
プログラミング応用演習 I

第4週
リスト

データ構造 Structure of Data / リスト List

- リスト / List

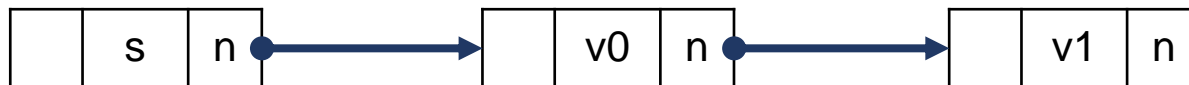
- データを保存するためのデータ構造(スタック、キューと同様)
 - 順番があって並べられたデータを表す
 - データの挿入、削除が容易にできる
 - データを決められた長さの配列(配列やベクトル)のデータを持たない
 - データの繋がりを持つものを「連結リスト」と呼ぶ
-
- Data structures for storing data (similar to stack and queue)
 - Representing ordered data
 - Easy insertion and deletion of data
 - No array type data with fixed length (such as array or vector)
 - It is called “Linked list”

データ構造 Structure of Data / リスト List

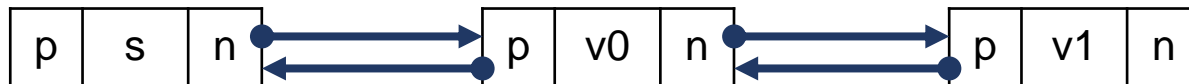
- リストの種類

p: prev
n: next

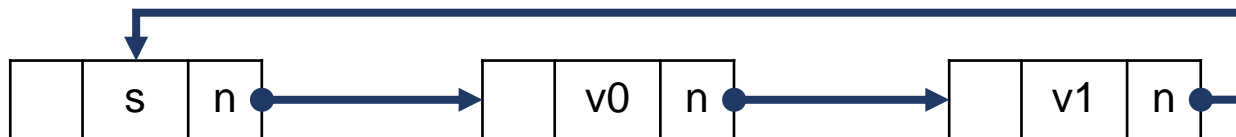
One-way list
単方向リスト



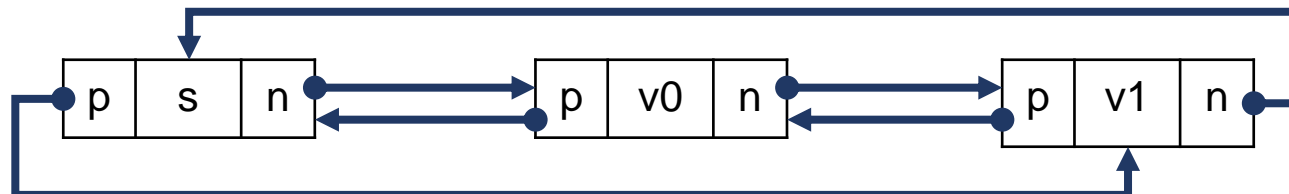
Two-way list
双方向リスト



One-way circulation list
単方向循環リスト



Two-way circulation list
双方向循環リスト



データ構造 Structure of Data / リスト List

• リストの種類

p: prev

One-way list
単方向リスト

- 一方向のみに要素がつながっているデータ
- 次の要素へのポインターを持つ
- 前の要素へは戻れないので、前の要素が必要な場合は先頭から探す必要がある

Two-way list
双方向リスト

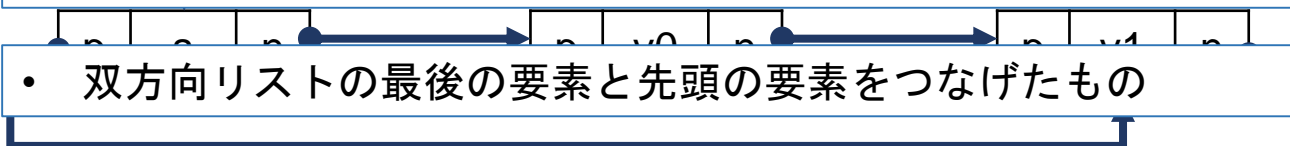
- 前後方向に要素がつながっているデータ
- 次の要素と前の要素へのポインターを持つ
- 前方、後方どちらへも容易に参照を行うことができる

One-way circulation list
単方向循環リスト

- 単方向リストと同じく、一方向のみに要素がつながっている
- 最後の要素の次の要素として先頭につなげる
- 要素の順番はあるが、先頭から何番目といった意味が薄い
- 全要素を参照する場合は、常に次に進んで開始時と同じ要素に到達すると終了する

Two-way circulation list
双方向循環リスト

- 双方向リストの最後の要素と先頭の要素をつなげたもの



データ構造 Structure of Data / リスト List

• リストの種類

p: prev

One-way list
単方向リスト

- Elements connected in only one direction
- Elements have a pointer to the next element
- You cannot refer the previous element, so if you need the previous element, you need to search it from the first.

