

## 「C言語中級編」

### typedef

#### 課題 1

① typedef char MYTYPE[10];

② typedef char \*MYTYPE[10];

③

```
typedef struct abc {
```

```
    int x;
```

```
    int y;
```

```
} *MYTYPE;
```

#### 課題 2

① 8バイト

② 4バイト

## 構造体とデータのカプセル化

ヘッダファイル(stack.h)

```
typedef struct _STACK {
    int *stack_area;
    int stack_pointer;
} STACK;

void init_stack(STACK *stack, int *stack_area);
void push(STACK *stack, int data);
int pop(STACK *stack);
```

ライブラリファイル(stack.c)

```
#include <stdio.h>
#include <stack.h>
void init_stack(STACK *stack, int *stack_area)
{
    stack->stack_area = stack_area;
    stack->stack_pointer = 0;
}
void push(STACK *stack, int data)
{
    if (stack->stack_pointer >= STACK_SIZE) {
        puts("エラー スタックオーバーフロー");
        return;
    }
    stack->stack_area[stack->stack_pointer++] = data;
}
int pop(STACK *stack)
{
    if (stack->stack_pointer == 0) {
        puts("エラー データがありません");
        return 0;
    }
    return stack->stack_area[--stack->stack_pointer];
}
```

## メインファイル

```
#include <stdio.h>
#include <stack.h>
#define STACK_SIZE    100
void main()
{
    int stack_area[STACK_SIZE];    //スタック領域確保
    STACK stack;                  //スタック定義

    init_stack(&stack, stack_area); //スタック初期化

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    printf("pop data = %d\n", pop(&stack));
    printf("pop data = %d\n", pop(&stack));
    printf("pop data = %d\n", pop(&stack));
    printf("pop data = %d\n", pop(&stack));
}
```

## リスト構造

### 課題 1

```
LIST *find(int data)
{
    LIST *p;
    for (p = first; p != NULL; p = p->next) {
        if (p->value == data) {
            return p;
        }
    }
    return NULL;
}
```

## 課題 2

insert関数を次のように変更する。

またendは要らなくなる。first==NULLの場合の特別処理も要らない。

```
LIST *first = NULL;    //リストの最初

//リストにデータを挿入する
void insert(int data)
{
    LIST *p;
    p = (LIST *)malloc(sizeof(LIST));    //メモリ領域を確保

    //エラー処理
    if (p == NULL) {
        puts("エラー  メモリがありません");
        return;
    }

    //初期化
    p->value = data;

    //リストの最初につなげる
    p->next = first;
    first = p;
}
```

## 課題 3

```
void delete(void)
{
    LIST *p;
    if (first == NULL) {
        puts("エラー  要素がありません");
        return;
    }
    p = first;
    first = first->next;
    free((LIST *)p); //メモリ領域を解放
}
```

#### 課題 4

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

//リスト構造体
struct _list {
    int value;
    struct _list *next;
};

//新しい型STACKを定義
typedef struct _list LIST;

LIST *first = NULL;    //リストの最初

//リストにデータを挿入する
void push(int data)
{
    LIST *p;
    p = (LIST *)malloc(sizeof(LIST));    //メモリ領域を確保

    //エラー処理
    if (p == NULL) {
        puts("エラー  メモリがありません");
        return;
    }

    //初期化
    p->value = data;

    //リストの最初につなげる
    p->next = first;
    first = p;
}

//リストの先頭を削除
int pop(void)
{
```

```
LIST *p;
int value;

if (first == NULL) {
    puts("エラー データがありません");
    return 0;
}

p = first;
value = p->value;    //データを取りだし

first = first->next;//リストの先頭を削除
free((LIST *)p);//メモリ領域を解放

return value;
}

void main(void)
{
    push(10);
    push(20);
    push(30);
    printf("pop data = %d\n", pop());
    printf("pop data = %d\n", pop());
    printf("pop data = %d\n", pop());
    printf("pop data = %d\n", pop());
}
```

## ソート (整列)

```
#include <stdio.h>
int max(int *a, int n)
{
    int mid;
    int x, y;

    //要素が一つしかなければそれが最大値
    if (n == 1) {
        return a[0];
    }
    //要素が2つの場合、大きい方が最大値
    if (n == 2) {
        if (a[0] > a[1]) {
            return a[0];
        }
        else {
            return a[1];
        }
    }

