CSE 230
Intermediate Programming in C and C++

Structures

Fall 2017

Stony Brook University

Instructor: **Shebuti Rayana**

http://www3.cs.stonybrook.edu/~cse230/

Introduction

- In C, you can define data types that are constructed from the fundamental types.
  - For example, an array type is an example of this; it is a derived type that is used to represent homogeneous data.
- In contrast, the structure type is used to represent heterogeneous data.
  - A structure has components, called members, that are individually named. Because the members of a structure can be of various types, the programmer can create aggregates of data that are suitable for a particular application.

Structures

- Provides a means to aggregate variables of different types
- Example: A structure to define a playing card
  - The spots on a card that represent its numeric value are called "pips." A playing card such as the three of spades has a pip value, 3, and a suit value, spades.

```
struct card {
   int pips;
   char suit;
} ;
```

(i) `struct` is a keyword, (ii) `card` is the structure tag name, and (iii) the variables `pips` and `suit` are members of the structure.

- The variable `pips` will take values from 1 to 13, representing ace to king; the variable `suit` will take values from ‘c’, ‘d’, ‘h’, and ‘s’, representing the suits clubs, diamonds, hearts, and spades, respectively.

Structures (cont.)

- The declaration can be thought of as a template; it creates the `struct card`, but no storage is allocated.
- The tag name, along with the keyword `struct`, can now be used to declare variables of this type.

```
struct card c1, c2;
```

– This declaration allocates storage for the identifiers `c1` and `c2`, which are of `struct card`.

```
 struct card {
   int pips;
   char suit;
 } c1,c2;
```

To access the members of a structure, member access operator “.” is used.

```
c1.pips = 3;
c1.suit = ‘s’;
structure_ variable. member_name
```

Structure (cont.)

- If we want `c2` to represent the same playing card as `c1, c2 = c1;`
  - This causes each member of `c2` to be assigned the value of the corresponding member of `c1`.
- Programmers commonly use the `typedef` mechanism when using structure types.

```
typedef struct card card;
```

- Now, if we want more variables to represent playing cards,

```
card  c3, c4, c5;
```

Structure Member Naming

- Within a given structure, the member names must be unique.
  - However, members in different structures are allowed to have the same name. This does not create confusion because a member is always accessed through a structure identifier.

```
struct fruit {
   char *name;
   int calories;
};
```

```
struct vegetable {
   char *name;
   int calories;
};
```

```
       struct fruit a;
       struct vegetable b;
```

- You can access `a.calories` and `b.calories` without ambiguity

Structure Declaration

- Structure declaration ::= struct_specifier declarator_list;
- Struct_specifier ::= struct tag_name

  I struct tag_name$_{opt}$ { { member_declaration} $_{1+}$ }
- tag_name :: = identifier
- member_declaration :: = type_specifier declarator_list
- declarator_list :: = declarator { , declarator }$_{0+}$

Structures (cont.)

- Structures can be complicated.
  - They can contain members that are themselves arrays or structures
  - we can have arrays of structures

```
struct card {
        int pips;
        char suit;
   }deck[52];
```

- the identifier deck is declared to be an array of `struct card`

- If a tag name is not supplied, then the structure type cannot be used in later declarations.

- It is usually good programming practice to associate a tag name with a structure type.

Example

```
struct {
    int day, month, year;
    char day_name[4]; /* Mon, Tue, Wed, etc. */
    char month_name[4]; /* Jan, Feb, Mar, etc. */
} yesterday, today, tomorrow;
```

*more variables of this type cannot be declared later.

```
struct date{
    int day, month, year;
    char day_name[4]; /* Mon, Tue, Wed, etc. */
    char month_name[4]; /* Jan, Feb, Mar, etc. */
} yesterday, today, tomorrow;
```

```
struct date yesterday, today, tomorrow;
```

Structures (cont.)

- When using typedef to name a structure type, the tag name may be unimportant.

```
typedef struct{
    float re;
    float im;
} complex;
complex a,  b, c[100];
```

– The type complex now serves in place of the structure type. The programmer achieves a high degree of modularity and portability by using typedef to name such derived types and by storing them in header files.

