手続き型プログラミング発展
第9回

（実習第5回）

スタック，キュー，リストに関する実習

2017年7月10日(月)

本日の内容

- **講義第4回の復習**
  - リスト
  - スタック
  - キュー
- **実習課題4**

連絡事項
7/17(月)の手続き型プログラミング発展は通常通り開講します。

講義第4回の復習

構造体と自己参照構造体

復習：構造体とは、任意の型の変数をまとめて管理するための型である。

```
#include <stdio.h>

struct xyz {
    int x;
    char y;
    double z;
};

int main(void) {
    struct xyz A = {
        1,
        'a',
        0.0
    };
```

構造体タグ

メンバー

構造体 xyz 型の
変数 A の宣言と初期化

typedef宣言

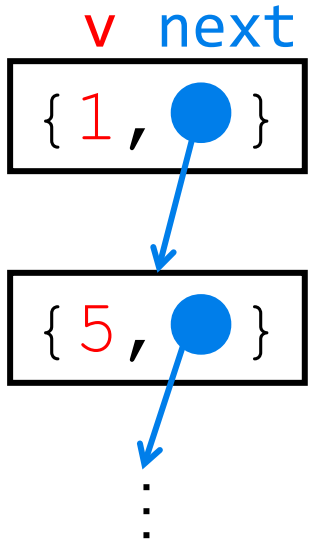
```
typedef struct Hoge {
        int x;
        char y;
        double z;
} Fuga;

Fuga A = {1, 'a', 0.0};
```

構造体と自己参照構造体

- 構造体が自身と同じ型のデータへのポインタをメンバとすることを**自己参照**という
- 自己参照構造体を用いることにより、再帰的なデータ構造を扱うことができる

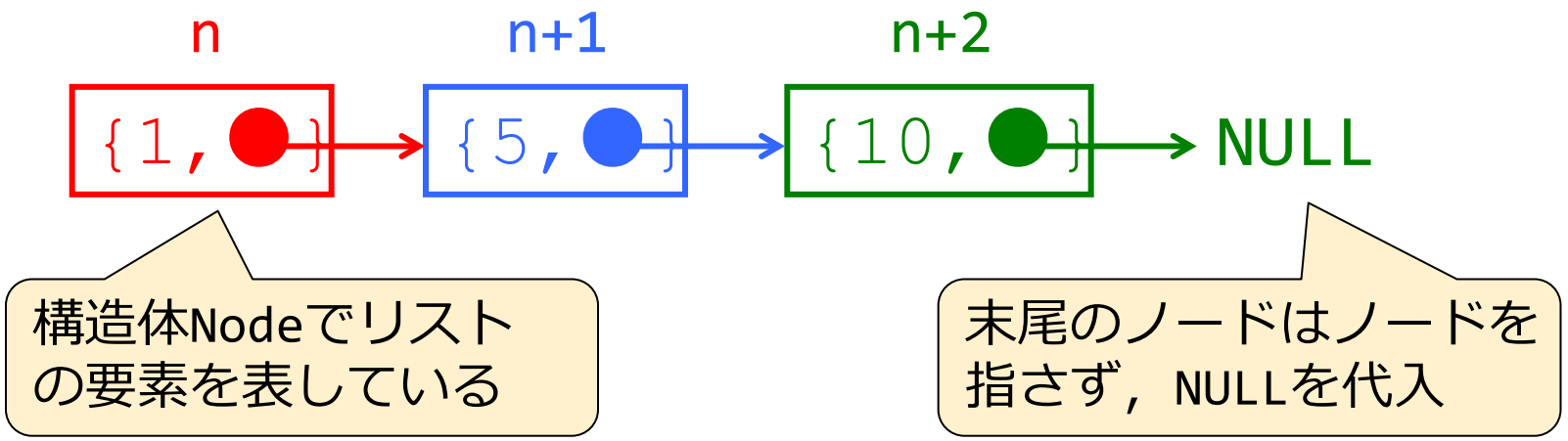
```
typedef struct Node {
    int v;
    struct Node *next;
} Node;
```



リスト

- リストは、データの列を表した再帰的データ構造
- ポインタでリストの要素を順次つなぐ方法で実現したリストを（片方向）連結リストという

```
struct Node *n;
```



リストの実装例


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

typedef struct Node {
    int v;
    struct Node *next;
} Node;

