;
  struct node *next;
};

typedef struct node Node;
```

  - データは文字（列）へのポインタのみ
  - 文字列そのものは保持しないことに注意

push_front() 関数

- 先頭に要素を挿入
  - 入力： リストの先頭へのポインタ、格納する文字列
  - 出力： （挿入後の）リストの先頭へのポインタ
  - malloc で、ノードおよび文字列のメモリを確保し、データ（文字列）を保存した後、ポインタを張り替える

```
Node *push_front(Node *begin, const char *str)
{
  Node *p = malloc(sizeof(Node));
  char *s = malloc(strlen(str) + 1);
  strcpy(s, str);
  p->str = s;
  p->next = begin;

  return p;
}
```

strlenでは末端の‘¥0’がカウントされないので

pop_front() 関数

- 先頭の要素を削除
  - 入力： リストの先頭へのポインタ
  - 出力： （削除後の）リストの先頭へのポインタ
  - free で、ノードおよび文字列のメモリを解放
  - ２番目だったノードが新たな先頭ノードになる

```
Node *pop_front(Node *begin)
{
  Node *p = begin->next;

  free(begin->str);
  free(begin);

  return p;
}
```

push_back() 関数

- 末尾に要素を追加
  - 末尾の要素に行きつくまでリストを先頭からたどり、その後ろに新たな要素を追加
  - ただし、リストが空のときは特別扱い

```
Node *push_back(Node *begin, const char *str)
{
  if (begin == NULL) return push_front(begin, str);