Accessing Members of a Structure

- Member access operators: “.” and “->”

In file class_info.h

```
#define CLASS_SIZE 100
struct student {
   char *last_name;
   int student_id;
   char grade;
} ;
```

Suppose we are writing a program called `class_info`, which generates information about a class of 100 students.

```
#include "class_info.h"
int main(void)
{
struct student tmp, class[CLASS_SIZE];
… …
tmp.grade = ‘A’; tmp.lastname = “john”;
tmp.student_id = 910017;
```

Accessing Members of a Structure

- Now suppose we want to count the number of failing students in a given class.
  - To do this, we write a function named `fail()` that counts the number of F grades in the array `class[]`.
- The grade member of each element in the array of structures must be accessed.

```
/* Count the failing grades. */
#include ”class_info.h"
int fail(struct student class[])
{
      int i, cnt 0;
      for (i = 0; i < CLASS_SIZE; ++i)
            cnt += class[i].grade == ‘F’;
      return cnt;
}
```

Accessing Members of a Structure

- C provides the member access operator -> to access the members of a structure via a pointer.
  - This operator is typed on the keyboard as a minus sign followed by a greater than sign.
  - If a pointer variable is assigned the address of a structure, then a member of the structure can be accessed by a construct of the form `pointer_to_structure -> member_name`
- A construct that is equivalent to the above is `(*pointer_to_structure).member_name`
- The parentheses are necessary. Along with () and [], the operators "." and -> have the highest precedence and associate from left to right.
  - Thus, the preceding construct without parentheses would be equivalent to `*(pointer_to_structure. member_name)`
  - This is an error because only a structure can be used with the "." operator, not a pointer to a structure.

Example: add complex numbers

In file complex.h

```
struct complex{
   double re; /*real part*/
   double im; /*imag part*/
};
typedef struct complex complex;
```

In file 2_add.c

```
#include <complex.h>
/* a = b + c */
void add(complex *a, complex *b, complex *c){
    a->re = b->re + c->re;
    a->im = b->im + c->im;
}
```

Example: Member Access

Declaration and Assignment		

struct student tmp, *p = &tmp;
tmp.grade = 'A';
tmp.last_name = "Casanova";
tmp.student_id = 910017;		

Expression	Equivalent Expression	Conceptual Value
tmp.grade	p->grade	A
tmp.last_name	p->last_name	Casanova
(*p).student_id	p->student_id	910017
*p->last_name+1	(*(p->last_name))+1	D
*(p->last_name + 2)	(p->last_name)[2]	s

Using Structures with Functions

- Structures can be passed as arguments to a function and can be returned from them.
- When a structure is passed as an argument to a function, it is passed by value, meaning that a local copy is made for use in the body.
  - If a member of the structure is an array, then the array gets copied as well.
  - If the structure has many members, or members that are large arrays, then passing the structure as an argument can be relatively inefficient.
- An alternate scheme is to write functions that take an address of the structure as an argument instead.

Example: Business Application

```
struct dept {
      char dept_name[25];
      int dep_no;
} ;
```

the compiler has to know the size of each member

```
typedef struct {

    char name[25];

    int employee_id;

    struct dept department;

    struct home_address *a_ptr;

    double salary;

} employee_data;
```

Structure type member

Pointer to a Structure

the compiler already knows the size of a pointer, this structure need not be defined first.

Example: Business Application

- Function to update employee information

```
employee_data update(employee_data e)
{
      printf(“Input the department number: “);
      scanf(“%d”, &n);
      e.department.dept_no = n;
      return e;
}
```

– we are accessing a member of a structure within a structure

`e.department.dept_no` is equivalent to `(e.department).dept_no`

- To use the function `update()`, we could write in `main()` or in some other function

```
employee_data e;

e = update(e);
```

Copy Problem

```
employee_data update(employee_data e)
{
      printf(“Input the department number: “);
      scanf(“%d”, &n);
      e.department.dept_no = n;
      return e;
}
```

```
    employee_data e;

    e = update(e);
```

- `e` is being passed by value, causing a local copy of `e` to be used in the body of the function; when a structure is returned from `update()` , it is assigned to `e`, causing a member-by-member copy to be performed. Because the structure is large, the compiler must do a lot of copy work.

Alternate: Update Function

```
void update(employee_data *p)
{
      printf(“Input the department number: “);
      scanf(“%d”, &n);
      p->department.dept_no = n;
}
```

`p->department.dept_no` is equivalent to `(p->department).dept_no`

This version of update() can be used in main() as follows:

```
employee_data e;

update(&e);
```

- Here, the address of e is being passed, so no local copy of the structure is needed within the update() function. For most applications this is the more efficient of the two methods.

