May 10, 04 10:40 **eg29−10scanf.c** Page 1/1

```c
/*
 * For each of the following scanf statements, note down
 * − whether compiler will give warning (when using −Wall)
 * − whether it will give a segmentation fault (and why)
 * − where the scanned value is stored and any bad side
 *   effects from it.
 */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *a;
    int  b;
    int  c[10];

    // scanf("%d", address)
    // scanf("%d", a);
    // scanf("%d", *a);
    // scanf("%d", &a);

    scanf("%d", b);
    scanf("%d", &b);

    scanf("%d", c);

    scanf("%d", *c);

    scanf("%d", &c);

    scanf("%d", c[0]);

    scanf("%d", &c[0]);

    return 0;
}
```

May 10, 04 11:18 **eg30−struct.c** Page

```c
/*
 * Introducing struct: its declaration, initialization, sizeo
 * and accessing members of a struct.
 *
 * Note that members of struct occupy consecutive location in
 * memory −− that's why the 2nd printf below works even thoug
 * it the number of arguments passed in is not correct.
 */
struct room {
    int number;
    int level;
    char *building;
};

int main() {
    struct room rooms[3] = {
        {4, 20, "SOC1"},
        {"S16", 5, 23},
        {"S14", 6, 02},
    };

    printf("Size of a room structure is %d\n", sizeof(struct room));
    // printf("My office is at %s %02d−%2d\n", rooms[0]);
    printf("My office is at %s %02d−%2d\n", rooms[0].building,
        rooms[0].level, rooms[0].number);
    return 0;
}
```

May 10, 04 11:22 **eg31−bytealign.c** Page 1/1

```c
/*
 * To access individual members in a struct, the members must
 * be aligned to word boundry.  (This is generally true for
 * variables −− in eg15, use gdb to check the address for x and
 * str and s.)
 *
 * Gaps between members are padded with unknown values.
 *
 * Segmentation fault and bus error −− what's the different?
 */
#include <stdio.h>

struct phone {
    char brand_id;
    float price;
    char x;
};

int main()
{
    printf("%d\n", sizeof(struct phone));
}
```

May 10, 04 11:27 **eg32−structptr.c** Page

```c
/*
 * Introduce the −> operator.
 *
 * Note that malloc returns void *, so we have to cast it to a
 * appropriate type.   We can change the content of a struct
 * if we pass it in as pointers.
 */
#include <stdlib.h>

struct room {
    char *building;
    int level;
    int number;
};

void move(struct room *r)
{
    r->building = "S16";
    r->level = 5;
    r->number = 10;
}

int main()
{
    struct room *r = (struct room*) malloc(sizeof(struct room
    r->building = "SOC1";
    r->level = 4;
    r->number = 20;

    move(r);

    return 0;
}
```

```c
/*
 * Introduces typedef.
 */

#include <stdlib.h>

struct room {
    char *building;
    int level;
    int number;
};

typedef struct room room;
typedef room *room_ptr;
typedef short integer;

void move(room_ptr r)
{
    r->building = "S16";
    r->level = 5;
    r->number = 10;
}

integer main()
{
    room_ptr r = (room_ptr) malloc(sizeof(room));
    r->building = "SOC1";
    r->level = 4;
    r->number = 20;

    move(r);

    return 0;
}
```



```c
/*
 * Using struct and pointers, we can implement higher
 * level data structure such as linked list, binary
 * trees, hash tables etc.
 */
#include <stdio.h>

typedef struct node {
    int data;
    struct node *next;
} node;

typedef struct list {
    struct node *head;
} list;

node *
node_new(int data)
{
    node *n = (node *)malloc(sizeof(node));
    n->data = data;
    n->next = NULL;

    return n;
}

int main()
{
    list l;
    node *curr;
    l.head = node_new(1);
    l.head->next = node_new(3);
    l.head->next->next = node_new(5);

    for (curr = l.head; curr != NULL; curr = curr->next)
    {
        printf("%d -> ", curr->data);
    }
    printf("NULL\n");
}
```

May 09, 04 14:05 **eg35−structarg.c** Page 1/1

```c
/*
 * You can pass a struct as argument to function, but not
 * recommended because of high overhead.
 */
struct room {
    char occupant[64];
    char building[12];
    int level;
    int number;
};

int is_my_office(room r)
{
    if (strcmp(r.occupant, "Ooi Wei Tsang") == 0)
        return 1;
    else
        return 0;
}

int main() {
    room r;
    strcpy(r.occupant, "Ooi Wei Tsang");
    strcpy(r.building, "SOC1");
    r.level = 4;
    r.number = 20;
    return is_my_room(r);
}
```

