



ce-9. ポインタ, 連結リスト

(Cプログラミング応用, 全14回)

<https://www.kkaneko.jp/cc/c/index.html>

金子邦彦

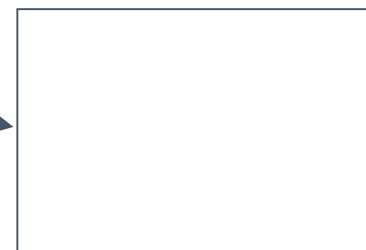


今まで説明してきた変数



```
#include "stdio.h"
#include <math.h>
#pragma warning(disable:4996)
int main()
{
    double x;
    double y;
    char buf[256];
    int i;
    double start_x;
    double step_x;
    FILE* fp;
    printf( "start_x = " );
    fgets( buf, 256, stdin );
    sscanf_s( buf, "%lf%n", &start_x );
    printf( "step_x = " );
    fgets( buf, 256, stdin );
    sscanf_s( buf, "%lf%n", &step_x );
    fp = fopen( "d:¥¥data.csv", "w" );
    for( i = 0; i < 20; i++ ) {
        x = start_x + ( i * step_x );
        y = sin( x );
        printf( "x= %f, y= %f¥n", x, y );
        fprintf( fp, "x=, %f, y=, %f¥n", x, y );
    }
    fprintf( stderr, "file z:¥¥data.csv created¥n" );
    fclose( fp );
    return 0;
}
```

プログラムが使う メモリ空間



メイン関数の
実行時に
自動的に確保

今日の内容



```
#include "stdio.h"  
#include <math.h>  
int main()  
{
```

new . . .

new の実行によりメモリアドレス
が得られる

プログラムが使う
メモリ空間

new を使い
必要な分を確保



今日の内容



```
#include "stdio.h"  
#include <math.h>  
int main()  
{
```

new . . .

delete . . .

不要になったら delete を実行
するのがルール

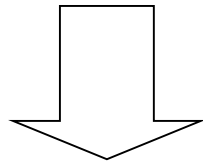
プログラムが使う メモリ空間

~~new を使い
必要な分を確保~~



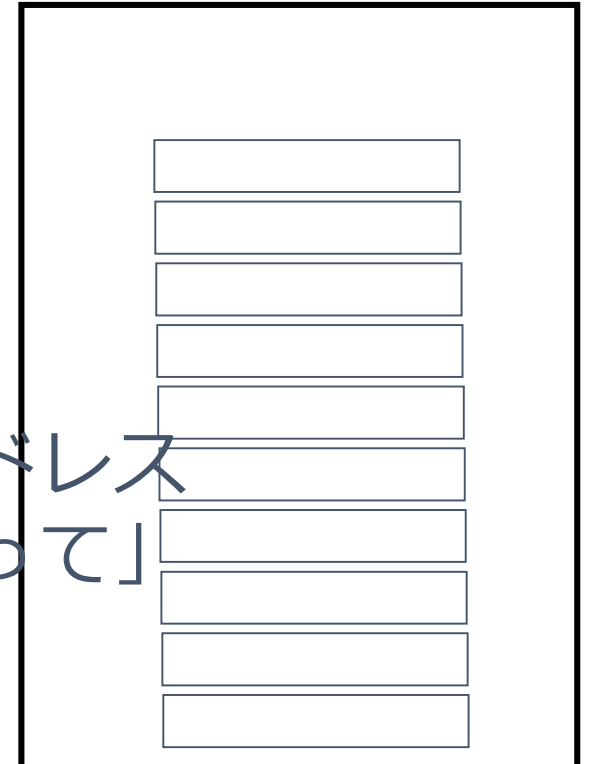


例えば, new を 10 回実行すると . . . プログラムが使う
メモリ空間



プログラミングテクニック

確保して得られた 10 個のメモリアドレス
を, メモリの「どこに」, 「どうやって」
格納しておくのか?