Initialization of Structures

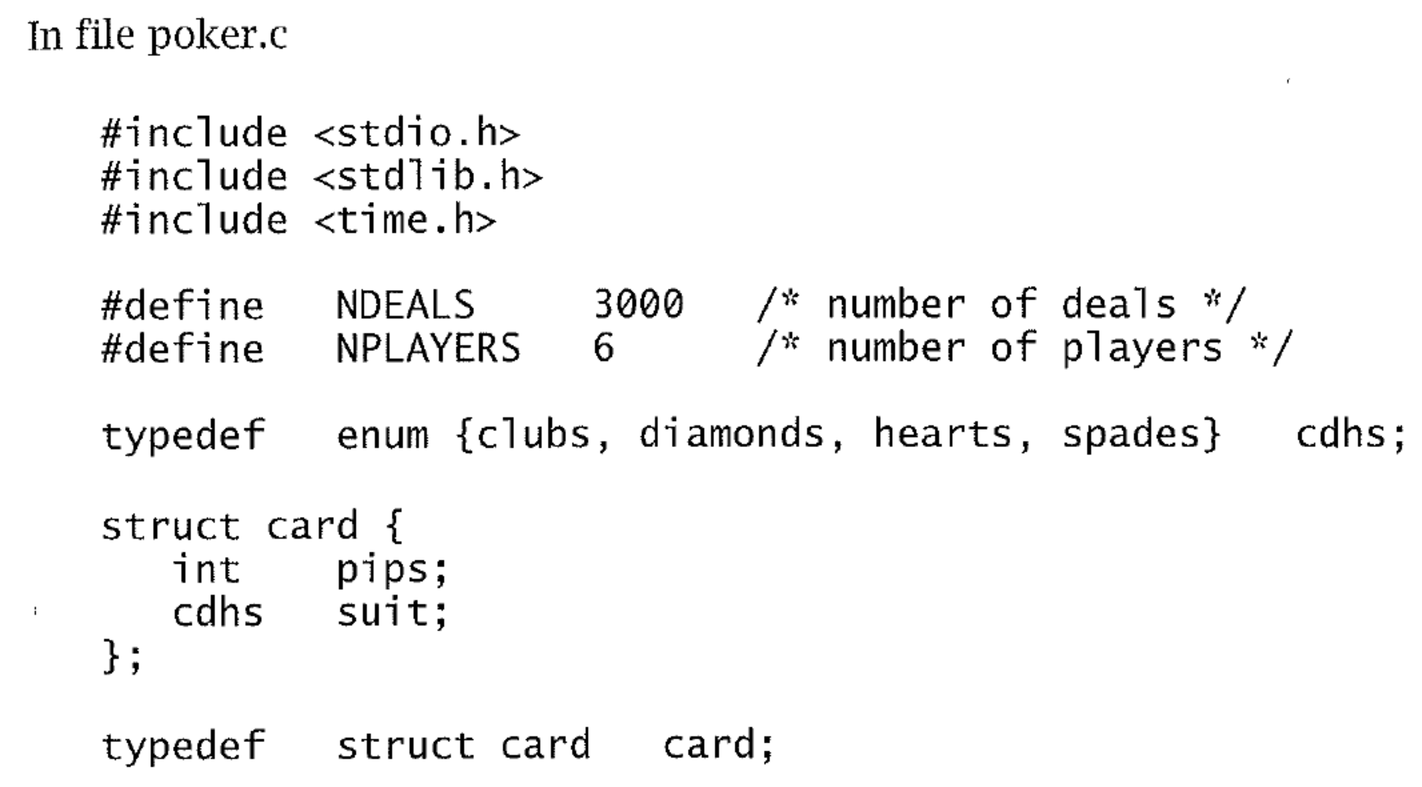
If not explicitly initialized by the programmer structures are automatically initialized by the system to zero. Structure initialization is similar to array.

```
card c = {l3, 'h'}; /* the king of hearts */
complex a[3][3] = {
{{1.0, -0.1}, {2.0, 0.2}, {3.0, 0.3}},
{{4.0, -0.4}, {5.0, 0.5}, {6.0, 0.6}},
}; /* a[2][] is assigned zeroes */
struct fruit frt = {"plum", 150};
struct home_address {
      char *street;
      char *city_and_state;
      long zip_code;
} address = {"87 West Street", "Aspen, Colorado", 80526};
struct home_address previous_address = {0};
```

The last example illustrates a convenient way to initialize all members of a structure to have value zero. It causes pointer members to be initialized with the pointer value NULL and array members to have their elements initialized to zero.

