print decoded letter
reset the tree

print letter
print space
reset
```c
#include "buttons.h"

morse_init();
while(1)
{
    switch(ME):
    {
        case dot:

            case interleter

                3
            in + 100 Hz
            ME = MCE();
        
```
Case dot:
  \texttt{Update topline('.');}
  \texttt{morsedecode(Morse\_DOT);} \Rightarrow \texttt{ME = NONE;}

Case interletter:
  \texttt{C = Morsedecode(EN1\_OF\_Char)}
  \texttt{Update middleline('c')}
  \texttt{ClearTopline()}
  \texttt{Morsedecode(Reset)}
LAT 0

E

in your code
\text{Chor Line[width]} = a \circ b \circ c
Advanced Language Concepts

- Unions
- Function pointers
- Void pointers
- Variable-length arguments
- Program arguments
Unions
Unions allow the same piece of memory to be used as different datatypes in different contexts. A single union can hold any datatype that is in its declaration.

- Unions:
  - May contain any number of members of any type
  - Are as large as their largest member
  - Initializing uses the datatype of its first member
  - Use exactly the same syntax as structures except `struct` is replaced with `union`
Unions
Creating unions

**Syntax**

```c
union UnionName {
    type1 memberName1;
    ...
    typen memberNamenn;
};
```

**Example**

```c
union MixedBag {
    char a; 8
    int b; 32
    float c; 32
};
```
Unions

Unions and \texttt{typedef}

\textbf{Syntax}

\begin{verbatim}
typedef union \textit{UnionTag}_{\textit{optional}} 
  \{ 
  \textit{type}_1 \textit{memberName}_1; 
  \ldots 
  \textit{type}_n \textit{memberName}_n; 
\} \textit{typeName}; 
\end{verbatim}

\textbf{Example}

\begin{verbatim}
typedef union 
  \{ 
  char a; 
  int b; 
  float c; 
\} MixedBag; 
\end{verbatim}
Unions
Initializing unions

Syntax

```c
union UnionName {
    type1 memberName1;
    ...
    typen memberNameN;
} variableName = {VALUE};
```

Example

```c
union MixedBag {
    char a;
    int b;
    float c;
} myBag = {'a'};
```
Unions

In memory

- Memory is only allocated to accommodate the union’s largest member

Example

```c
typedef union {
    char a;
    short b;
    float c;
} MixedBag;

MixedBag x;
```

Space allocated for `x` is `sizeof(float)`
Unions

In memory

- Memory is only allocated to accommodate the union’s largest member

Example

```c
typedef union {
    char a;
    short b;
    float c;
} MixedBag;

MixedBag x;
```
Unions
In memory

• Memory is only allocated to accommodate the union’s largest member

Example

typedef union {
    char a;
    short b;
    float c;
} MixedBag;

MixedBag x;

Data Memory (RAM)

0x800 0x804 0x808 0x80C

x.b only occupies the lowest two bytes of the union
Unions

In memory

- Memory is only allocated to accommodate the union’s largest member

```c
typedef union {
    char a;
    short b;
    float c;
} MixedBag;

MixedBag x;
```

X. C occupies all four bytes of the union

Data Memory (RAM)

0x800
0x804
0x808
0x80C
Unions
Accessing members

Example

typedef union {
  char a;
  int b;
  float c;
} MixedBag;

MixedBag myBag = {'a'};
printf("myBag: char=%c, int=%d, float=%f",
    myBag.a, myBag.b, myBag.c);
Unions
Real-world example

Example: Binary tree for storing chars, ints, or floats

```c
typedef union {
    char asChar;
    int asInt;
    float asFloat;
} AnyData;

typedef enum {
    CHAR,
    INT,
    FLOAT,
} DataType;

typedef struct Node {
    struct Node *leftChild;
    struct Node *rightChild;
    DataType type;
    AnyData data;
} Node;
```
Function pointers
Function Pointers

- Pointers may also be used to point to functions
  - Because it's just a memory address
- Provides a more flexible way to call a function, by providing a choice of which function to call
- Makes it possible to pass functions to other functions
- Not extremely common, but very useful in the right situations
Function Pointers