Node *cons(int v, Node *list) {
    Node *new = malloc(sizeof(Node));
    if (new == NULL) exit(2);
    new->v = v;
    new->next = list;
    return new;
}
```

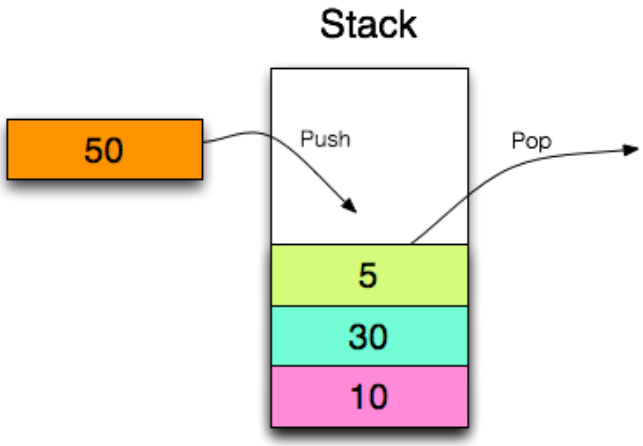
```c
int main(void) {
    Node *tmp;
    Node *list = cons(10, NULL);
    list = cons(5, list);
    list = cons(1, list);

    while (list != NULL) {
        printf("%d\n", list->v);
        tmp = list;
        list = list->next;
        free(tmp);
    }

    return 0;
}
```

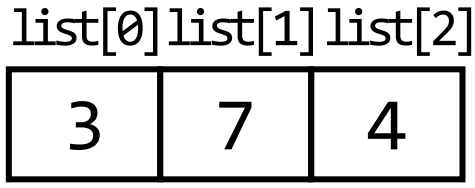
スタックとその実装

- **スタック**
  - データを後入れ先出し（Last In First Out, LIFO）の構造で保持
  - **push**　先頭にデータを挿入する
  - **pop**　先頭のデータを取り出す
  - pushは任意の値を指定可能だが、popでは取り出す値を指定できない



- **スタックの実装方法**
  - 固定長配列による実装
    - スタックオーバーフローが発生する可能あり

```
int list[3];
```

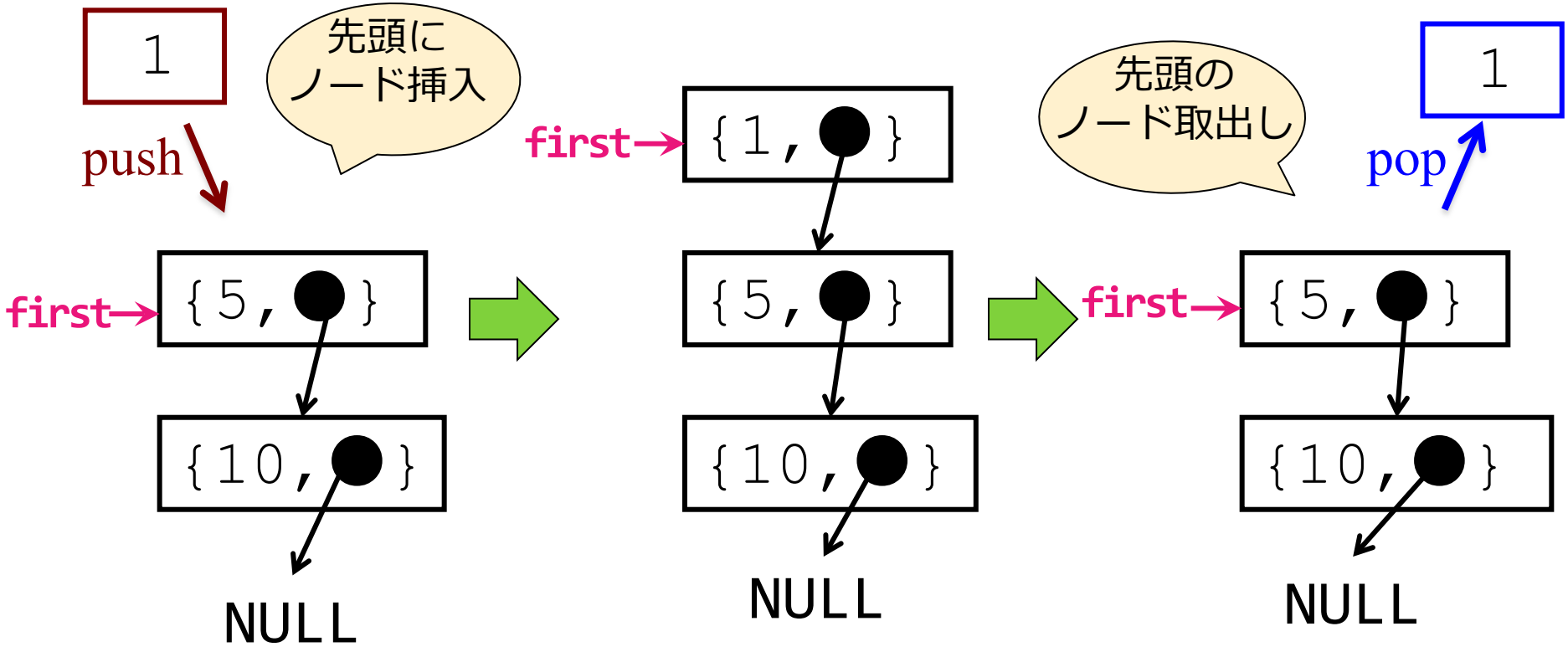


pushするとスタックオーバーフロー発生

  - 連結リストによる実装
    - 次スライドで説明

連結リストによるスタックの実装

- 連結リストと，連結リストの先頭のノードを指すスタックポインタ (first) を用いてスタックを実現



スタックの実装例


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

typedef struct Node {
    int v;
    struct Node *next;
} Node;

void push(Node **first, int v) {
    Node *new = malloc(sizeof(Node));
    if (new == NULL) exit(2);
    new->v = v;
    new->next = *first;
    *first = new;
}

int pop(Node **first) {
    int v;
    Node *tmp = *first;
    if (tmp == NULL) exit(2);
    v = tmp->v;
    *first = tmp->next;
    free(tmp);
    return v;
}
```