An Example: Playing Poker

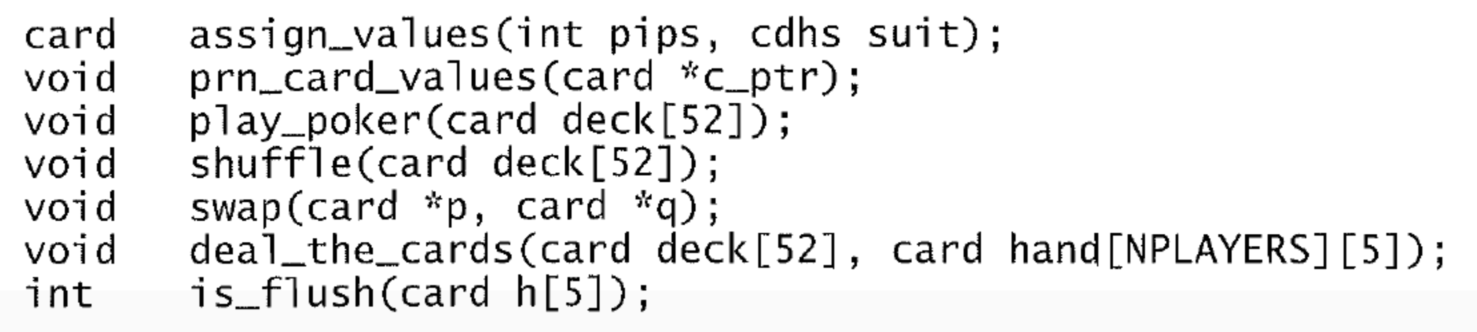
- The program will compute the probability that a flush is dealt, meaning that all five cards in a hand are of the same suit.



In file poker.c

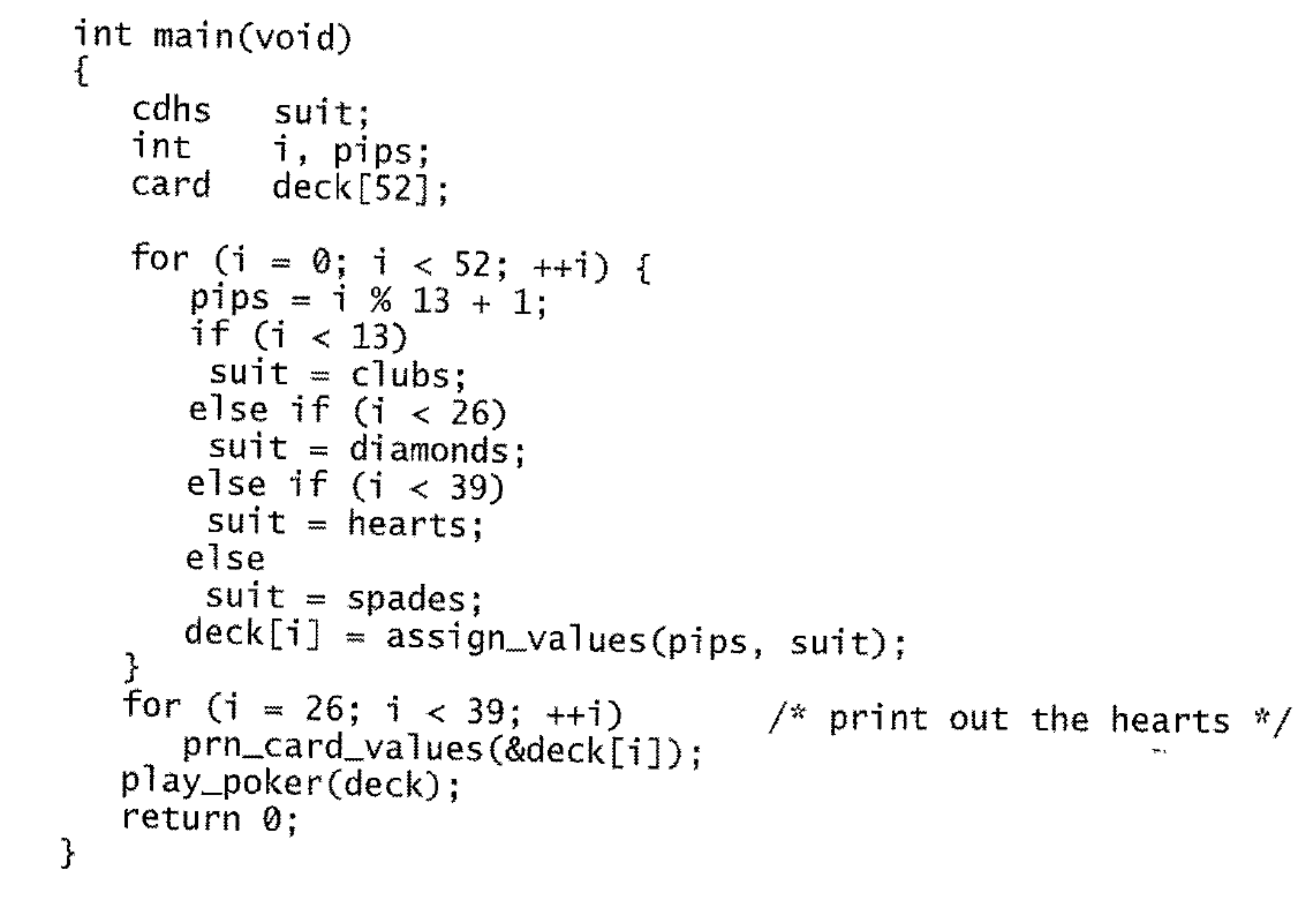
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define   NDEALS     3000   /* number of deals */
#define   NPLAYERS   6      /* number of players */