Declaration

- A function pointer is declared much like a function prototype:

  ```c
  int (*fp)(int x);
  ```

- Here, we have declared a function pointer with the name `fp`:
  - The function it points to takes one int parameter
  - The function it points to returns an int
Function Pointers

Initialization

- A function pointer is initialized by setting the pointer name equal to the function name

If we declare the following:

```c
int (*fp)(int x);  // Function pointer
int Foo(int x);    // Function prototype
```

We can initialize the function pointer like this:

```c
fp = Foo;           // fp now points to Foo
```
Function Pointers
Calling a Function via a Function Pointer

- The function pointed to by fp from the previous slide may be called like this:

\[
y = fp(x) ;
\]

- This is the same as calling the function directly:

\[
y = Foo(x) ;
\]
Function Pointers

Passing a Function to a Function

Example: Understanding the Mechanism

```c
int x;
int Foo(int a, int b); // Function prototype

// Function definition with function pointer parameter
int Foobar(int a, int b, int (*fp)(int, int))
{
    return fp(a, b); // Call function passed by pointer
}

void main(void)
{
    x = Foobar(5, 12, Foo); // Pass address of foo
}
```
Function Pointers
Passing a Function to a Function

Example: Evaluate a definite integral (approximation)

```c
float Integrate(float from, float to, float (*f)(float))
{
    float sum = 0.0;
    float x;
    int n;

    // Evaluate integral\{a,b\} f(x) dx
    const float span = to - from;
    for (n = 0; n <= 100; n++) {
        x = ((n / 100.0) * span) + from;
        sum += (f(x) * span) / 101.0;
    }
    return sum;
}
```

Adapted from example at: http://en.wikipedia.org/wiki/Function_pointer
Function Pointers
Passing a Function to a Function

Example: Generic LinkedList

```c
typedef struct ListItem {
    struct ListItem *previousItem;
    struct ListItem *nextItem;
    void *data;
} ListItem;

int LinkedListPrint(const ListItem *list,
    void (*Print)(const ListItem *));

int LinkedListSort(ListItem *list,
    const ListItem **(Compare)(const ListItem *));
```
configure.h

user runfoo

user runbar

C \[ \gamma \] = runfoo, runbar
Void pointers
Void pointers

**Definition**

Void pointers are pointers that can hold a pointer to any type of data.

- **Cannot be dereferenced**
  - The size of the data cannot be inferred
  - Needs to be cast first
- **Cannot point to functions**
- **Are big enough to store any pointer**
Void pointers
Implicit casting

- Implicitly cast to other pointer types

Example

```c
Node *node = malloc(sizeof(Node));

int *node = malloc(sizeof(Node));

void *node = malloc(sizeof(Node));
```
Void pointers
Dereferencing

• Void pointers cannot be dereferenced

Example

```c
void *node = malloc(sizeof(Node));

node->data = 'a';
```
Void pointers
Dereferencing

- Void pointers cannot support pointer math
  - No associated size

Example

```c
void *node = malloc(2 * sizeof(Node));
(node + 1)->data = 'b';
```
Variable-length arguments

Variadic
Variable-length arguments

Syntax

\texttt{type \textit{Name}(type_1 \ arg_1, \ldots, type_n \ arg_n, \ldots)}.

- Requires at least one named argument
- ... states that the number and types the arguments may vary
  - It must be the last argument
- \texttt{<stdarg.h>} defines macros for iterating through all arguments
Variable-length arguments

Argument count

• No way to know how many arguments
• Solutions:
  – A count argument
  – A sentinel value
  – Use a formatting string like printf/scanf
Variable-length arguments

Iteration: Count argument

Example

```c
#include <stdarg.h>
int AllSum(int count, ...)
{
    // Declare our argument pointer
    va_list argPtr;

    // Grab the first argument
    va_start(argPtr, count);

    int sum = 0;
    for (; count > 0; --count) {
        sum += va_arg(argPtr, int);
    }
    va_end(argPtr);

    return sum;
}
```
Variable-length arguments