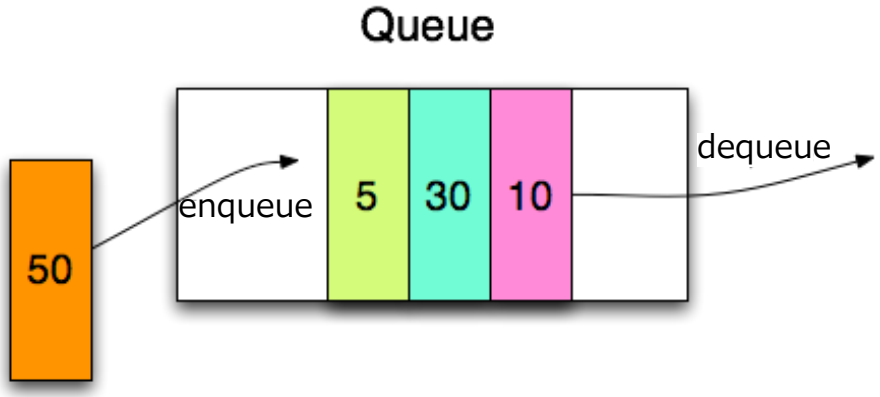
```c
int main(void) {
    Node *first = NULL;
    push(&first, 10);
    push(&first, 5);
    push(&first, 1);

    while (first != NULL) {
        printf("%d\n", pop(&first));
    }

    return 0;
}
```