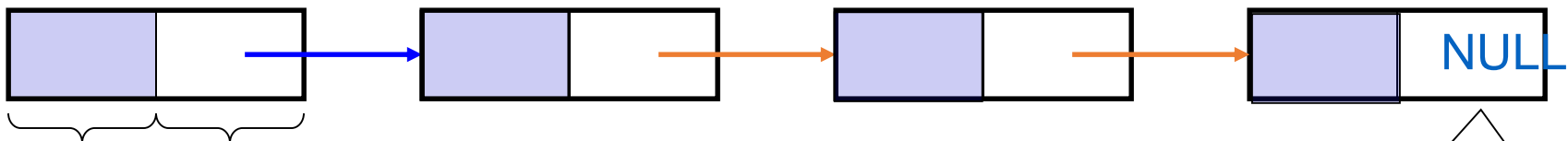


メモリエリアが
10 回確保される

リストの模式図



データ部分とポインタ部分で 1 単位を構成する
(これをリストセルと呼ぶ)



データ部分 ポインタ部分

(= 次のリストセル
のメモリアドレス)

リストの末端を表
す

メモリ中では
[00 00 00 00] という
値をとる

リストセル

リストの基本操作の例



- リストの生成 (new_list)

生成前 何も無い \rightarrow 生成後  サイズ1
のリスト

- リストの追加 (insert_list)

生成前 長さ k のリスト \rightarrow 生成後 長さ $k+1$ のリスト
($k \geq 0$) (先頭に追加)

- リストの削除 (delete_list)

生成前 長さ k のリスト \rightarrow 生成後 長さ $k-1$ のリスト
($k \geq 1$) (ある特定要素値を持つものを削除)

- リストの探索 (find_list)

リスト内から、ある特定要素値を持つものを探索



再帰的構造体：リストの定義の例

```
struct data_list {  
  
    int data;  
  
    struct data_list *next;  
  
};
```

data	next
int 型のデータ	メモリアドレス



リストの生成を行う関数

プログラムが使う
メモリ空間

```
struct data_list *new_list(int x)
```

```
{
```

確保したメモリエリアの
アドレスが c にセットされる

```
struct data_list *c = new(struct data_list);
```

```
c->data = x;
```

```
c->next = NULL;
```

```
return c;
```