typedef   enum {clubs, diamonds, hearts, spades}   cdhs;

struct card {
   int    pips;
   cdhs   suit;
};

typedef   struct card   card;
```

An Example: Playing Poker



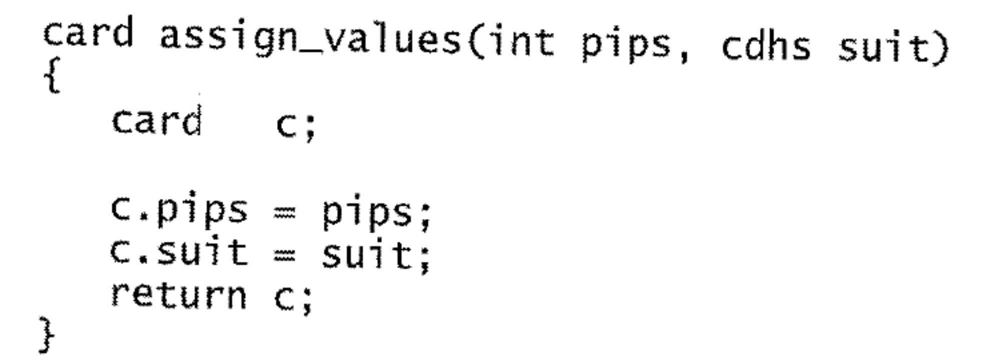
```
card   assign_values(int pips, cdhs suit);
void   prn_card_values(card *c_ptr);
void   play_poker(card deck[52]);
void   shuffle(card deck[52]);
void   swap(card *p, card *q);
void   deal_the_cards(card deck[52], card hand[NPLAYERS][5]);
int    is_flush(card h[5]);
```

An Example: Playing Poker

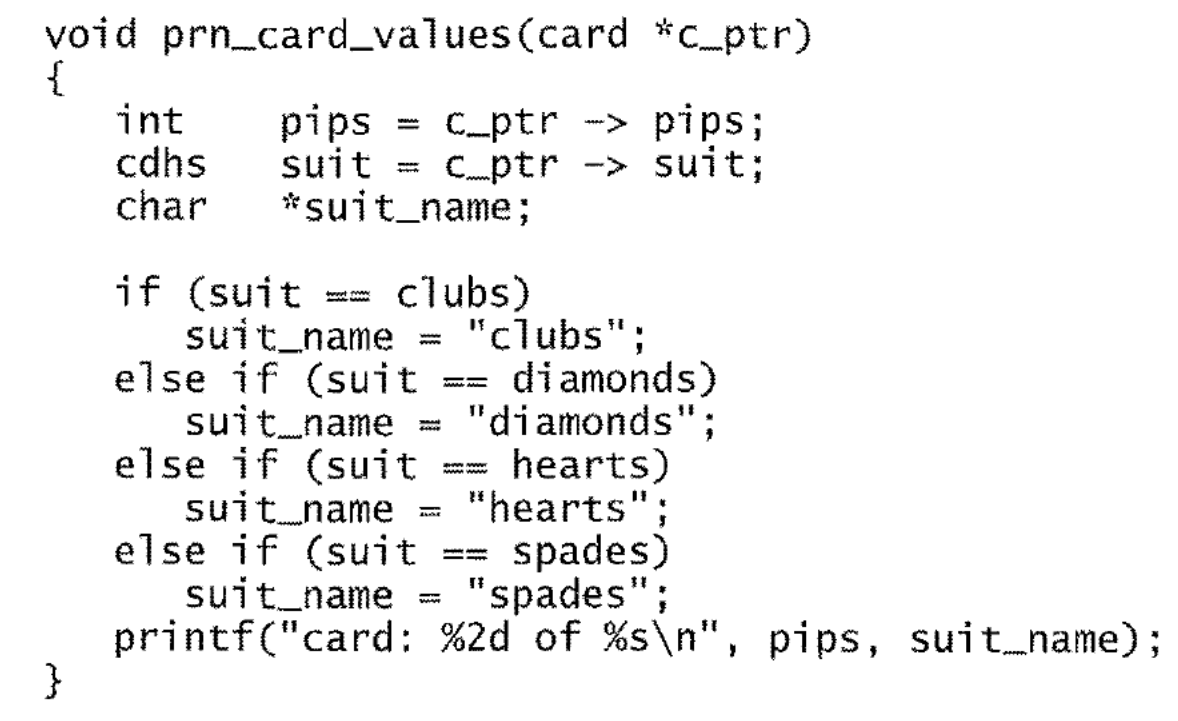