キューとその実装

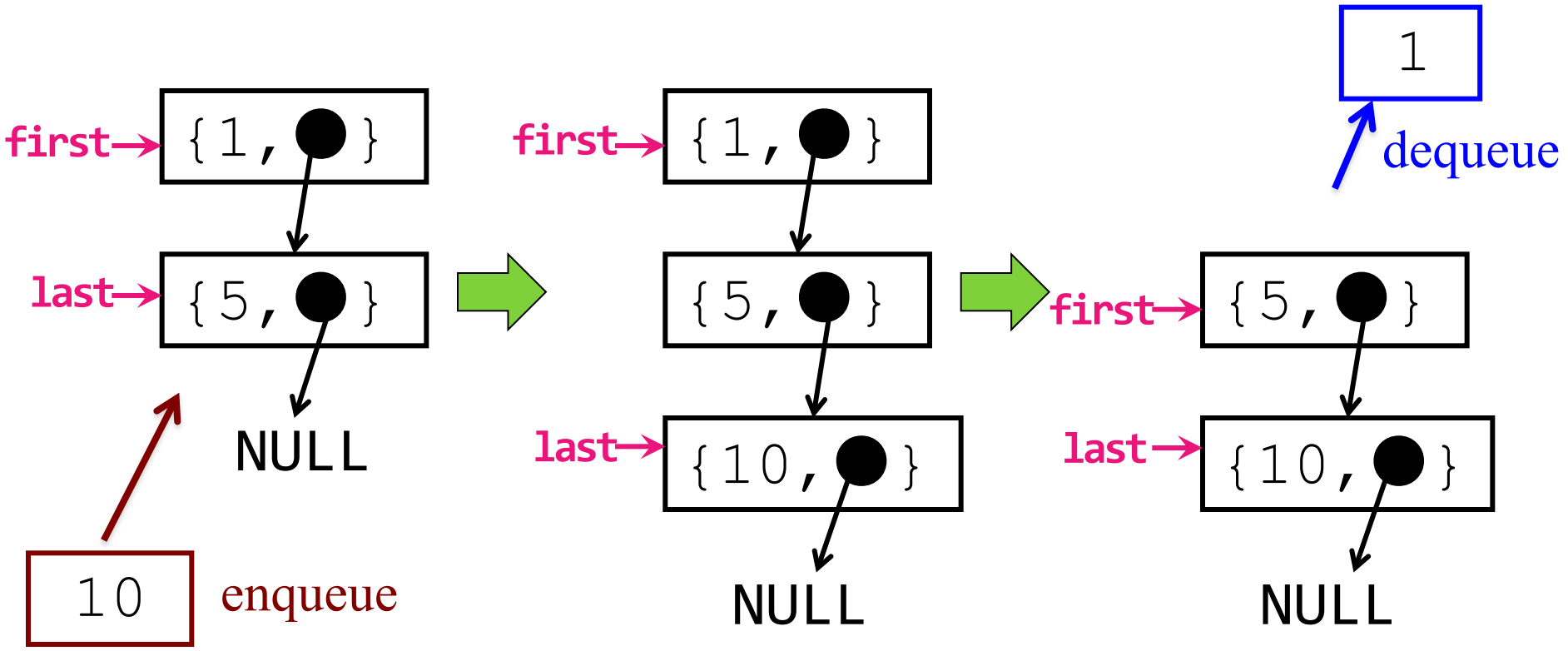
- **キュー**
  - データを先入れ先出し（First In First Out, FIFO）の構造で保持
  - **enqueue**　最後尾にデータを挿入する
  - **dequeue**　先頭のデータを取り出す
  - enqueueは任意の値を指定可能だが、dequeueでは取り出す値を指定できない



- **キューの実装方法**
  - 連結リストによる方法（次スライドで説明）

連結リストによるキューの実装

- 連結リストと、先頭のノードを指すポインタ (first) と末尾のノードを指すポインタ (last) を用いてキューを実現



キューの実装例


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

typedef struct Node {
    int v;
    struct Node *next;
} Node;

void enqueue(Node **first, Node **last,
             int v) {
    Node *new = malloc(sizeof(Node));
    if (new == NULL) exit(2);
    new->v = v;
    if (*first == NULL)
        *first = new;
    else
        (*last)->next = new;
    *last = new;
}
```

```c
int dequeue(Node **first) {
    int v;
    Node *tmp = *first;
    if (tmp == NULL) exit(2);
    v = tmp->v;
    *first = tmp->next;
    free(tmp);
    return v;
}

int main(void) {
    Node *first = NULL, *last;
    enqueue(&first, &last, 1);
    enqueue(&first, &last, 5);
    enqueue(&first, &last, 10);

    while (first != NULL) {
        printf("%d\n", dequeue(&first));
    }

    return 0;
}
```

実習課題4

実習課題h-ex4-1 A課題

- **整数値を格納可能なスタックを実装し、以下の機能を有するプログラムを作成せよ。**
  - 「push␣整数値」という入力が与えられたら、与えられた整数値をスタックに格納する。
  - 「pop」という入力が与えられたら、スタックから整数をpopして標準出力に出力する。
  - 「printstack」という入力が与えられたら、現在のスタックの内容を先頭から順にスペース区切りで標準出力に出力する。

入力例

```
push␣3 ↲
push␣5 ↲
push␣-10 ↲
pop ↲
pop ↲
push␣7 ↲
push␣-1 ↲
printstack ↲
```

出力

```
-10 ↲
5 ↲
-1␣7␣3 ↲
```

実習課題h-ex4-2 A課題

- **入力された文字列中の3種類の括弧 ( ), { }, [ ]の並びの整合性が取れているかどうかを判断するプログラムを作成せよ。**
  - 「括弧の並びの整合性が取れている文字列」とは、以下のように括弧の対応が正しい文字列のことを言う。

    ```
    (po[ke(mon){go}])
    (((((tepukiso)))tepuhatsu))
    ```

    ※以下の文字列は「括弧の並びの整合性が取れていない」

    ```
    (po[ke(mon]{go}))
    ((((tepukiso))tepuhatsu)
    ```

  - 入力される文字列は'(', ')', '{', '}', '[', ']'と、英字大文字・小文字で構成されるものとする。
  - 入力された文字列の整合性が取れている場合は0、取れていない場合は1を出力せよ。

実習課題h-ex4-2 A課題

入力例1