    //配列の真ん中
    mid = n / 2;
    x = max(a, mid);
    y = max(&a[mid], n-mid);
    if (x > y) {
        return x;
    }
    else {
        return y;
    }
}

void main(void)
{
    int a[] = {7, 56, 12, 67, 78, 0, 11, 23, 43, 46, 76, 78, 79, 98, 54, 10, 78, 98, 5, 3, 11, 7, 0, 3, 5, 55};
    printf("最大値=%d\n", max(a, sizeof(a)/sizeof(int)));
}
```

## マクロ

```
#define wait(count)    { long l; for (l = 0; l < (count); l++) ; }
```

## 浮動小数点と数学関数

```
#include <stdio.h>
#include <math.h>

typedef struct _XY {
    double x;
    double y;
} XY;

double distance(XY p1, XY p2)
{
    return sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y)*(p1.y-p2.y));
}

void main(void)
{
    XY xy1 = {0, 0};
    XY xy2 = {10, 10};

    printf("距離=%f¥n", distance(xy1, xy2));
}
```

## 関数へのポインタ

```
#include <stdio.h>

typedef int (*FUNCP)(int, int);

int kekka(int *data, int n, FUNCP func)
{
    int i, k = data[0];
    for (i = 1; i < n; i++) {
        k = func(k, data[i]);
    }
    return k;
}

int sum(int m, int n)
{
    return m+n;
}

int mul(int m, int n)
{
    return m*n;
}

int max(int m, int n)
{
    return (m>n)?m:n;
}

int min(int m, int n)
{
    return (m>n)?n:m;
}

void main(void)
{
    int a[] = {1, 2, 3, 4, 5};
    printf("和=%d\n", kekka(a, sizeof(a)/sizeof(int), sum));
    printf("積=%d\n", kekka(a, sizeof(a)/sizeof(int), mul));
    printf("最大値=%d\n", kekka(a, sizeof(a)/sizeof(int), max));
    printf("最小値=%d\n", kekka(a, sizeof(a)/sizeof(int), min));
}
```

## 共用体

```
#include <stdio.h>

double mul2(double x)
{
    short exp;
    union {
        double d;
        unsigned long l[2];
    } dl;

    dl.d = x;
    exp = (dl.l[0] & 0x7ff00000) >> 20;
    exp++;
    dl.l[0] &= ~0x7ff00000;
    dl.l[0] |= (((unsigned long)exp & 0x7ff) << 20);

    return dl.d;
}

void main(void)
{
    printf("%fの2倍=%f\n", 3.1415, mul2(3.1415));
}
```

## I/Oレジスタのビットフィールドによる割り付け

```
union _adcsr {
    struct _adcsr_b {
        unsigned char CH:3;
        unsigned char CKS:1;
        unsigned char SCAN:1;
        unsigned char ADST:1;
        unsigned char ADIE:1;
        unsigned char ADF:1;
    } BIT;
    unsigned char BYTE;
};

#define ADCSR (*(volatile union _adcsr *)0xFFFFE8)
```

## 可變引數

```
#include <stdio.h>
#include <stdarg.h>

void put_int(int value)
{
    char buf[128];
    int i;
    char c;

    i = 0;
    while (1) {
        c = value % 10;
        buf[i++] = c + '0';
        value /= 10;
        if (value == 0) {
            break;
        }
    }

    while (1) {
        fputc(buf[--i], stdout);
        if (i == 0) {
            break;
        }
    }
}

void my_printf(char *str, ...)
{
    va_list ap;
    char c;
    int value;
    char *s;

    va_start(ap, str);
    while (c = *str++) {
        if (c == '%') {
```

```

        c = *str++;                                //次の文字読み込み
        switch (c) {
        case 'd':
            value = va_arg(ap, int);
            put_int(value);
            break;
        case 's':
            s = va_arg(ap, char *);
            while (c = *s++) {
                fputc(c, stdout);
            }
            break;
        case 'c':
            value = va_arg(ap, int);
            fputc(value, stdout);
            break;
        default:
            fputc(c, stdout);
        }
    }
    else {
        fputc(c, stdout);
    }
}

va_end(ap);
}

void main(void)
{
    my_printf("test string\n");
    my_printf("--- %s ---", "Hello, World");
    my_printf("value = %d\n", 23100);
    my_printf("value = %d\n", 1234);
    my_printf("char = %c\n", 'A');
}

```

このテキストでは説明していないがdo-while文を使うとput\_int関数をもっと簡素になる。

```
void put_int(int value)
{
    char buf[128];
    int i;
    char c;

    i = 0;
    do {
        c = value % 10;
        buf[i++] = c + '0';
    } while (value /= 10);

    do {
        fputc(buf[--i], stdout);
    } while (i == 0);
}
```

do-while文は、ループの最後に条件判定を行う。最低でも一回はループを実行する。