Iteration: Sentinel value

Example

```c
#include <stdarg.h>
int AllSum(int arg1, ...) {
    // Declare our argument pointer
    va_list argPtr;

    // Grab the first argument
    va_start(argPtr, arg1);

    int arg, sum = 0;
    for (arg = arg1; arg; arg = va_arg(argPtr, int)) {
        sum += arg;
    }
    va_end(argPtr);

    return sum;
}
```
Writing programs

Return values
Arguments
Writing Programs

Return values

- In a standard C environment, there is an Operating System
- Programs are started, execute, and end within the OS
- The return value allows for a program to return a code indicating its operation
- Most useful when writing daemons or programs that are not directly executed by the user
Writing Programs

Return values

- Returning 0 indicates successful operation
- Returning non-zero indicates error

Example

```c
int main(void)
{
    return 0;
}
```
Writing Programs

Return values

- `<stdlib.h>` defines `EXIT_SUCCESS` and `EXIT_FAILURE`

Example

```c
int main(void)
{
    return EXIT_SUCCESS;
}
```
Writing Programs

Return values

Syntax

```c
void exit(int status);
```

- Defined in `<stdlib.h>`

Example

```c
int main(void)
{
    exit(EXIT_FAILURE);

    return EXIT_SUCCESS;
}
```
Writing Programs

Program arguments

• Programs can take a variable number of arguments
  – Just like functions

• The number of arguments is known

• Only makes sense in a multi-process environment
  – Doesn't work with XC32
Writing Programs

Program arguments

Syntax

```c
int main(int argc, char *argv[]);
```

- Arguments are passed as strings
- First argument is the program name

Example

```
ls -hal ~
mkdir .ssh
ln -s ~/Dropbox/config/.ssh .ssh
```
Writing Programs

Program arguments

```
ln -s ~/Dropbox/config/.ssh .ssh
```

Syntax

```
int main(int argc, char *argv[]);
```
Writing Programs

Example: Output all program arguments

```c
int main(int argc, char *argv[]) {
    int i;
    for (i = 0; i < argc; ++i) {
        printf("%s ", argv[i]);
    }

    return EXIT_SUCCESS;
}

if (argc < 4) exit
```
CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne

Spring 2016
File I/O
File I/O

- Most data on computers are stored in files
- So accessing data reads and writes to these files
- And in a Unix environment, everything is a file
  - Serial ports
  - Network connections
  - Hard drives
  - Displays
- So everything can be controlled via file access
File I/O

Standard files

- Three special files that are automatically opened and closed
  - `stdin`: standard input (keyboard/serial port)
  - `stdout`: standard output (screen)
  - `stderr`: standard error (screen)
File I/O
The standard library

- `<stdio.h>` contains functions for working with files
- Its concept of a file includes:
  - Filename
  - File access mode
  - File size
  - Current position
File I/O
Using files

- Files are opened with `fopen()`
- Files are read and written to:
  - `fprintf()`, `fscanf()` – Formatting strings
  - `fputc()`, `fgetc()` – Characters
  - `fputs()`, `fgets()` – Lines
  - `fread()`, `fwrite()` – Blocks
- Files are closed with `fclose()`
File I/O
Using files

• Only a limited number of files can be opened at a time
  – Per process
  – Also per OS
• Very large on modern Oses
  – \( \geq 2048 \) usually
• For the XC32: 8

-\( \text{fcloser() did nothing} \)
The standard library uses a single struct to store the metadata of the file.

```c
typedef struct _iobuf {
    char * _ptr;
    int _cnt;
    char * _base;
    unsigned short _flag;
    short _file;
    size_t _size;
} FILE;
```
File I/O

fopen()

Syntax

```c
FILE *fopen(const char *name, const char *mode);
```

- **name** is a C string with the filename
- **mode** is the mode to open the file in
  - "r" opens for reading
  - "w" opens for writing
  - "a" opens for appending
  - "b" specifies binary

- Returns the file pointer
ASCII

UNICODE

'a'

'π'