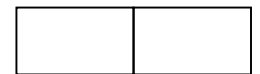
```
}
```

data に x の値を,
next に NULL をセット

c に入っているメモリアドレスを使い,
構造体にアクセス

メモリエリア
を確保

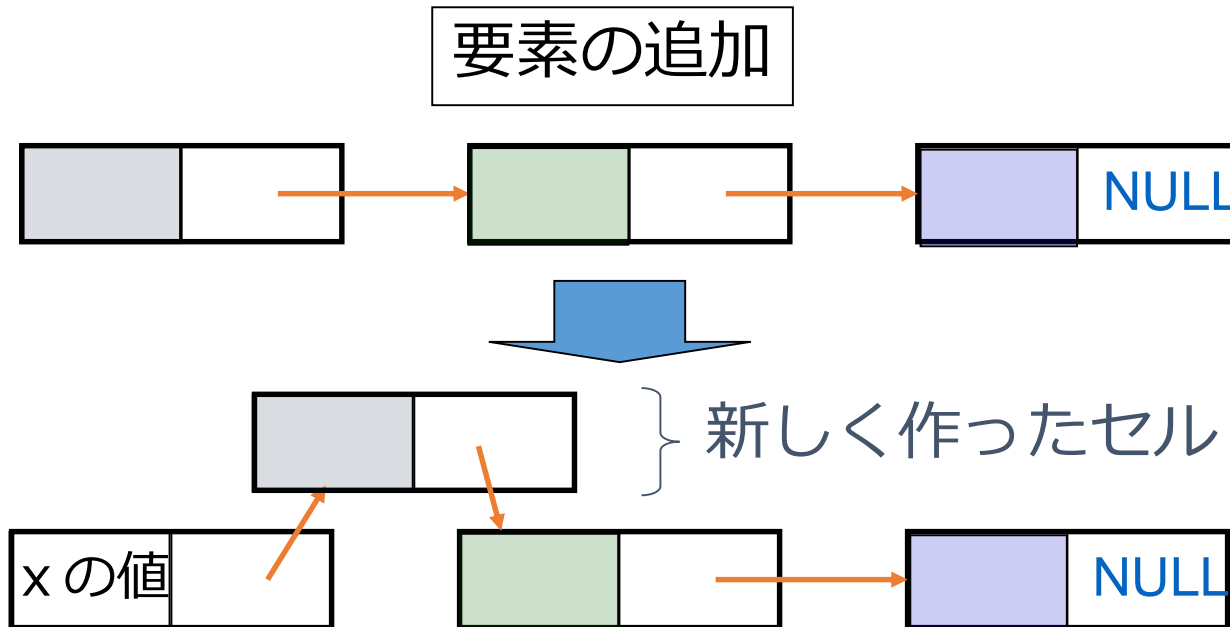
data next



リストの追加を行う関数



```
void insert_list(struct data_list *p, int x)
{
    struct data_list *c = new(struct data_list);
    c->data = p->data;
    c->next = p->next;
    p->data = x;
    p->next = c;
}
```



下記のような、サイズ4のリストを作るプログラム





new_list(89)



このメモリアドレス = a



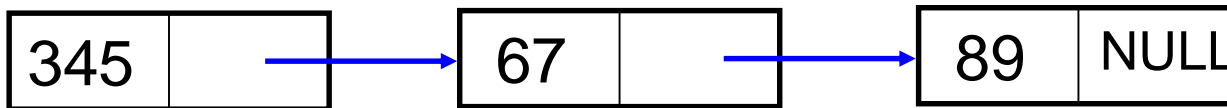
insert_list(a, 67)



このメモリアドレス = a



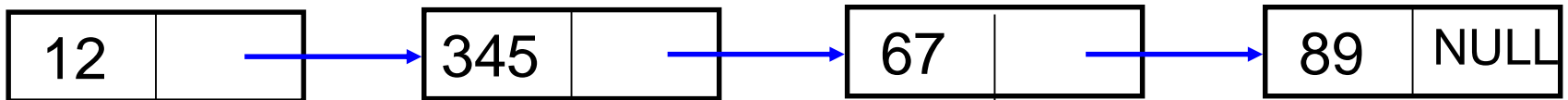
insert_list(a, 345)



このメモリアドレス = a



insert_list(a, 12)



このメモリアドレス = a



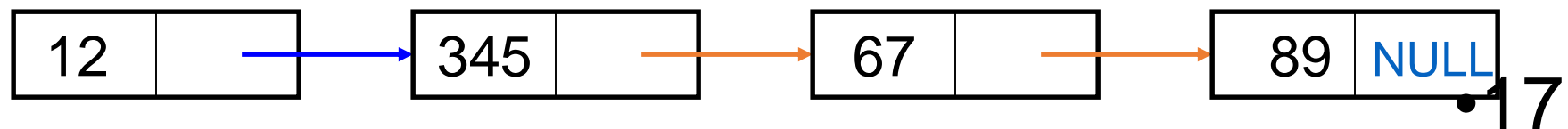
```
#include "stdio.h"
#include <math.h>
struct data_list {
    int data;
    struct data_list *next;
};
struct data_list *new_list(int x)
{
    struct data_list *c = new(struct data_list);
    c->data = x;
    c->next = NULL;
    return c;
}
void insert_list(struct data_list *p, int x)
{
    struct data_list *c = new(struct data_list);
    c->data = p->data;
    c->next = p->next;
    p->data = x;
    p->next = c;
}
int main()
{
    int ch;
    struct data_list *a;
    a = new_list( 89 );
    insert_list( a, 67 );
    insert_list( a, 345 );
    insert_list( a, 12 );
    printf( "Enter キーを1,2回押してください。プログラムを終了します\n");
    ch = getchar();
    ch = getchar();
    return 0;
}
```