Two-way list
双方向リスト

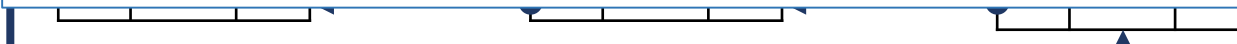
- Elements connected in two direction
- Elements have pointers to the next and the previous elements
- Easy to reference either forward or backward

One-way circulation list
単方向循環リスト

- Elements are connected in only one direction(same as one-way list)
- Connecting to the first element as the next element of the last element
- For referring to all elements, continuously move to the next and finish searching when the same element as at the start is reached

Two-way circulation list
双方向循環リスト

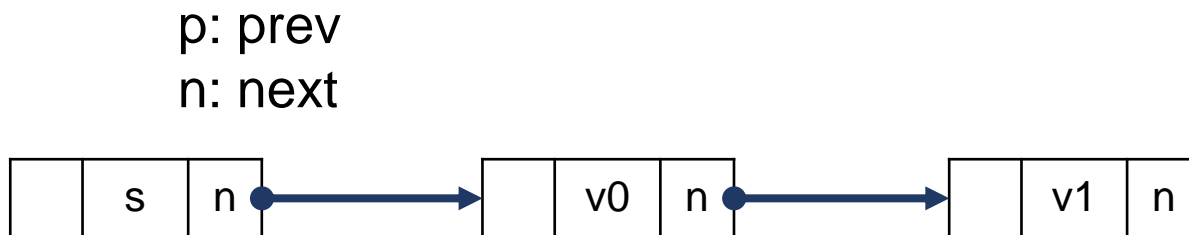
- Connecting to the first element as the next element of the last element (for two-way list)



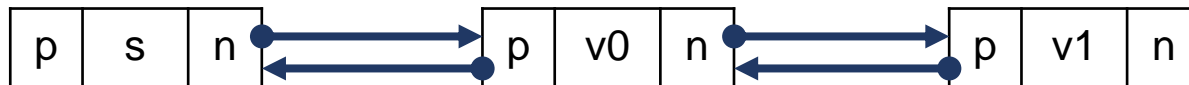
データ構造 Structure of Data / リスト List

• リストの種類

One-way list
単方向リスト



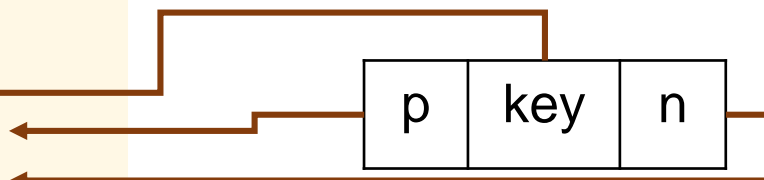
Two-way list
双方向リスト



- リンクの要素をノードとして表す
- 次のノード、前のノードを持つ
- Keyがデータ要素

Declaration of struct

```
/* structure */  
typedef struct node {  
    int key;  
    struct node *prev;  
    struct node *next;  
} Node;
```



Description of Source Code (List)

- Source code is lecture_4_1.c

```
/* header files */
#include <stdio.h>
#include <stdlib.h>

/* define */
#define STR_MAX 256

/* structure */
typedef struct node {
    int key;
    struct node *prev;
    struct node *next;
} Node;

/* functions */
Node *initialize(void);
void insertNextTo(Node *x, int in_key);
Node *searchNodeByKey(Node *x, int key);
void deleteKey(Node *x, int del_key);
void print_list(Node *x);
```

Assignment 4.1 / Hint

単方向リストにするため prev を削除
For making one-way list, delete prev.