May 10, 04 11:48 **eg36−structcmp.c** Page

```c
/*
 * Introducing memcmp −− like strcmp but compares content of
 * memory.  Why doesn't this always work when we use memcmp
 * to compare two struct?
 */

#include <stdio.h>

struct phone {
    char brand_id;
    float price;
};

int main()
{
    struct phone p1, p2;

    p1.brand_id = 'N';
    p1.price = 3.0;

    p2.brand_id = 'N';
    p2.price = 3.0;

    if (memcmp(&p1,&p2,sizeof(struct phone)) == 0) {
        printf("p1 is the same as p2\n");
    } else {
        printf("p1 is not the same as p2\n");
    }
}
```

May 09, 04 15:31 **eg37−structcmp.c** Page 1/1

```c
/*
 * Comparing two struct by comparing member by member.
 * This will surely work.
 */

#include <stdio.h>

struct phone {
    char brand_id;
    float price;
};

int phone_equal(struct phone *p1, struct phone *p2)
{
    return (p1->brand_id == p2->brand_id &&
            p1->price == p2->price);
}

int main()
{
    struct phone p1, p2;
    p1.brand_id = 'N';
    p1.price = 3.0;
    p2.brand_id = 'N';
    p2.price = 3.0;

    if (phone_equal(&p1,&p2)) {
        printf("p1 is the same as p2\n");
    } else {
        printf("p1 is not the same as p2\n");
    }
}
```

May 10, 04 11:53 **eg38−structfile.c** Page

```c
#include <stdio.h>
/*
 * We can fread/fwrite struct from/to binary files.  But this
 * is not portable.
 */
struct room {
    char *building;
    int level;
    int number;
};

int main() {
    struct room r[3] = {{"SOC1",4,20},{"S16",5,10},{"S15",1,4}

    int i;
    FILE *f;

    f = fopen("room.db","wb");
    for (i = 0; i < 2; i++)
    {
        fwrite(&r[i],sizeof(struct room),1,f);
    }
    fclose(f);
}
```

May 09, 04 14:07 **eg39−main.c** Page 1/1

```c
/*
 * main function that calls a function foo in another file.
 */
#include <stdio.h>

int main()
{
    foo();
    return 0;
}
```

May 09, 04 14:08 **eg39−module.c** Page

```c
/*
 * Definition of a function.  Without main(), this
 * C file should be compiled to an object file, and
 * linked with the main() program.
 */
#include <stdio.h>

void foo(char *str)
{
    printf("%s foo\n", str);
}
```

```c
/*
 * static function cannot be access outside of
 * this C file.
 */
#include <stdio.h>

static void
foo(char *str)
{
    printf("%s foo\n", str);
}
```



```c
/*
 * There is only one copy of a static local variable
 * in the whole program.
 */
#include <stdio.h>

typedef struct node {
    int data;
    struct node *next;
} node;

typedef struct list {
    struct node *head;
} list;

node *
node_new()
{
    static int count = 0;

    node *n = (node *)malloc(sizeof(node));
    n->data = count++;
    n->next = NULL;
}

int main()
{
    list l;
    node *curr;
    l.head = node_new();
    l.head->next = node_new();
    l.head->next->next = node_new();

    for (curr = l.head; curr != NULL; curr = curr->next)
    {
        printf("%d -> ", curr->data);
    }
    printf("NULL\n");
}
```

```c
/*
 * Another feature that preprocessor provides
 * is macro -- function-like syntax that are
 * expanded by text-subsitutation.  Also called
 * call-by-name.
 *
 * Because macro is expanded with text-subsitution,
 * strange behaviour can occur if you are not careful.
 * See the examples below.  Use gcc -E to expand the
 * macros and see the expansion result.  How to
 * fix it?
 */
#define MAX(a,b) (a > b) ? a : b

int main()
{
    int x = 9;
    int y = 10;
    printf("%d\n", MAX(x,y));

    printf("%d\n", MAX(x,y++));
    printf("%d\n", y);

    x = 9;
    y = 10;
    printf("%d\n", MAX(y,x)+4);
}
```



```c
/*
 * Combination of #ifdef and #define macro can
 * be very useful.
 */
#ifdef DEBUG
#define LOG(str,arg) fprintf(stderr, str, arg)
#else
#define LOG(str,arg)
#endif

int fac(int n)
{
    LOG("n is %d\n", n);
    if (n == 0)
        return 1;
    else {
        int result = n*fac(n-1);
        LOG("returning %d\n", result);
        return result;
    }
}

int main()
{
    printf("10! is %d\n", fac(10));
    return 0;
}
```

```
/*
 * Macro can be compound statements, by new lines must be
 * escaped with "\".  This example introduces a _very_
 * _very_ useful macro for debugging called ASSERT.
 *
 * Standard C version is called assert (small letters) and
 * you can #include <assert.h> to use it.
 */
#include <stdio.h>

#define ASSERT(cond,msg) {\
    if (!(cond)) {\
        fprintf(stderr, "assertion failed in file %s, line %d\n",\
        __FILE__,__LINE__ );\
        fprintf(stderr, msg);\
    }\
}

int main()
{
    int i = -1;
    ASSERT(i > 0, "i is less than zero\n");
}
```