実際のメモリの中身



```
16 進数でメモリの中身を表示
: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f | 0123456789abcdef
-----
003710B0 : c 0 0 0 88 11 37 0 | 12, ポインタ | .....7.....
003710C0 : ab ab ab ab ee fe ee fe | .....
003710D0 : 9 0 9 0 9b 7 1c 0 | .....7.0+7.
003710E0 : 0 0 0 0 0 0 0 0 | .....
003710F0 : 2f 0 0 0 fd fd fd fd | 59 0 0 0 0 0 0 0 | 89, NULL | .....
00371100 : fd fd fd fd ab ab ab ab | .....
00371110 : 0 0 0 0 0 0 0 0 | .....
00371120 : 30 2b 37 0 68 11 37 0 | .....0+7.h.7.....
00371130 : 8 0 0 0 1 0 0 0 | .....1.....
00371140 : 43 0 0 0 f8 10 37 0 | 67, ポインタ | .....7.....
00371150 : ab ab ab ab ee fe ee fe | .....
00371160 : 9 0 9 0 ad 7 1c 0 | .....7.p;7.
00371170 : 0 0 0 0 0 0 0 0 | .....
00371180 : 32 0 0 0 fd fd fd fd | 59 1 0 0 40 11 37 0 | 345, ポインタ | .....
00371190 : fd fd fd fd ab ab ab ab | .....
003711A0 : 0 0 0 0 0 0 0 0 | .....K.....
```

16進数表示であることに注意



構造体リストの長所



- 動的メモリ管理(new)により、必要な分だけのメモリを、好きなときに得られる。
 - セル数の制限を気にしなくてよい
- 削除したセルの再利用も簡単化できる。
 - ごみ集めの自動化が可能

例題 1 . リストの要素の表示



- リストの個々の要素をたどり，要素値を順に表示するプログラム

リストの表示を行う関数



```
void print_list(struct data_list *p)
{
    struct data_list *x;
    if ( p == NULL ) {
        return;
    }
    printf( "%d", p->data );
    x = p->next;
    while( x != NULL ) {
        printf( ", %d", x->data );
        x = x-> next;
    }
    printf( "¥n" );
}
```



```
#include "stdio.h"
#include <math.h>
struct data_list {
    int data;
    struct data_list *next;
};

void print_list(struct data_list *p)
{
    struct data_list *x;
    if ( p == NULL ) {
        return;
    }
    printf( "%d", p->data );
    x = p->next;
    while( x != NULL ) {
        printf( ", %d", x->data );
        x = x->next;
    }
    printf( "¥n" );
}

struct data_list *new_list(int x)
{
    struct data_list *c = new(struct data_list);
    c->data = x;
    c->next = NULL;
    return c;
}

void insert_list(struct data_list *p, int x)
{
    struct data_list *c = new(struct data_list);
    c->data = p->data;
    c->next = p->next;
    p->data = x;
    p->next = c;
}

int main()
{
    int ch;
    struct data_list *a;
    a = new_list( 89 );
    insert_list( a, 67 );
    insert_list( a, 345 );
    insert_list( a, 12 );
    print_list( a );
    printf( "Enter キーを1,2回押してください。プログラムを終了します¥n");
    ch = getchar();
    ch = getchar();
    return 0;
}
```