```
(po[ke(mon){go}])
```

出力

```
0
```

入力例2

```
(hanero[koi(kin]{gu}))
```

出力

```
1
```

入力例3

```
FUNCTION()
```

出力

```
0
```

実習課題h-ex4-3 B課題

- **n人の客(C_1,…,C_n)がパン屋に並んでいる。各客C_iは、購入したいパンの個数N_i>0があるが、パン屋は先頭の客による買い占めを防ぐため、一度に最大m個までパンを売ることにした。1会計が終わった段階で、パンの個数がN_iに満たない客C_iは再度行列の最後尾に並び直すものとし、希望の個数を購入できた客は行列から抜けるものとする。以下に示す入力に対し、1会計ごとの行列の状態を出力するプログラムを作成せよ。**
  - 客の人数n、1会計の最大販売数m、各客の購入希望数N_iが改行区切りで入力される。
  - 客はC1，C2，…という文字列で表すこととし、客の行列の順にスペース区切りで出力せよ。
  - 1行目に行列の初期状態（1回目の会計前の状態）を出力せよ。
  - 行列に並んだ客がいなくなったら出力をやめること。
  - データ構造にキューを用いること。

実習課題h-ex4-3 B課題

入力例

5 ↲ ←n
3 ↲ ←m
2 ↲ ←N_1
5 ↲ ←N_2
10 ↲ ←N_3
4 ↲ ←N_4
5 ↲ ←N_5

出力

C1␣C2␣C3␣C4␣C5↲
C2␣C3␣C4␣C5↲
C3␣C4␣C5␣C2↲
C4␣C5␣C2␣C3↲
C5␣C2␣C3␣C4↲
C2␣C3␣C4␣C5↲
C3␣C4␣C5↲
C4␣C5␣C3↲
C5␣C3↲
C3↲
C3↲

実習課題の提出方法と締切

- **手順**
  1. 作成したプログラムをレポートシステムから
     アップロードする
  2. レポートシステム上で実行結果がACと表示されたら、
     番号札を取ってTAの試問を受ける
  3. TAによる試問に合格したら完了
- **本日 16:20までに実行結果がACになる(A課題 締切)**
  - 本日 16:20を過ぎると遅刻の減点とする
    （出さないとその課題は0点となるので出した方が良い）
  - TAの試問の締切は特にないが、今日中が望ましい
- **7/17 12:00までに実行結果がACになる (B課題 締切)**
  - 7/17 12:00を過ぎると遅刻の減点とする
    （出さないとその課題は0点となるので出した方が良い）
  - TAの試問は今日中もしくは7/17実習授業にて

レポートシステム

http://cs-www.edu.c.titech.ac.jp:4040/toj

プログラム提出後、実行結果がACになったら
正しい動作が確認できたということ。

ホーム　課題　レポート提出

タイトル	締め切り	提出時刻	実行結果	TAチェック	
ex00-1	2017-04-07 16:20:00	2017-04-13 03:02:47	AC	未チェック	提出する
ex00-2	2017-04-07 16:20:00	2017-04-13 03:02:58	AC	未チェック	提出する
ex1-1	2017-04-13 16:20:00	2017-04-13 03:03:08	AC		
ex1-2	2017-04-13 16:20:00	2017-04-13 03:03:16	AC	未チェック	提出する
ex1-3	2017-04-13 16:20:00	2017-04-13 03:03:26	AC	未チェック	提出する
ex1-4	2017-04-13 16:20:00	2017-04-13 03:03:34	WA		

ACはプログラムの動作が正しいことを表す

WAはプログラムの動作が正しくないことを表す

TAの試問を受ける前に

- **以下のようなケースはTAから合格がもらえません**
  - プログラムの先頭のコメントが無い

```
/*  h-ex4-1
    tokodai.t.aa
    Taro Tokodai          */
```

  - プログラム中に適切にコメントが入っていない
  - 変数名が日本語(ローマ字)、1文字(慣習的に使うものを除く)
  - インデントがズレていたり規則が一貫していない
  - -Wallを付けてコンパイルしたときに警告が出る
  - main関数の書き方が講義第1回で指示されたものでない

```
int main(void){
  XXXXX;
  YYYYY;
  return 0;
}
```