Description of Source Code (List)

```
sentinel_node = initialize();  
while(1) {  
    puts("Enter the operation number.");  
    printf("1. Insert, 2. Delete, 3. Print, 4. Exit: ");  
    fgets(str, sizeof(str), stdin);  
    num = atoi(str); /* */  
    /* */  
    switch(num) {  
    case 1:  
        printf("Input a key to insert: ");  
        fgets(str, sizeof(str), stdin);  
        num = atoi(str);  
        /* assignment 4.2 */  
        insertNextTo(sentinel_node, num);  
        break;  
    case 2:  
        printf("Input a key to delete: ");  
        fgets(str, sizeof(str), stdin);  
        num = atoi(str);  
        deleteKey(sentinel_node, num);  
        break;  
    case 3: print_list(sentinel_node);  
            break;
```

先頭のみ値を持たず先頭であることを示すノード(sentinel_node)となっている

Enter (read)
Operation number

Assignment 4.2 / Hint
データを入れるための基準のキーを問い合わせる
Query the key to insert new data


Switch Operation

Description of Source Code (List)

- Source code is lecture_4_1.c

```
Node *initialize(void) {
    Node *x;
    if ((x = (Node *)malloc(sizeof(Node))) == NULL) {
        puts("No resource remained.");
        exit(1);
    }
    x->prev = x;
    x->next = x;
    return x;
}
```

Assignment 4.1



```
void insertNextTo(Node *x, int in_key) {
    Node *ins;
    ins = initialize();
    /* */
    ins->key = in_key;
    ins->prev = x;
    ins->next = x->next;
    /* */
    x->next->prev = ins;
    x->next = ins;
}
```

Description of Source Code (List)

- Source code is lecture_4_1.c

```
/* x should be sentinel_node */
Node *searchNodeByKey(Node *x, int key) {
    Node *search = x->next; // the target for deletion
    // stop looping if the search is <x> or the target
    while(!(search == x || search->key == key)) {
        search = search->next;
    }
    return search;
}
```

Assignment 4.1

For inserting node, we need to know the previous node of required node

```
/* x should be sentinel_node */
void deleteKey(Node *x, int del_key) {
    Node *search = searchNodeByKey(x, del_key);
    if (search == x) {
        printf("%d is not found.\n", del_key);
    } else {
        search->prev->next = search->next;
        search->next->prev = search->prev;
        free(search);
    }
}
```

Assignment 4.1

Description of Source Code (List)

- Source code is lecture_4_1.c

```
/* x should be sentinel_node */
void print_list(Node *x) {
    Node *print = x->next;
    if (x->next == x) {
        puts("No data found.");
    } else {
        while(print != x) {
            printf("%d ", print->key);
            print = print->next;
        }
        putchar('\n');
    }
}
```

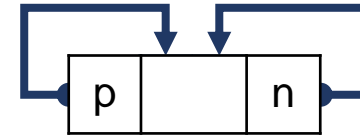
Description of Source Code (List) / Detail description

- Source code is lecture_4_1.c

```
Node *initialize(void) {  
    Node *x;  
    if ((x = (Node *)malloc(sizeof(Node))) == NULL) {  
        puts("No resource remained.");  
        exit(1);  
    }  
    x->prev = x;  
    x->next = x;  
    return x;  
}
```

```
void insertNextTo(Node *x, int in_key) {  
    Node *ins;  
    ins = initialize();  
    /* */  
    ins->key = in_key;  
    ins->prev = x;  
    ins->next = x->next;  
    /* */  
    x->next->prev = ins;  
    x->next = ins;  
}
```

Initialize()



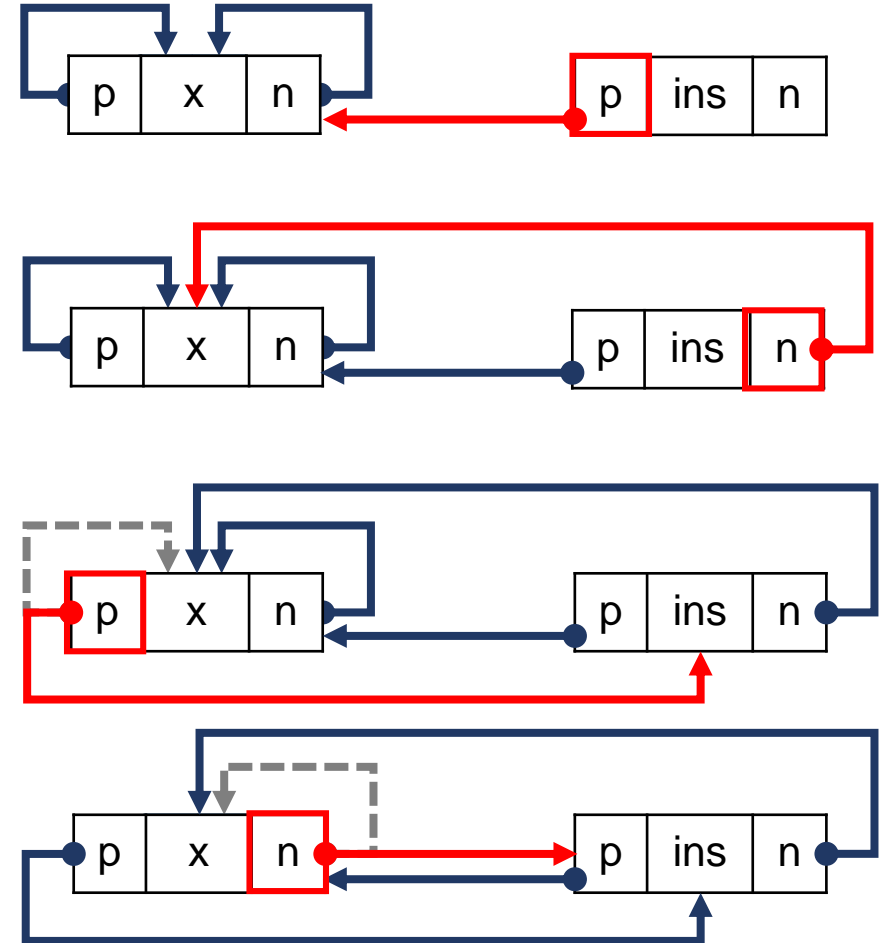
Description of Source Code (List) / Detail description