実行結果の例

```
12, 345, 67, 89  
Enter キーを1,2回押してください。プログラムを終了します
```

例題 2. リストの探索

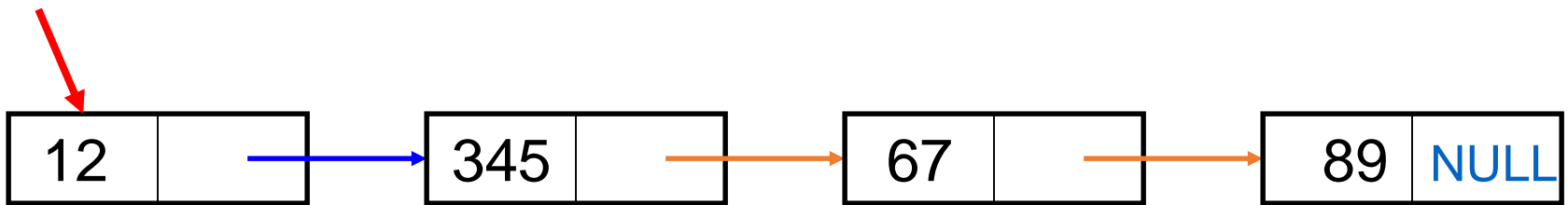


- リストの個々の要素をたどり，要素による探索を行うプログラム

リストの探索



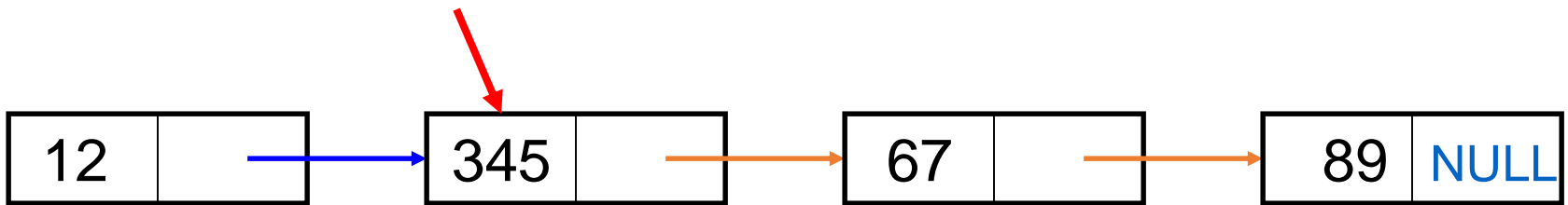
find_list(67)



リストの探索



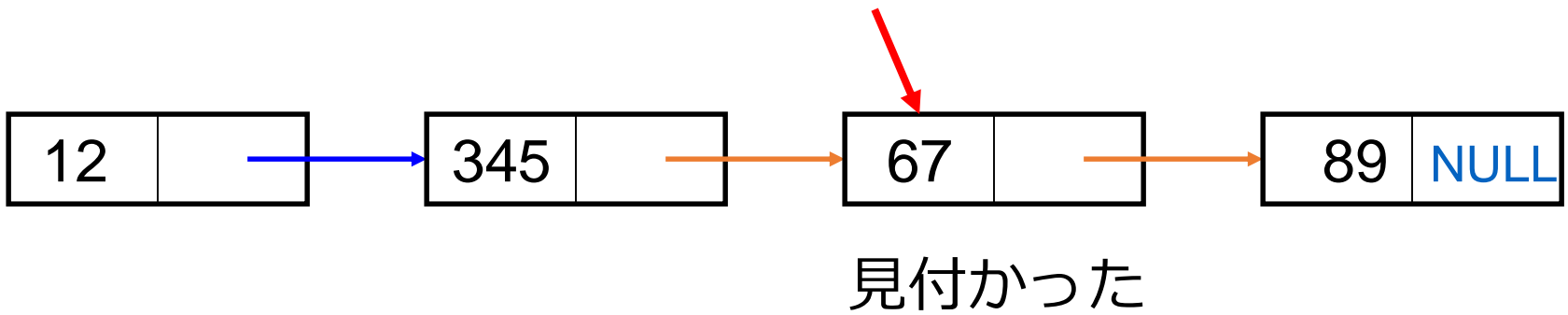
find_list(67)



リストの探索



find_list(67)



リストの探索を行う関数



```
int find_list(struct data_list *p, int data )
{
    struct data_list *x;
    x = p;
    while( x != NULL ) {
        if ( x->data == data ) {
            return 1;
        }
        x = x-> next;
    }
    return 0;;
}
```



```
#include "stdio.h"
#include <math.h>
struct data_list {
    int data;
    struct data_list *next;
};

#include "stdio.h"
int find_list(struct data_list *p, int data )
{
    struct data_list *x;
    x = p;
    while( x != NULL ) {
        if ( x->data == data ) {
            return 1;
        }
        x = x-> next;
    }
    return 0;;
}

struct data_list *new_list(int x)
{
    struct data_list *c = new(struct data_list);
    c->data = x;
    c->next = NULL;
    return c;
}

void insert_list(struct data_list *p, int x)
{
    struct data_list *c = new(struct data_list);
    c->data = p->data;
    c->next = p->next;
    p->data = x;
    p->next = c;
}

int main()
{
    int ch;
    struct data_list *a;
    a = new_list( 89 );
    insert_list( a, 67 );
    insert_list( a, 345 );
    insert_list( a, 12 );
    if ( find_list( a, 65 ) ) {
        printf( "65はリスト中にある\n" );
    }
    if ( find_list( a, 67 ) ) {
        printf( "67はリスト中にある\n" );
    }
    printf( "Enter キーを1,2回押してください。プログラムを終了します\n");
    ch = getchar();
    ch = getchar();
    return 0;
}
```