  Node *p = begin;
  while (p->next != NULL) {
    p = p->next;
  }

   :  // 末尾に要素を追加
}
```

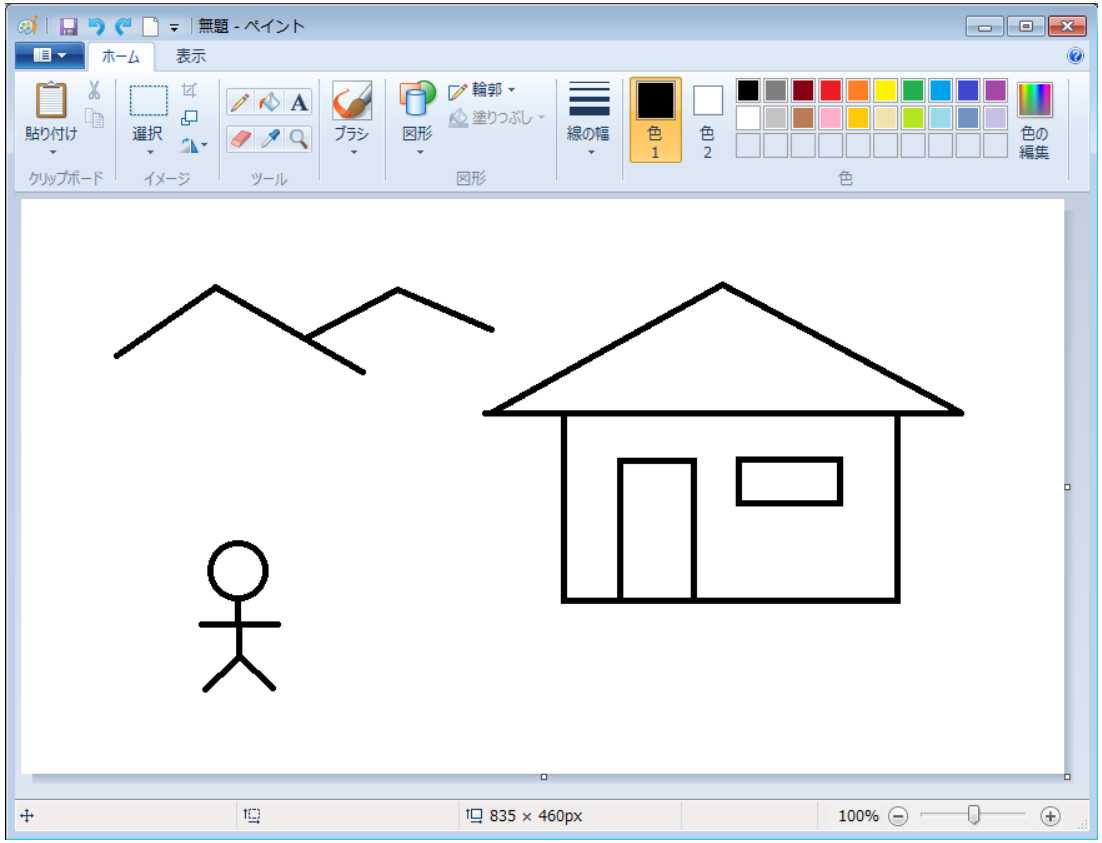
実習1

- list.c に、末尾の要素を削除する pop_back() 関数を追加せよ
  - 計算量は O(*n*) で構わない

リスト中の要素数 *n* に比例する計算量という意味

ペイントソフト

- コマンド入力方式で絵を描く
  - 描画機能
    - 線を描く
    - 長方形を描く
    - 円を描く
    - :
  - Undo
    - 直前のコマンドの取り消し
  - 履歴の保存



サンプルプログラム paint.c

- コンパイル＆実行

  % gcc paint.c

  % ./a.out

  0 > line 10 10 20 10 ← (10, 10) から (20, 10) まで線を引く

  1 > save ← コマンド履歴の保存

  2 > quit ← 終了

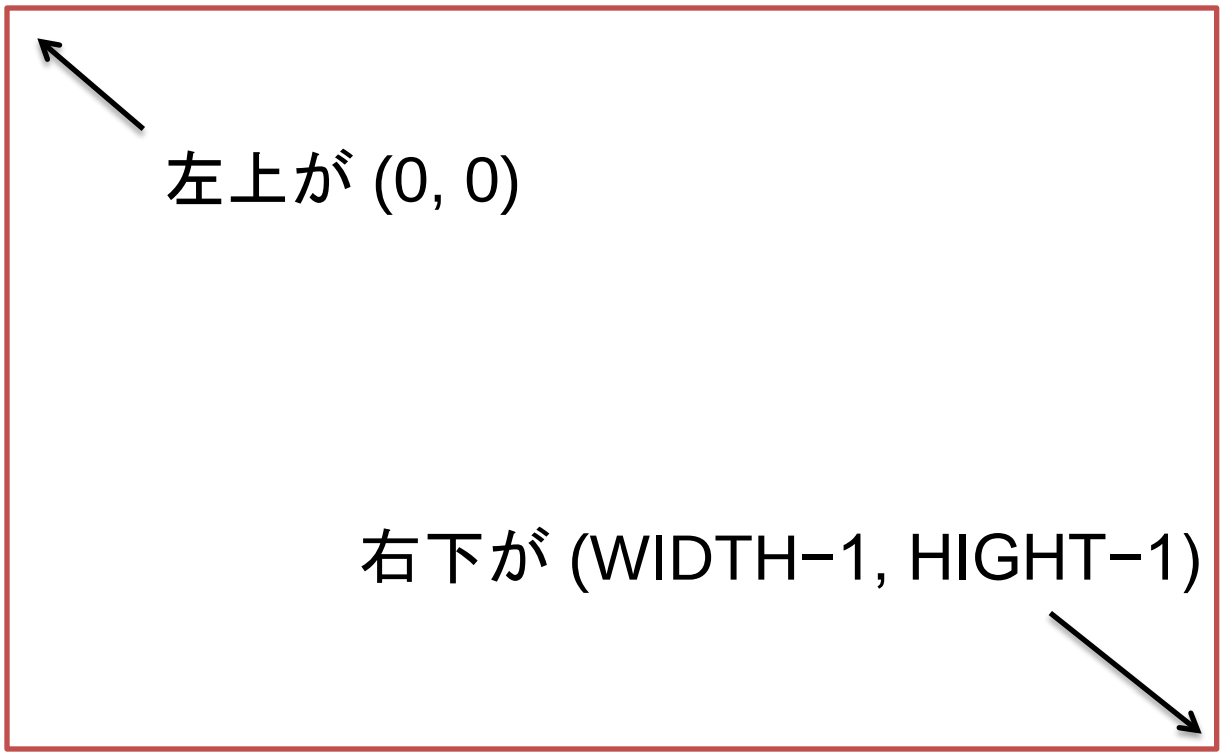
- ターミナルをもうひとつ開く（キャンバス用）

  % tail –f canvas.txt

  ターミナルのサイズをマウスで調整して ---------- が左上にくるように

キャンバス

