Coding a 0 & 1s Parser

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

#define MAXNUMTOKENS 100
#define MAXTOKENSIZE 20
#define strsame(A,B) (strcmp(A, B)==0)
#define ERROR(PHRASE) { fprintf(stderr, \
           "Fatal Error %s occurred in %s, line %d\n", PHRASE, \
           __FILE__, __LINE__); \
           exit(EXIT_FAILURE); }

struct prog{
   char wds[MAXNUMTOKENS][MAXTOKENSIZE];
   int cw; // Current Word
};
typedef struct prog Program;

void Prog(Program *p);
void Code(Program *p);
void Statement(Program *p);

int main(void)
{
   Program* prog = calloc(1, sizeof(Program));
   int i=0;
   while(scanf("%s", prog->wds[i++])==1 && i<MAXNUMTOKENS);
   assert(i<MAXNUMTOKENS);
   Prog(prog);
   printf("Parsed OK\n");
   return 0;
}
```

Coding a 0 & 1s Parser

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

#define MAXNUMTOKENS 100
#define MAXTOKENSIZE 20
#define strsame(A,B) (strcmp(A, B)==0)
#define ERROR(PHRASE) { fprintf(stderr, \
           "Fatal Error %s occurred in %s, line %d\n", PHRASE, \
           __FILE__, __LINE__); \
           exit(EXIT_FAILURE); }

struct prog{
   char wds[MAXNUMTOKENS][MAXTOKENSIZE];
   int cw; // Current Word
};
typedef struct prog Program;

void Prog(Program *p);
void Code(Program *p);
void Statement(Program *p);

int main(void)
{
   Program* prog = calloc(1, sizeof(Program));
   int i=0;
   while(scanf("%s", prog->wds[i++])==1 && i<MAXNUMTOKENS);
   assert(i<MAXNUMTOKENS);
   Prog(prog);
   printf("Parsed OK\n");
   return 0;
}
```

```c
void Prog(Program *p)
{
   if(!strsame(p->wds[p->cw], "BEGIN")){
      ERROR("No BEGIN statement ?");
   }
   p->cw = p->cw + 1;
   Code(p);
}

void Code(Program *p)
{
   if(strsame(p->wds[p->cw], "END")){
      return;
   }
   Statement(p);
   p->cw = p->cw + 1;
   Code(p);
}

void Statement(Program *p)
{
   if(strsame(p->wds[p->cw], "ONE")){
      return;
   }
   if(strsame(p->wds[p->cw], "NOUGHT")){
      return;
   }
   ERROR("Expecting a ONE or NOUGHT ?");
}
```

Running the Parser

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

```
BEGIN ONE NOUGHT NOUGHT END
```

Parsed OK

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

```
BEGIN ONE NOUGHT NOUGHT END
```

Parsed OK

```
BEGIN END
```

Parsed OK

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

```
BEGIN ONE NOUGHT NOUGHT END
```

Parsed OK

```
BEGIN END
```

Parsed OK

```
BEGIN
 ONE
 TWO
END
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

```
BEGIN ONE NOUGHT NOUGHT END
```

Parsed OK

```
BEGIN END
```

Parsed OK

```
BEGIN
 ONE
 TWO
END
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
BEGIN
 ONE
 NOUGHT
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

```
BEGIN ONE NOUGHT NOUGHT END
```

Parsed OK

```
BEGIN END
```

Parsed OK

```
BEGIN
 ONE
 TWO
END
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
BEGIN
 ONE
 NOUGHT
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
 ONE
 NOUGHT
END
```

Fatal Error No BEGIN statement ? occurred in p01a.c, line 55

- Notice that the END statement is actually used as the recursive base-case in the formal grammar in the function Code().

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

```
BEGIN ONE NOUGHT NOUGHT END
```

Parsed OK

```
BEGIN END
```

Parsed OK

```
BEGIN
 ONE
 TWO
END
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
BEGIN
 ONE
 NOUGHT
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
 ONE
 NOUGHT
END
```

Fatal Error No BEGIN statement ? occurred in p01a.c, line 55

- Notice that the END statement is actually used as the recursive base-case in the formal grammar in the function Code().
- The parser doesn't actually **do** anything other than check that the input is **valid** or not.

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

```
BEGIN ONE NOUGHT NOUGHT END
```

Parsed OK

```
BEGIN END
```

Parsed OK

```
BEGIN
 ONE
 TWO
END
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
BEGIN
 ONE
 NOUGHT
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
 ONE
 NOUGHT
END
```

Fatal Error No BEGIN statement ? occurred in p01a.c, line 55

- Notice that the END statement is actually used as the recursive base-case in the formal grammar in the function Code().
- The parser doesn't actually **do** anything other than check that the input is **valid** or not.
- An interpreter performs the required operations (e.g. printing to the screen in this case) alongside the parser checking the syntax.

Running the Parser

```
BEGIN
  ONE
  NOUGHT
  ONE
END
```

Parsed OK

```
BEGIN ONE NOUGHT NOUGHT END
```

Parsed OK

```
BEGIN END
```

Parsed OK

```
BEGIN
 ONE
 TWO
END
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
BEGIN
 ONE
 NOUGHT
```

Fatal Error Expecting a ONE or NOUGHT ? occurred in p01a.c, line 79

```
 ONE
 NOUGHT
END
```

Fatal Error No BEGIN statement ? occurred in p01a.c, line 55

- Notice that the END statement is actually used as the recursive base-case in the formal grammar in the function Code().
- The parser doesn't actually **do** anything other than check that the input is **valid** or not.
- An interpreter performs the required operations (e.g. printing to the screen in this case) alongside the parser checking the syntax.
- A slight modification to the code is required to produce an interpreter.

Interpreters are Modified Parsers

```
void Statement(Program *p)
{
   if(strsame(p->wds[p->cw], "ONE")){
      printf("1\n");
      return;
   }
   if(strsame(p->wds[p->cw], "NOUGHT")){
      printf("0\n");
      return;
   }
   ERROR("Expecting a ONE or NOUGHT ?");
}
```

Interpreters are Modified Parsers

```
void Statement(Program *p)
{
   if(strsame(p->wds[p->cw], "ONE")){
      printf("1\n");
      return;
   }
   if(strsame(p->wds[p->cw], "NOUGHT")){
      printf("0\n");
      return;
   }
   ERROR("Expecting a ONE or NOUGHT ?");
}
```

Execution :

```
BEGIN
ONE NOUGHT ONE NOUGHT
END
1
0
1
0
```

Interpreters are Modified Parsers

```
void Statement(Program *p)
{
   if(strsame(p->wds[p->cw], "ONE")){
      printf("1\n");
      return;
   }
   if(strsame(p->wds[p->cw], "NOUGHT")){
      printf("0\n");
      return;
   }
   ERROR("Expecting a ONE or NOUGHT ?");
}
```

Execution :

```
BEGIN
ONE NOUGHT ONE NOUGHT
END
1
0
1
0
```

- I've also taken out the "Parsed OK" message.

Interpreters are Modified Parsers

```
void Statement(Program *p)
{
   if(strsame(p->wds[p->cw], "ONE")){
      printf("1\n");
      return;
   }
   if(strsame(p->wds[p->cw], "NOUGHT")){
      printf("0\n");
      return;
   }
   ERROR("Expecting a ONE or NOUGHT ?");
}
```

Execution :

```
BEGIN
ONE NOUGHT ONE NOUGHT
END
1
0
1
0
```

- I've also taken out the "Parsed OK" message.
- To extend the parser to be an interpreter you might now need to 'understand' what the input means - the context-free requirement is removed somewhat.