- Source code is lecture_4_1.c

```
Node *initialize(void) {  
    Node *x;  
    if ((x = (Node *)malloc(sizeof(Node))) == NULL) {  
        puts("No resource remained.");  
        exit(1);  
    }  
    x->prev = x;  
    x->next = x;  
    return x;  
}
```

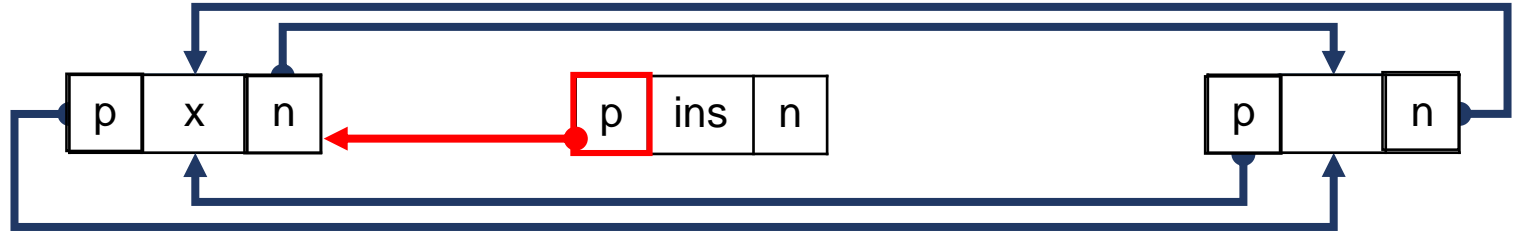
```
void insertNextTo(Node *x, int in_key) {  
    Node *ins;  
    ins = initialize();  
    /* */  
    ins->key = in_key;  
    ins->prev = x;  
    ins->next = x->next;  
    /* */  
    x->next->prev = ins;  
    x->next = ins;  
}
```