```
int main(void)
{
   cdhs   suit;
   int    i, pips;
   card   deck[52];

   for (i = 0; i < 52; ++i) {
      pips = i % 13 + 1;
      if (i < 13)
       suit = clubs;
      else if (i < 26)
       suit = diamonds;
      else if (i < 39)
       suit = hearts;
      else
       suit = spades;
      deck[i] = assign_values(pips, suit);
   }
   for (i = 26; i < 39; ++i)        /* print out the hearts */
      prn_card_values(&deck[i]);
   play_poker(deck);
   return 0;
}
```

An Example: Playing Poker

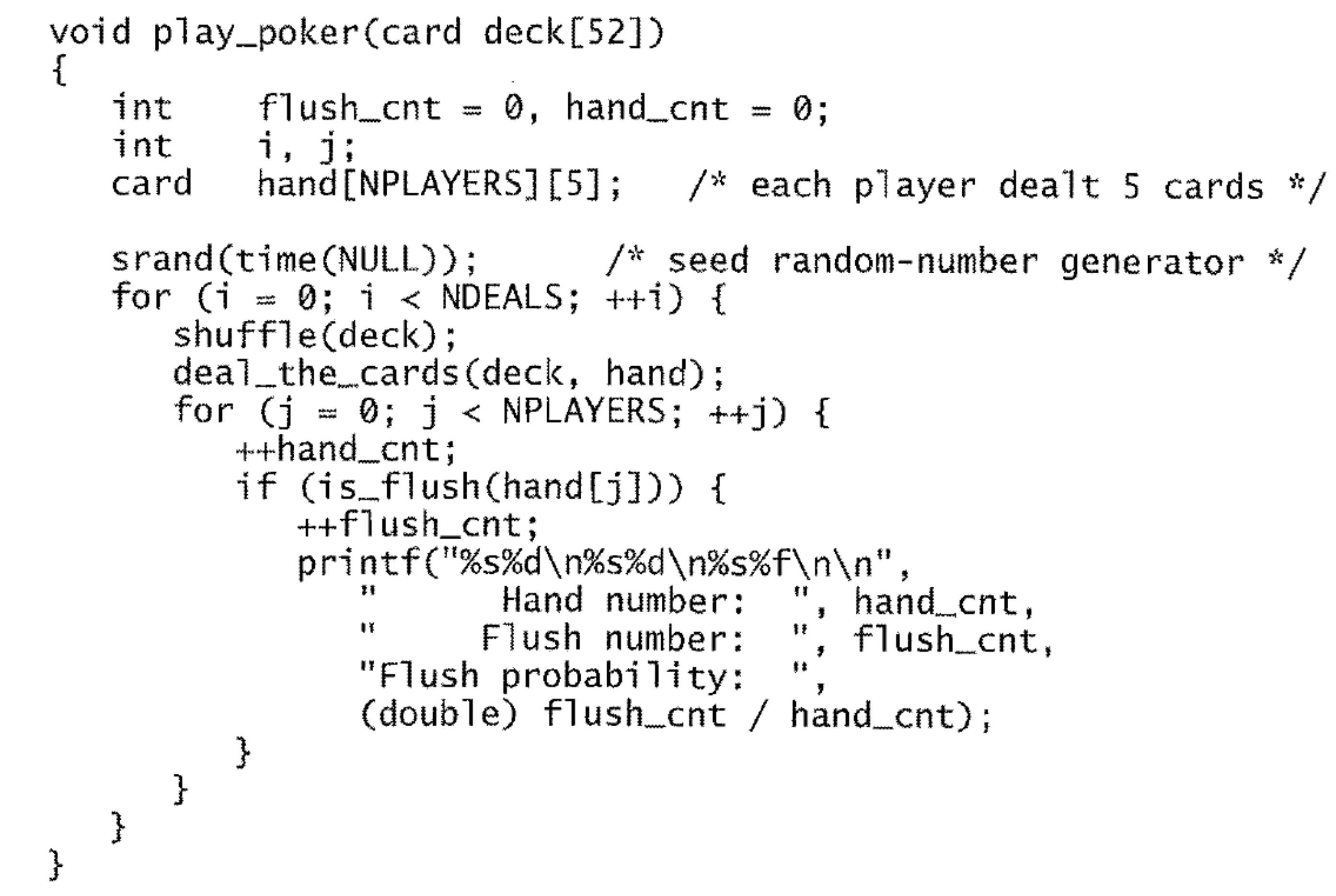