```
char canvas[WIDTH][HIGHT]
```



※メモリ上では、ドットが上下方向に連続することになるが
canvas[x 座標][y 座標] としたかったので

interpret_command() 関数

- コマンド文字列をトークン列に分解して最初の単語を取り出す
  - strtok関数で最初のトークンを取得
    - デリミタ(区切り文字)は空白文字
    - 文字列が破壊されるのでコピーしたものを渡す

```
int interpret_command(const char *command)
{
  char buf[BUFSIZE];
  strcpy(buf, command);

  char *s = strtok(buf, " ");
```

interpret_command 関数

- 残りの文字列もトークンに分解
  - strtok関数で2つ目以降のトークンを取得
    - 第1引数をNULLにして呼び出せばよい
  - atoi関数で文字列を整数値に変換

```
if (strcmp(s, "line") == 0) {
  int x0, y0, x1, y1;
  x0 = atoi(strtok(NULL, " "));
  y0 = atoi(strtok(NULL, " "));
  x1 = atoi(strtok(NULL, " "));
  y1 = atoi(strtok(NULL, " "));
  draw_line(x0, y0, x1, y1);
  return 0;
}
```

draw_line() 関数

- 線を引く
  - 始点と終点をn等分して点を打つ
  - nは点と点の隙間が空かないように決める

```
void draw_line(int x0, int y0, int x1, int y1)
{
  int i;
  const int n = max(abs(x1 – x0), abs(y1 – y0));
  for (i = 0; i <= n; i++) {
    const int x = x0 + i * (x1 - x0) / n;
    const int y = y0 + i * (y1 - y0) / n;
    canvas[x][y] = '#';
  }
}
```

Undo について

- ひとつ前のコマンドを取り消す
  - キャンバスを初期化し、最初のコマンドから直前のコマンドまでを実行しなおす
  - コマンドの履歴の長さを1減らす
    - ひとつ前のコマンドのぶん

```
init_canvas();
for (i = 0; i < *hsize - 1; i++) {
  interpret_command(history[i], NULL);
}
(*hsize)--;
```

実習2

- コマンド履歴の保存方法を改良せよ
  - list.c を応用し、メモリの無駄、コマンド履歴長の制限がなくなるように

課題（締め切り12/9）

1. paint.c に長方形を描くコマンドと円を描くコマンドを追加せよ
   - コマンドの名前、引数の形式は任意
   - プログラムを添付すること（ファイル名は “paint1.c”）

2. ファイルに保存されたコマンド履歴を読み込み絵を再描画するコマンド “load” を追加せよ
   - プログラムを添付すること（ファイル名は “paint2.c”）

1. ［発展課題］ paint.c に、他に有用なコマンド（塗りつぶし、エフェクトをかける、コピー＆ペースト、BMP形式で保存、など）を追加せよ
   - プログラムを添付すること（ファイル名は “paint3.c”）
   - 追加したコマンドについて簡単に説明すること

課題の提出方法

- 宛先
  - software2@logos.t.u-tokyo.ac.jp
- Subject
  - 形式: SOFT-MM-DD-NNNNNNX
    - MM: 月
    - DD: 日（授業が行われた日）
    - NNNNNNX: 学籍番号
- 本文
  - 冒頭に学籍番号、氏名を明記

参考　三角関数

```
double sin(double x);
double cos(double x);
double tan(double x);
```

- 三角関数の値を計算
  - 引数の単位はラジアン
- <math.h> をインクルード
- コンパイル時に -lm オプション