CS 24000 - Programming In C

Week Five: More expressions, statements, arrays of arrays, memory allocation,

Zhiyuan Li
Department of Computer Science
Purdue University, USA

Survey

- SIG OPS and SIGCSE are currently planning to have a C workshop sometime this semester
- They ask to conduct a survey at the end of the class
  - Would you be interested in attending a C workshop?

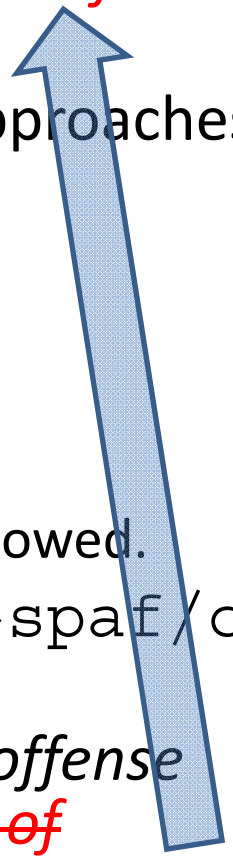
- Would you be interested in attending a C workshop?
- (a) Yes
- (b) No

Reminder on lab/project sharing policy

- Copying from other sources than one's own work is a case of cheating
- Letting others use one's code is also a case of cheating
- First offense gets a "-100" for the particular lab/project. Second offense gets an "F" for the course and a record in Dean of Students Office
- Dispute of penalty will be handled with the participation of Deans of Students Office

Reminder of the integrity policy

- *Any case of cheating will be handled by the Dean of students*
- You are encouraged to discuss problems and approaches but:
  - Sharing solution is not allowed.
  - Buying solutions is not allowed.
  - Copying code from the internet is not allowed.
  - Copying code from other students is not allowed.
  - Copying partial code from other students is not allowed.
- `http://homes.cerias.purdue.edu/~spaf/cpolicy.html`
- *First offense get a "-100" for the work. Second offense gets an "F" for the course ~~and a record in Dean of Students Office~~*



Revision of the integrity policy

- *Any case of cheating will be handled by the Dean of students*
- You are encouraged to discuss problems and approaches but:
  - Sharing solution is not allowed.
  - Buying solutions is not allowed.
  - Copying code from the internet is not allowed.
  - Copying code from other students is not allowed.
  - Copying partial code from other students is not allowed.
- `http://homes.cerias.purdue.edu/~spaf/cpolicy.html`
- *First offense get a "-100" for the work. Second offense gets an "F" for the course*

More suggestions on how to understand C better

- For questions “can I write this way?”, do programming experiments yourself
- To better understand compiler error message
  - Search on-line
- When not sure why the result is not accepted by autograder
  - Check your internal data and result
  - Examine (printout) and compare character by character if necessary

Postfix expressions: Structure References

- The "." operator E1.E2
  - E1 must be a reference to a structure or a union (which is a special case of structures)
  - E2 must be the name of a member of the structure/union
- The "->" operator E1->E2
  - E1 must be a pointer to a structure/union
  - E2 must be the name of a member of the structure/union
- Can be an lvalue unless the member is of an array type

Structures

- A C *struct* is a collection of one or more variables, possibly of different types

```
struct  {
    int x;                          /* a member of the
struct*/
    char c;                        /* another member of
the struct */
 }    y;
```

- A struct may have a name (*tag*), e.g. slot in the example below.

```
struct slot {
  int x;
  char c;
};                    /* then you do not need to declare y yet */
```

Comparison with Java

```
class Slot {
{
    int x;
    char c;
}
slot;
```

```
struct slot {
    int x;
  char c;
    }
```

```
typedef struct
    int x;
  char c;
    }
```

- Java C C

- Difference between the two:
  - No inheritance in C
  - No associated methods in C
  - Meanings of the declaration of a variable of the type are different

- **Slot y;**
  - In **Java**, declares y being a reference, i.e. a pointer, to an instance of Slot.
    - No memory is allocated until y = **new(Slot);**
- **Struct slot y; /* suppose slot is a tag */**
  - In **C**, memory is allocated to an instance of the struct slot referenced by **y**. (You can access a member of slot by writing y.c as in Java)
- **Struct slot *y; /* suppose slot is a tag */**
  - In C, declares x to be a pointer to an instance of struct Slot
    - No instance is created, no memory is allocated
    - If there is (eventually) an instance of Slot that y points to, you can access a member by writing y->c