Insertion to node with initial condition

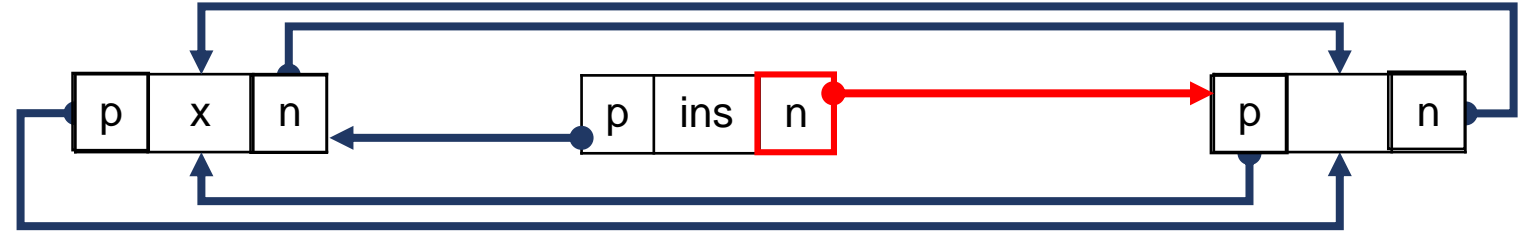


Description of Source Code (List) / Detail description

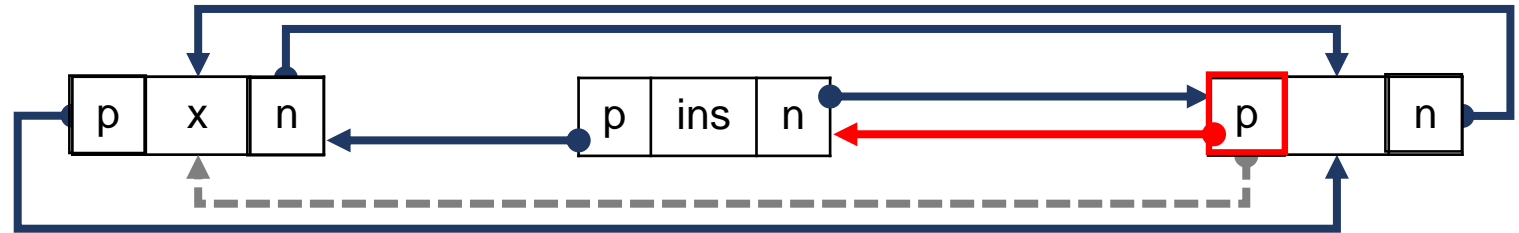
`ins->prev = x;`



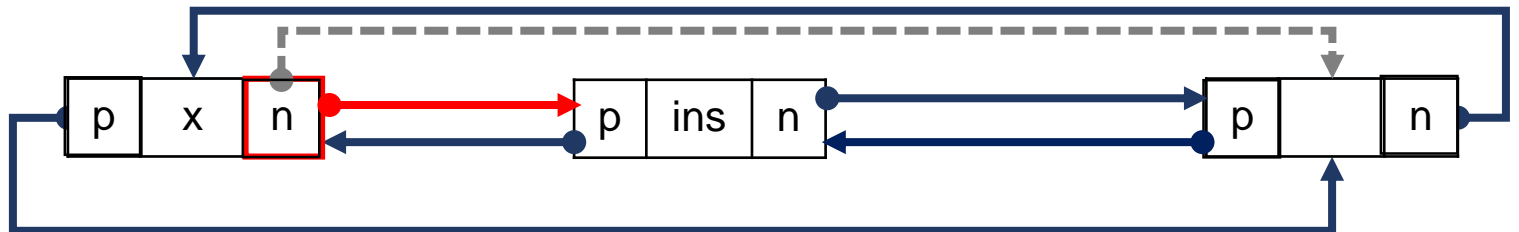
`ins->next =
x->next;`



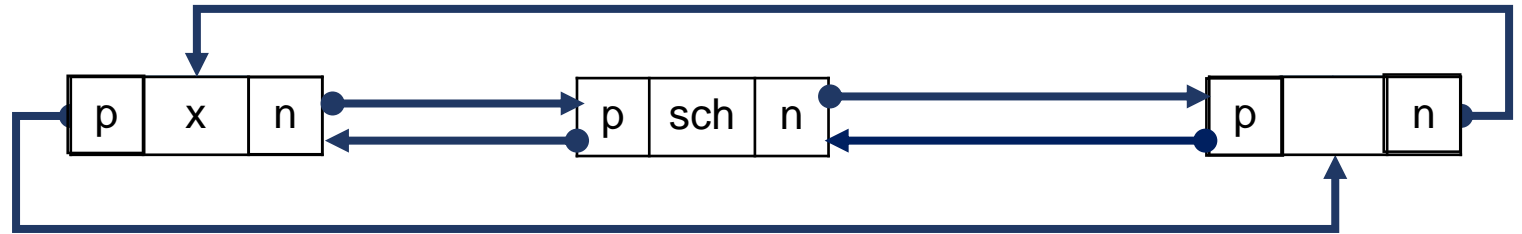
`x->next->prev =
ins;`



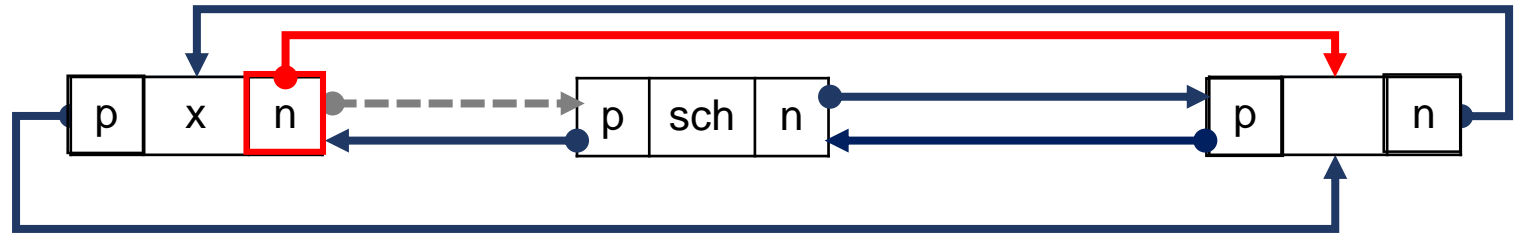
`x->next = ins;`



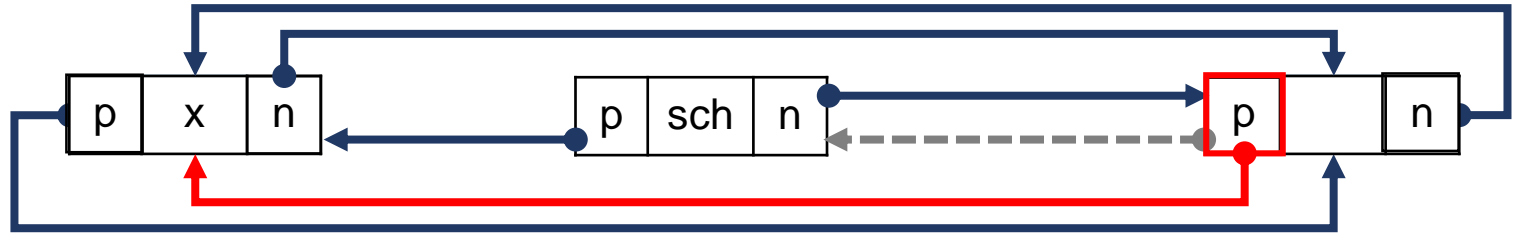
Description of Source Code (List) / Detail description



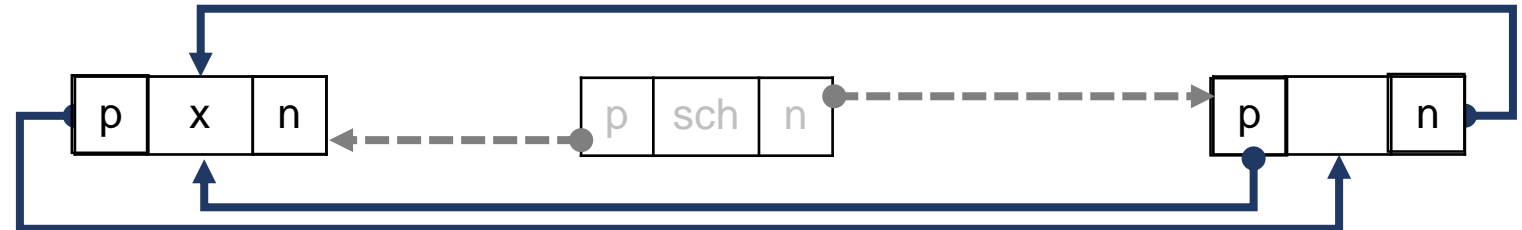
`search->prev->next =
search->next;`



`search->next->prev =
search->prev;`



`free(search);`



C言語の振り返り1 (C language / return value)

• 返り値とは

- 何らかの結果を関数実行後に使えるようにします
- void と宣言すると値は返らず、関数の中身だけ実行します
- 関数宣言の返り値型と return で返す型は同じにする必要があります

```
type_of_return_value function(arguments) {  
    statements (body of program)  
    return value; // type_of_return_value  
}
```

具体例/Example

```
double function(int x, int y) {  
    double res = (double)x / y;  
    return res;  
}
```

返り値の型がvoidの場合

Type of return value is void

```
void function(int x, int y) {  
    double res = (double)x / y;  
    printf("res = %f\n", res);  
    return;  
}
```

返り値をつけない
No return value

return無くても良い

```
double function(void) {  
    int x = 3;  
    int y = 4;  
    double res = (double)x / y;  
    return res;  
}
```

引数がないとき、参照できる変数なし
If there is no arguments, there is no available variable.

C言語の振り返り1 (C language / return value)

• 返り値とは

- 何らかの結果を関数実行後に使えるようにします
- void と宣言すると値は返らず、関数の中身だけ実行します
- 関数宣言の返り値型と return で返す型は同じにする必要があります

```
type_of_return_value function(arguments) {  
    statements (body of program)  
    return value; // type_of_return_value  
}
```

返り値の型がvoidの場合

Type of return value is void

```
void function(int x, int y) {  
    double res = (double)x / y;  
    printf("res = %f\n", res);  
    return;  
}
```

function(10, 3); // 関数の返り値を代入しない

具体例/Example

```
double function(int x, int y) {  
    double res = (double)x / y;  
    return res;  
}
```

double res = function(10, 3);

function(11, 7);
// 返り値は失われる

C言語の振り返り1 (C language / return value)