```
card assign_values(int pips, cdhs suit)
{
   card   c;

   c.pips = pips;
   c.suit = suit;
   return c;
}
```



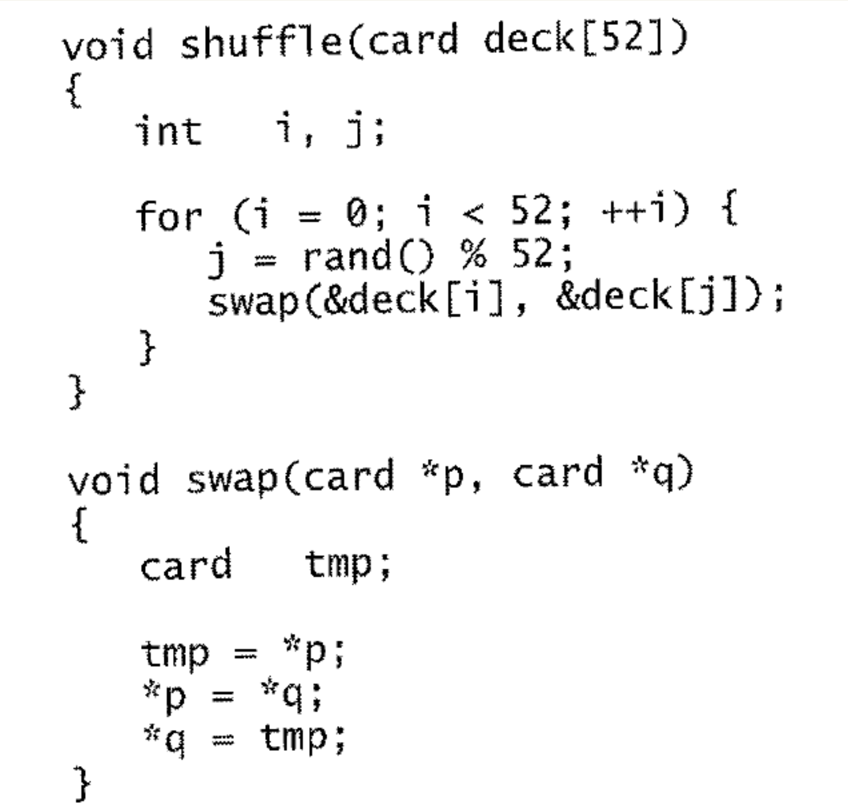
```
void prn_card_values(card *c_ptr)
{
   int    pips = c_ptr -> pips;
   cdhs   suit = c_ptr -> suit;
   char   *suit_name;