16 bits
## File I/O

### File modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r</code></td>
<td>Open a text file for reading.</td>
</tr>
<tr>
<td><code>w</code></td>
<td>Truncate to zero length or create a text file for writing.</td>
</tr>
<tr>
<td><code>a</code></td>
<td>Append; open or create a text file for writing at the end-of-file.</td>
</tr>
<tr>
<td><code>rb</code></td>
<td>Open a binary file for reading.</td>
</tr>
<tr>
<td><code>wb</code></td>
<td>Truncate to zero length or create a binary file for writing.</td>
</tr>
<tr>
<td><code>ab</code></td>
<td>Append; open or create a binary file for writing at the end-of-file.</td>
</tr>
<tr>
<td><code>r+</code></td>
<td>Open a text file for read/write.</td>
</tr>
<tr>
<td><code>w+</code></td>
<td>Truncate to zero length or create a text file for read/write.</td>
</tr>
<tr>
<td><code>a+</code></td>
<td>Append; open or create a text file for read/write. You can read data anywhere in the file, but you can write data only at the end-of-file.</td>
</tr>
<tr>
<td><code>r+b</code> or <code>rb+</code></td>
<td>Open a binary file for read/write.</td>
</tr>
<tr>
<td><code>w+b</code> or <code>wb+</code></td>
<td>Truncate to zero length or create a binary file for read/write.</td>
</tr>
<tr>
<td><code>a+b</code> or <code>ab+</code></td>
<td>Append; open or create a binary file for read/write. You can read data anywhere in the file, but you can write data only at the end-of-file.</td>
</tr>
</tbody>
</table>
File I/O
fread()

Syntax
```c
size_t fread(void *ptr, size_t size,
             size_t count, FILE *stream);
```

- **ptr** – The buffer to write into
- **size** – The size of each element to read
- **count** – The number of elements to read
- **stream** – The pointer to the file
- Returns the number of elements read
  - Less than count indicates error or EOF
File I/O

Syntax

```c
size_t fread(void *ptr, size_t size,
             size_t count, FILE *stream);
```

- **ptr** – The buffer to write into
- **size** – The size of each element to read
- **count** – The number of elements to read
- **stream** – The pointer to the file
- Returns the number of elements read
  - Less than count indicates error or EOF
File I/O

feof()

Syntax

```c
intfeof(FILE *stream);
```

- `stream` – The pointer to the file
- Returns a non-zero value if the stream is at the end of the file, 0 otherwise
File I/O

fseek()

Syntax

```c
int fseek(FILE *stream, long offset, int origin);
```

- **stream**  – The pointer to the file
- **offset**  – The bytes to move from the current location
- **origin**  – The reference location: either SEEK_SET, SEEK_CUR, or SEEK_END
- Returns 0 if successful, otherwise returns a non-zero value
File I/O

fclose()

Syntax

```c
int fclose(FILE *stream);
```

- `stream` – The pointer to the file
- Returns 0 if successful, otherwise returns EOF
  - EOF is a macro, generally -1
Example

```c
int main(void)
{
    // Open the file, terminating if there was an error
    FILE *pFile = fopen("/room1.txt", "rb");
    if (pFile == NULL) {
        puts("Error opening file.");
        return EXIT_FAILURE;
    }

    // Count the characters in the file.
    int n = 0;
    while (fgetc(pFile) != EOF) {
        ++n;
    }

    // Output the results, if we succeeded
    if (feof(pFile)) {
        printf("Total bytes read: %d\n", n);
        fclose(pFile);
        return EXIT_SUCCESS;
    }

    // Otherwise output an error
    puts("Error occurred before reading end of file.");
    fclose(pFile);
    return EXIT_FAILURE;
}
```
File formats
File formats

Types

- Two groups:
  - Text
  - Binary

- Text are easier to process, but larger
- Binary are harder to process, but smaller
- Many formats are now zipped text files so the data is easy to parse, but the size is small
  - .docx/.xlsx for example
<metaData>
  <messageInfo name = "System Time" pgn = "126992" size = "8">
    <desc>Represents the current data and time</desc>
    <field
      name = "Days since epoch"
      type = "int"
      offset = "16"
      length = "16"
      signed = "no"
      units = "days"
      endian = "little"
    />
  </messageInfo>