- Return value

- Let some result be available after the function is executed
- Declaring “void” will not return a value, only the function’s contents will be executed
- The function declaration and the type returned must be the same

```
type_of_return_value function(arguments) {  
    statements (body of program)  
    return value; // type_of_return_value  
}
```

具体例/Example

```
double function(int x, int y) {  
    double res = (double)x / y;  
    return res;  
}
```

戻り値の型がvoidの場合

Type of return value is void

```
void function(int x, int y) {  
    double res = (double)x / y;  
    printf("res = %f\n", res);  
    return;  
}
```

戻り値をつけない
No return value

return無くても良い

```
double function(void) {  
    int x = 3;  
    int y = 4;  
    double res = (double)x / y;  
    return res;  
}
```

引数がないとき、参照できる変数なし
If there is no arguments, there is no available variable.

C言語の振り返り1 (C language / return value)

- Return value

- Let some result be available after the function is executed
- Declaring “void” will not return a value, only the function’s contents will be executed
- The function declaration and the type returned must be the same

```
type_of_return_value function(arguments) {  
    statements (body of program)  
    return value; // type_of_return_value  
}
```

返り値の型がvoidの場合

Type of return value is void

```
void function(int x, int y) {  
    double res = (double)x / y;  
    printf(“res = %f¥n”, res);  
    return;  
}
```

```
function(10, 3); // Do not use function  
return value
```

具体例/Example

```
double function(int x, int y) {  
    double res = (double)x / y;  
    return res;  
}
```

```
double res = function(10, 3);
```

```
function(11, 7);  
// The return value is lost
```

C言語の振り返り2 (C language / increment)

• インクリメント演算子

- 前につくのと後ろにつくので意味が変わるので注意すること
- 前につくと 値を変えた**後の**値が得られる(**変数の値は変わる**)
- 後ろにつくと 値を変える**前の**値が得られる(**変数の値は変わる**)

```
int cnt = 0;
cnt++; // cnt = cnt + 1; / cnt += 1;
++cnt; // cnt = cnt + 1; / cnt += 1;
cnt--; // cnt = cnt - 1; / cnt -= 1;
--cnt; // cnt = cnt - 1; / cnt -= 1;
```

```
int counter = 0;
int tmp = 0;

counter++;
printf("tmp = %d, counter = %d\n", tmp, counter);

tmp = counter++;
printf("tmp = %d, counter = %d\n", tmp, counter);

tmp = ++counter;
printf("tmp = %d, counter = %d\n", tmp, counter);

tmp = 0, counter = 1
tmp = 1, counter = 2
tmp = 3, counter = 3
```

C言語の振り返り2 (C language / increment)

- Increment operator
- Note that the meaning changes as it is attached to the front or to the back.
 - If put in front, you get the value **after** changing the value (the value changes)
 - If put in back, you get the value **before** changing the value (the value changes)

```
int cnt = 0;
cnt++; // cnt = cnt + 1; / cnt += 1;
++cnt; // cnt = cnt + 1; / cnt += 1;
cnt--; // cnt = cnt - 1; / cnt -= 1;
--cnt; // cnt = cnt - 1; / cnt -= 1;
```

```
int counter = 0;
int tmp = 0;

counter++;
printf("tmp = %d, counter = %d\n", tmp, counter);

tmp = counter++;
printf("tmp = %d, counter = %d\n", tmp, counter);

tmp = ++counter;
printf("tmp = %d, counter = %d\n", tmp, counter);

tmp = 0, counter = 1
tmp = 1, counter = 2
tmp = 3, counter = 3
```

C言語の振り返り3 (C language / increment)

- if / for / while の後に文について
 - 一つの文(statement)が実行されます
 - 波括弧でくくられた範囲は一つの文です

```
if (a == 0) x = 0; else x = 1;
```

```
if (a == 0)  
x = 0;
```

```
if (a == 0)  
x = 0;  
x = 1;
```

← Ifの次の文(常に実行される)
Next statement

```
if (a == 0) {  
x = 0;  
x = 1;  
}
```

```
if ( <Boolean> ) {  
    // body  
}
```

```
While ( <Boolean> ) {  
    // body  
}
```

```
For (<init> ; <Boolean>; <next> )  
{  
    // body  
}
```

C言語の振り返り3 (C language / increment)

- Statements after if / for / while
 - Single statement will be executed
 - The range enclosed in curly brackets is a single statement

```
if (a == 0) x = 0; else x = 1;
```

```
if (a == 0)  
x = 0;
```

```
if (a == 0)  
x = 0;  
x = 1;
```