   if (suit == clubs)
      suit_name = "clubs";
   else if (suit == diamonds)
      suit_name = "diamonds";
   else if (suit == hearts)
      suit_name = "hearts";
   else if (suit == spades)
      suit_name = "spades";
   printf("card: %2d of %s\n", pips, suit_name);
}
```

An Example: Playing Poker

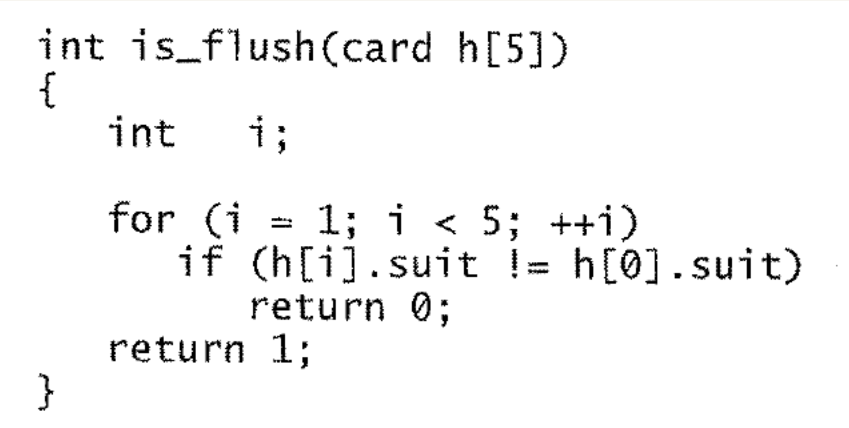


```
void play_poker(card deck[52])
{
   int    flush_cnt = 0, hand_cnt = 0;
   int    i, j;
   card   hand[NPLAYERS][5];   /* each player dealt 5 cards */

   srand(time(NULL));      /* seed random-number generator */
   for (i = 0; i < NDEALS; ++i) {
      shuffle(deck);
      deal_the_cards(deck, hand);
      for (j = 0; j < NPLAYERS; ++j) {
         ++hand_cnt;
         if (is_flush(hand[j])) {
            ++flush_cnt;
            printf("%s%d\n%s%d\n%s%f\n\n",
               "      Hand number:  ", hand_cnt,
               "     Flush number:  ", flush_cnt,
               "Flush probability:  ",
               (double) flush_cnt / hand_cnt);
         }
      }
   }
}
```

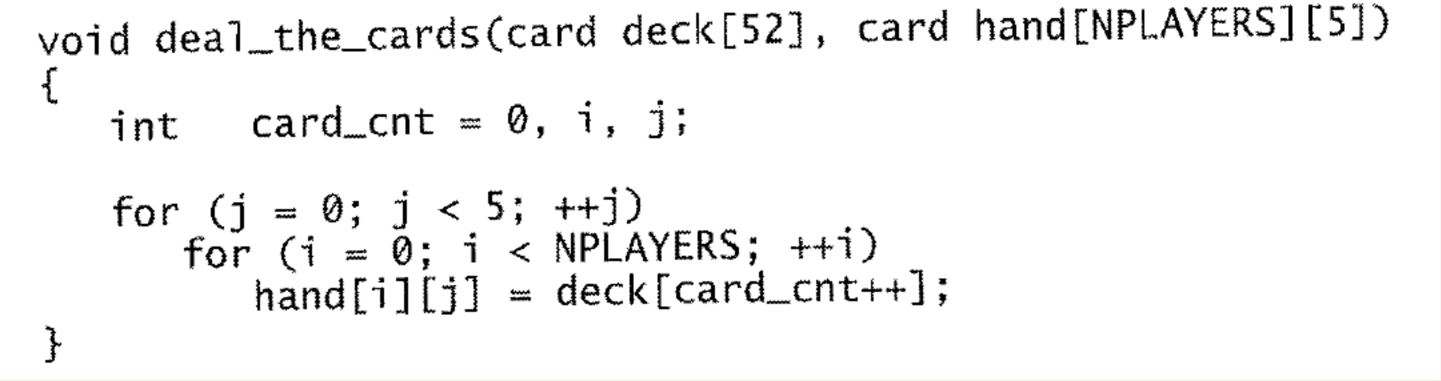
An Example: Playing Poker



```
void shuffle(card deck[52])
{
   int   i, j;

   for (i = 0; i < 52; ++i) {
      j = rand() % 52;
      swap(&deck[i], &deck[j]);
   }
}

void swap(card *p, card *q)
{
   card   tmp;

   tmp = *p;
   *p = *q;
   *q = tmp;
}
```



```
int is_flush(card h[5])
{
   int   i;

   for (i = 1; i < 5; ++i)
      if (h[i].suit != h[0].suit)
         return 0;
   return 1;
}
```



```
void deal_the_cards(card deck[52], card hand[NPLAYERS][5])
{
   int   card_cnt = 0, i, j;

   for (j = 0; j < 5; ++j)
      for (i = 0; i < NPLAYERS; ++i)
         hand[i][j] = deck[card_cnt++];
}
```