例題 3 . リストの削除

- リストの個々の要素をたどり, 要素の削除を行うプログラム

リストの削除



delete_list(67)



リストの削除



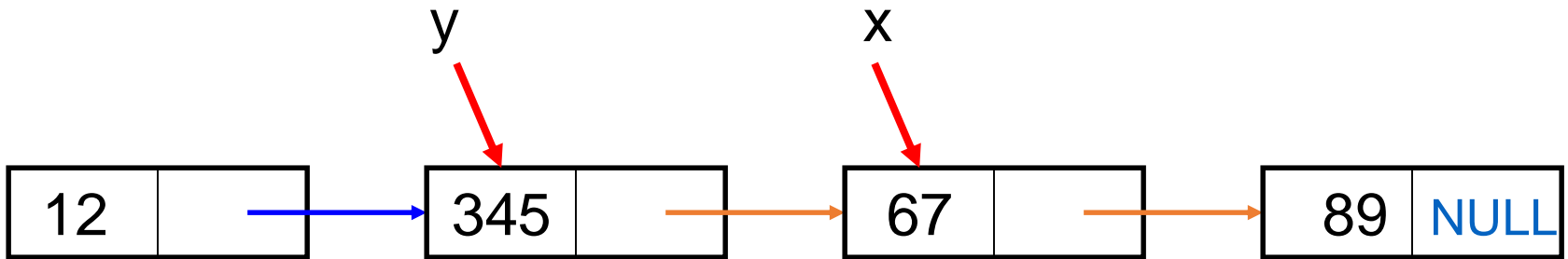
delete_list(67)



リストの削除



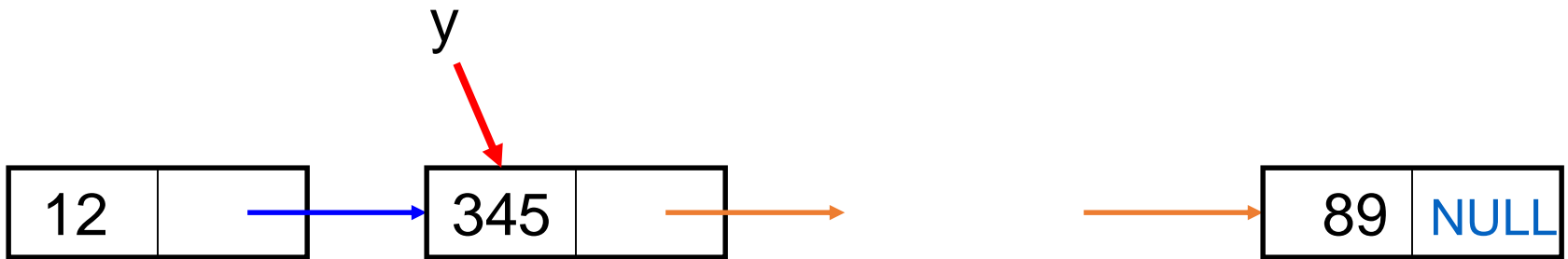
delete_list(67)



リストの削除



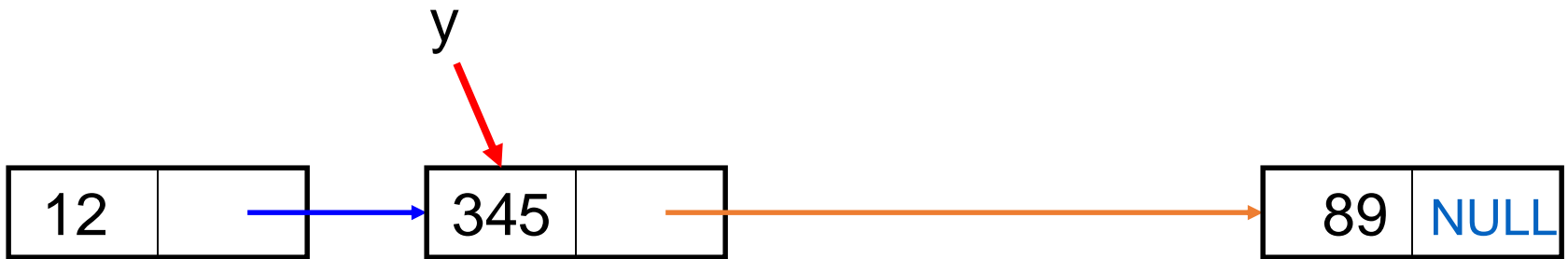
delete_list(67)