← Ifの次の文(常に実行される)
Next statement

```
if (a == 0) {  
x = 0;  
x = 1;  
}
```

```
if ( <Boolean> ) {  
    // body  
}
```

```
While ( <Boolean> ) {  
    // body  
}
```

```
For (<init> ; <Boolean>; <next> )  
{  
    // body  
}
```

C言語の振り返り4 (C language / size of type)

- 整数の範囲 (おおよそ10bit(1024)が 10^3)
 - char型 8bit 256 (-128 ~ 127)
 - short型 16bit 65536 (-32768 ~ 32767)
 - int型 32bit 4294967296 (-2147483648 ~ 2147483647)
 - long型 64bit 18446744073709551616 (-9223372036854775808 ~ 9223372036854775807)
 - long long 型 は long型と同じ (環境によって変わる)
 - ポインターは64bit (64bit CPU / 64bit OSだから)
- Sizeof演算子
 - sizeof(型名 または 変数名) で 型のサイズをバイト単位で得ることができる
 - 配列、構造体、ポインターに使うことができる
- pointer_quiz.c の最初に使い方の例を示している

C言語の振り返り4 (C language / size of type)

- Range of integer (10bit(1024) is nealy 10^3)
 - char 8bit 256 (-128 ~ 127)
 - short 16bit 65536 (-32768 ~ 32767)
 - int 32bit 4294967296 (-2147483648 ~ 2147483647)
 - long 64bit 18446744073709551616 (-9223372036854775808 – 9223372036854775807)
 - long long is the same size of long (depends on environment)
 - Size of pointer is 64bit (Because of 64bit CPU / 64bit OS)
- Sizeof operator
 - sizeof(type name or variable) / you can get the size in bytes
 - Can be used for arrays, structures, and pointers
- There are examples in pointer_quiz.c

変数のスコープ

• 変数のスコープ

- 変数はスコープ(有効な場所)を持ちます
- 基本は {} で囲われた内部で有効です
- main{ } の外(最も外のスコープ)に書くと大域変数(どこからでも使える変数)となります

```
/* header files */
#include <stdio.h>
/* define */
#define ARRAY_MAX 5

/* structure */
typedef struct queue {
    int array[ARRAY_MAX];
    int front;
    int rear;
} Queue;

/* static memory allocation */
Queue static_var;
// これ以降ならどこでも使える
/* main */
int main(void) {
```

```
/* main */
int main(void) {
    // local memory allocation
    Queue local_var; // main(void) { } の中でしか使えない

    {
        // local memory allocation
        Queue local_var2; // { } の中でしか使えない
        static_var.front = 1;
        local_var.front = 2;
        local_var2.front = 4;
    }
    static_var.front = 1;
    local_var.front = 2;
    local_var2.front = 4; // compile error
    return 0;
}
```

Scope of variables

- Scope of variables
 - Variables have a scope (valid location)
 - Inside the { } enclosure, variables are valid (can be refer)
 - If written outside of main{ } (the outermost scope), it becomes a global variable

```
/* header files */
#include <stdio.h>
/* define */
#define ARRAY_MAX 5

/* structure */
typedef struct queue {
    int array[ARRAY_MAX];
    int front;
    int rear;
} Queue;

/* static memory allocation */
Queue static_var;
// can be used anywhere after here
/* main */
int main(void) {
```

```
/* main */
int main(void) {
    // local memory allocation
    Queue local_var; // can be used in main(void) { }

    {
        // local memory allocation
        Queue local_var2; // can be used in { }
        static_var.front = 1;
        local_var.fronnd = 2;
        local_var2.front = 4;
    }
    static_var.front = 1;
    local_var.fronnd = 2;
    local_var2.front = 4; // compile error
    return 0;
}
```

ポインタークイズ / Quiz for pointer

- Source code in Google Classroom / `pointer_quiz.c`
- You can check your understandings by yourself
- Example of assignments
- `example_2_1.c`
 - Using global variable for Fibonacci numbers
- `example_2_2_b.c`
 - Using loop for factorize (slow but avoiding deep recursion)

C言語の振り返り(用語)

- (括弧(小括弧) parenthesis
- { 波括弧(中括弧) curly brace
- [角括弧(大括弧) square bracket
- : コロン colon
- ; セミコロン semi-colon
- < 小なり less-than
- > 大なり greater-than
- - ハイフン hyphen
- _ アンダーバー under-bar
- * アスタリスク asterisk
- @ アットマーク at (at mark)