  <messageInfo name = "Rudder" pgn = "127245" size = "6">
    <desc>Represents the current rudder position</desc>
    <field
      name = "Position"
      type = "int"
      offset = "32"
      length = "16"
      signed = "yes"
      units = "rad"
      scaling = "0.0001"
      endian = "little"
    />
  </messageInfo>
</metaData>
File formats

Text: CSV
File formats

Binary: ZIP

FILE ENTRY 1 <data>
Relative offset 1
Local header 1

FILE ENTRY 2 <data>
Relative offset 2
Local header 2

FILE ENTRY 3 <data>
Relative offset 3
Local header 3

FILE ENTRY 4 <data>
Relative offset n
Local header n

CENTRAL DIRECTORY
File entry 1
File entry 2
File entry 3

<data>
File formats

Binary: RPG

• Needed a format to store each room in a dungeon

• Requirements
  – Title
  – Description
  – Items in the room
  – Exits:
    • Which room
    • What direction
File formats

Binary: RPG

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
</table>

File formats

Binary: RPG

- But it would be cool if the rooms could change depending on items the player has encountered
  - Like keys
- So we want different versions of the room for:
  - Description
  - Items
  - Exits
## File formats

**Binary: RPG**

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(repeated)
File formats

Binary: RPG

<table>
<thead>
<tr>
<th>size</th>
<th>ASCII data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
File formats

Binary: RPG

<table>
<thead>
<tr>
<th>size</th>
<th>binary data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Item requirements</td>
</tr>
</tbody>
</table>
File formats

Binary: RPG

<table>
<thead>
<tr>
<th>size</th>
<th>ASCII data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Item requirements</td>
</tr>
</tbody>
</table>
File formats

Binary: RPG

<table>
<thead>
<tr>
<th>size</th>
<th>binary data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Item requirements</td>
</tr>
</tbody>
</table>

Maxwell James Dunne – Spring 2016
File formats

Binary: RPG

<table>
<thead>
<tr>
<th>North</th>
<th>East</th>
<th>South</th>
<th>West</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Item requirements</td>
<td>Description</td>
<td>Items contained</td>
</tr>
</tbody>
</table>

Exits
# File formats

**Binary: RPG**

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1</td>
<td>Requires key, no items</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 2</td>
<td>No requirements, contains key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/Room32.txt
## File formats

**Binary: RPG**

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 The Throne Room</td>
<td>Version 1: Requires key, no items</td>
<td>Version 2: No requirements, contains key</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/Room32.txt
### File formats

**Binary: RPG**

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Version 1: Requires key, no items

Version 2: No requirements, contains key

/Room32.txt
A large metal throne forged of swords of previous kings sits prominently here. Your dad is rarely in it, however, instead ruling the kingdom from his council's chambers.

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1</td>
<td>Requires key, no items</td>
<td>Version 2: No requirements, contains key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/Room32.txt
File formats

Binary: RPG

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1</td>
<td>Requires key, no items</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 2</td>
<td>No requirements, contains key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/Room32.txt
## File formats

**Binary: RPG**

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Version 1: Requires key, no items

<table>
<thead>
<tr>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Version 2: No requirements, contains key

/Room32.txt
File formats

Binary: RPG

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1:</td>
<td>Requires key, no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>items</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 2:</td>
<td>No requirements,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>contains key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/Room32.txt
A large metal throne forged of swords of previous kings sits prominently here. Your dad is rarely in it, however, instead ruling the kingdom from his council’s chambers. You feel the weight of the castle key stolen earlier in your pocket.

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1: Requires key, no items</td>
<td>Version 2: No requirements, contains key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/Room32.txt
## File formats

**Binary: RPG**

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Version 1**: Requires key, no items
- **Version 2**: No requirements, contains key

/Room32.txt
## File formats

**Binary: RPG**

<table>
<thead>
<tr>
<th>Title</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
<th>Item requirements</th>
<th>Description</th>
<th>Items contained</th>
<th>Exits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Version 1:** Requires key, no items

**Version 2:** No requirements, contains key

/Room32.txt