リストの削除



delete_list(67)



リストの削除を行う関数



```
void delete_list(struct data_list **p, int data )
{
    struct data_list *x;
    struct data_list *y;
    x = *p;
    y = NULL;
    while( x != NULL ) {
        if ( x->data == data ) {
            if ( y != NULL ) {
                y->next = x->next;
                delete( x );
                break;
            }
            else {
                (*p) = x->next;
                delete( x );
                break;
            }
        }
        y = x;
        x = x-> next;
    }
    return;
}
```



```
#include "stdio.h"
#include <math.h>
struct data_list {
    int data;
    struct data_list *next;
}

void delete_list(struct data_list **p, int data )
{
    struct data_list *x;
    struct data_list *y;
    x = *p;
    y = NULL;
    while( x != NULL ) {
        if( x->data == data ) {
            if( y == NULL ) {
                y->next = x->next;
                delete( x );
                break;
            }
            else {
                (*p) = x->next;
                delete( x );
                break;
            }
        }
        y = x;
        x = x-> next;
    }
    return;
}

void print_list(struct data_list *p)
{
    struct data_list *x;
    if( p == NULL ) {
        return;
    }
    printf( "%d", p->data );
    x = p->next;
    while( x != NULL ) {
        printf( ", %d", x->data );
        x = x-> next;
    }
    printf( "\n" );
}

struct data_list *new_list(int x)
{
    struct data_list *c = new(struct data_list);
    c->data = x;
    c->next = NULL;
    return c;
}

void insert_list(struct data_list *p, int x)
{
    struct data_list *c = new(struct data_list);
    c->data = p->data;
    c->next = p->next;
    p->data = x;
    p->next = c;
}

int main()
{
    int ch;
    struct data_list *a;
    a = new_list( 89 );
    insert_list( a, 67 );
    insert_list( a, 345 );
    insert_list( a, 12 );
    print_list( a );
    delete_list( &a, 67 );
    print_list( a );
    printf( "Enter キーを1,2回押してください。プログラムを終了します\n");
    ch = getchar();
    ch = getchar();
    return 0;
}
```

再帰的構造体を用いた実装法



- データとポインタをメンバとする構造体を定義する。
- リストの末端はNULL（空ポインタ）
- 先頭の要素セルをおさえるためのポインタ変数を使う（例題 1, 例題 2, 例題 3 でのメイン関数内での変数 a）
 - 最後尾の要素セルをおさえるためのポインタ変数が役立つ場合もある（次回授業でのリストの併合のケース）
- 必要に応じて途中のセルをおさえるためのポインタ変数を適宜用意

