/* * 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;
}

/* * Introducing struct: its declaration, initialization, sizeof, * 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 though * it the number of arguments passed in is not correct. */

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

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].building,
    // rooms[0].level, rooms[0].number);
    printf("My office is at %s %02d-%2d\n", rooms[0].building,
            rooms[0].level, rooms[0].number);
    return 0;
}
/*
 * 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
", sizeof(struct phone));
}

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

#include <stdio.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()
{
    printf("%d
", sizeof(struct phone));
}

May 10, 04 11:27 eg32-structptr.c Page 1/1

/*
 * 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 = "SOC1";
    r->level = 4;
    r->number = 20;
    move(r);
    return 0;
}
#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;
}

#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");
}
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_office(r);
}
/* * 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");
  }
}

#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 < 3; i++)
  {
    fwrite(&r[i], sizeof(struct room), 1, f);
  }
  fclose(f);
}
/* main function that calls a function foo in another file. */

#include <stdio.h>

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

/* 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);
}
/* static function cannot be access outside of * this C file. */
#include <stdio.h>

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

#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");
}
Another feature that preprocessor provides is macro -- function-like syntax that are expanded by text-substitution. Also called call-by-name.

Because macro is expanded with text-substitution, 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?

```c
#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);
}
```

Combination of #ifdef and #define macro can be very useful.

```c
#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");
}