- Type name can also be used after a struct has been declared in a typedef statement
- After **typedef struct {int x; char c;} slot;** /* slot is a type */
- **we can declare two instances of slot**

```
slot s1, s2;
```

- Pointers to structures can be defined

```
slot* p = &s1;
```

- Two equivalent syntactic ways to access members by reference

```
p->x
(*p).x
```

An Example: *struct.c*

```
#include <stdio.h>
main() {

        struct {int a; int b;} x, *p;
        p = &x;
        x.a=0;
        printf("x.a = \t %d\n", x.a);
        printf("p->a = \t %d\n", p->a);
        printf("(*p).a = \t %d\n", (*p).a);
}
```

Postfix increment/decrement

- `E++` or `E—`
  - E must be a postfix expression that has an lvalue
  - Both E++ and E-- have the value of E at the time of the evaluation
  - After the evaluation point, E gets incremented/decremented by 1
  - The result is not an lvalue.

Combinations of postfix expressions

- Evaluated from left to right
- f(arg1, arg2)->a[i]
  - Calls f(), which returns a pointer to a structure that has a member that is an array a[]
  - Use integer i to address the ith element of a[]

In what follows, we shall go quickly through a long list of different operators

Unary Operators

- One lower level of precedence than postfix expressions
- First, we have prefix increment/decrement

  `++E` or `--E`

- E must have an lvalue
  - i++ has a higher precedence and its result is no longer an lvalue
  - `--i++` is equivalent to –(i++) and will get a compiler error message
- The value of E is the value *after* the pre-increment/pre-decrement

Address operator

- Next, we have Address Operators
  - &E
- E must be
  - an lvalue referring neither to a *bit-field* nor to an object declared as *register*,
    - Cannot write &(p++) or &arr for array arr[]
    - But can write &arr[3]
    - If p is a pointer, then &p[3] is the address of p[3]
      - Because the postfix operator [ ] has a higher precedence than &
  - or must be of *function* type.
- The result is a *pointer* to the object or function referred to by the lvalue.
- If the type of the operand is T, the type of the result is ``*pointer to T*.''

It is easy to use a pointer to overwrite a large area of memory -- hence the potential hazard

```
#include <stdio.h>    /* badsweep.c */

static int sx;
static int sa[100];
static int sy;

int main() {
int *p;
  for(p= &sx;
      p <= &sx+200;
      p++)     *p = 42;

  printf("sx = \t%i\n",sx);
  printf("sa[0] = \t%i\n",sa[0]);
  printf("sa[109] = \t%i\n",sa[109]);
  printf("sy = \t%i\n",sy);
}
```

A New Example to see effect of E++ (better than the one discussed in class)

- Purpose of showing *increment.c*
  - *Compiler error if misuse non-lvalue*
  - *Difference between *p++ and (*p)++*

```
#include <stdio.h>
main() {
        int i=0, j[2], *p, *q;
        p = &i;
        q = j;
        printf("i++ = \t %d\n", i++);
        printf("i is now  \t %d\n", i);
        printf("*p is now  \t %d\n", *p);

/*      (i++)++;  */
        j[0]=2;
        j[1]=3;
        *q++ = 0;
        printf("j is now  \t %p\n", j);
        printf("j[0] is now  \t %d\n", j[0]);
        printf("j[1] is now  \t %d\n", j[1]);
        printf("q is now  \t %p\n", q);

        q = j;
        printf("q is now  \t %p\n", q);
        printf("(*q)++ is now  \t %d\n", (*q)++);
        printf("q is now  \t %p\n", q);
        printf("(*q)++ is now  \t %d\n", (*q)++);
        printf("q is now  \t %p\n", q);
        printf("j[0] is now  \t %d\n", j[0]);
        printf("j[1] is now  \t %d\n", j[1]);
}
```

Reminder of midterm 1

- Next Thursday, Feb. 14, in class
- Look at Piazza announcement for rules and preparations