About BPL
Here is a typical BPL program:

```c
int fact(int n) {
    if (n == 0)
        return 1;
    else
        return n*fact(n-1);
}

void main( void ) {
    int x;
    x = 1;
    while (x < 10) {
        write(x);
        write( fact(x) );
        writeln();
        x = x + 1;
    }
}
```
• A program is a sequence of declarations, one of which should be of a function main().

• White space is ignored in BPL.

• The semicolon acts as a statement terminator.

• The comment delimiters in BPL are /* and */

• The grammar for BPL in the reference manual defines exactly how the syntax works. If you aren't familiar with grammars, we'll start working with them by the end of this week.
The datatypes of BPL are

- integer as in `int x;`
- string as in `string s;`
- pointer to integer as in `int *p;`
- pointer to string as in `string *q;`
- integer array, as in `int A[10];`
- string array, as in `string B[5];`
BPL has no string library; strings are only constant.

There is no dynamic memory allocation, though functions can have local data, including local arrays. Pointers exist in BPL to allow for C-style call-by-reference functions.

Note that, unlike C, BPL does runtime bounds checking on arrays. If you declare array A to have length 5 and pass it to a function that tries to access entry [10], an error occurs. The compiler doesn't catch this; the assembly language code your compiler generates catches it.