実習



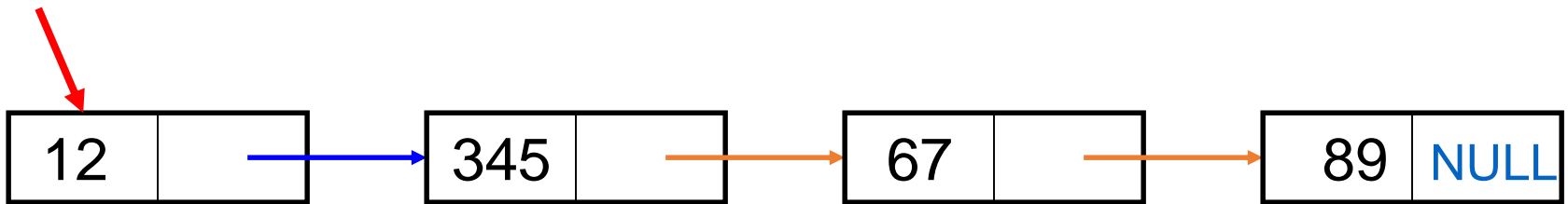
- リストの長さを求める関数を作成しなさい

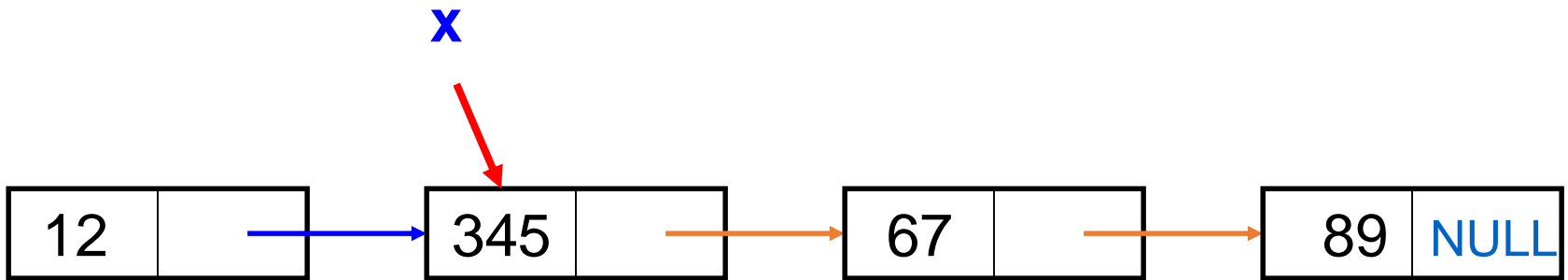
lengthの実行状況

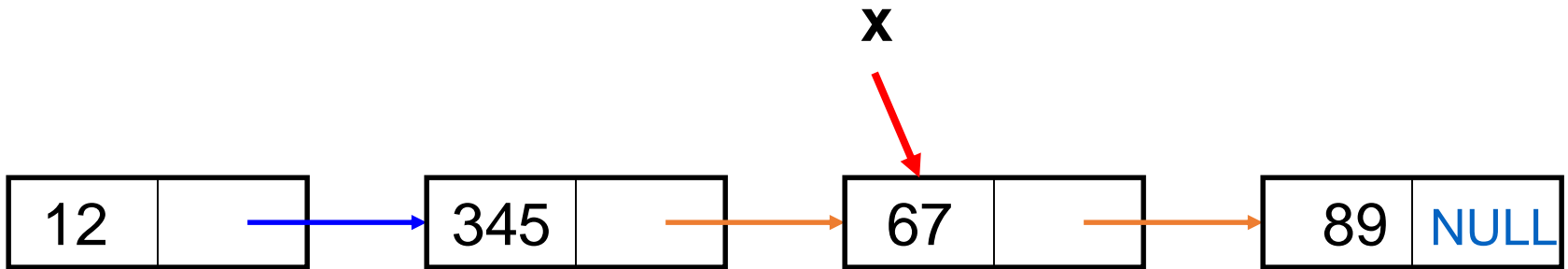


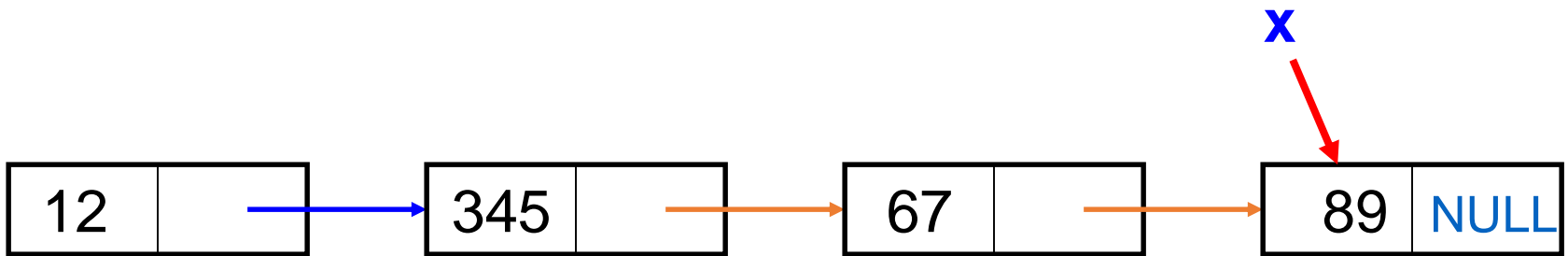
最初

x











結果：4