Lecture 18

CIS 208

Wednesday, March 30, 2005
April 15th: Happy Tax day.

Review: Wednesday, the 13th
bit fields

- structs that hold only bit indicators
- like an array of booleans
bit field

struct tag {
    type name1: length;
    type name2: length;
    
    type nameN: length;
} variable_list;
bit field types

- must be either int, unsigned, or signed.

- length is the number of bits assigned to that label

- total size can’t be larger that 1 word
example

struct status_type{
    unsigned send : 2;  // 2 bits wide
    unsigned rec : 3;   // 3 bits wide
    unsigned p1 : 1;
    unsigned p2 : 1;
} status ;
accessing

once again, . or ->

same rules, blah blah blah.

status.rec = 4; status.p2 = 1;
if (status.p1) {...}
run time errors

- attempting to give a value outside of range.

```python
status.send = 5;
status.rec = -1;
```
can mix and match

combine bit fields with other structs

```c
struct emp {
    struct adr address;
    float pay;
    unsigned lay_off : 1;
    unsigned hourly: 1;
    unsigned deductions: 3;
};
```
restrictions

- Single bits variables must be unsigned.
- Can’t find address of bit field variable
- Bit field variables can’t be arrayed.
Uses:

- Limited storage (can store several Booleans in just one byte)
- Some devices only transmit info encoded into bits.
int ungetc(char ch, FILE *stream)

- returns a character to an input stream.
- Can’t unget EOF

- Will be the next character read from stream.

- returns value equal to ch if successful, else EOF

- one character pushback is guaranteed. Some compilers will accept more.
example

#include <stdio.h>
void read_word(FILE *fp, char *token) {
    while (isalpha(*token = getc(fp))) token++;
    ungetc(fp, *token);
}

read_word("count/10");
Conditional Compilation Directives

- Might not want some code compiled.
- These preprocessor commands tell compiler to skip over section of code.
- Allow for easy porting.
```c
#if constant_expression
statement sequence
#endif

- if the expression, then the statement sequence is compiled.
- expression must be constant, no variables.
- use Macros.
- Can be nested
```
#if #else #elif #endif

#include <stdio.h>
#define MAX 10;

void main(void) {
    #if MAX > 99
        printf("compiled for array greater than 99\n");
    #else
        printf("compiled for small array\n");
    #endif
}
#elif == else if

#define US 0
#define ENGLAND 1
#define FRANCE 2
#define ACTIVE_COUNTRY US

#if ACTIVE_COUNTRY == US
    char currency[] = "dollar";
#elif ACTIVE_COUNTRY == ENGLAND
    char currency[] = "pound";
#else
    char currency[] = "franc";
#endif
```
#ifdef #ifndef
    ifndef macro_name
        statement sequence
    endif
    True if macro_name is defined.
#endif
    ifndef macro_name
        statement sequence
    endif
    True if macro_name is undefined.
```
```c
#include

int main(int argc, char *argv[])
{

    if (argc > 1)
    {
        printf("Hello, %s!
", argv[1]);
    }
    else
    {
        printf("Hello, World!
");
    }

    return 0;
}
```
#undef

- undefined a macro.
- after command, macro name is useless.

- can be re-defined anytime afterwards.
sets the current line number

#include <stdio.h>
#line 100 //reset the line counter
void main(void) {
    printf("%d", __LINE__); //line 101
}

prints out 101.
pre-defined macros

_LINE__   :: current line number (int)
__FILE__   :: current file name (string)
__DATE__   :: date month/day/year (string)
__TIME__   :: Time, hour:minute:second (string)
__STDC__   :: constant decimal 1
/*This is C-Style
*/

//This is C++ style.
//Might not work with all compilers.
C++

 Starts on Friday

 Won’t be on test 2