Enumerations

Symbolic constants

- An enumeration is a set of named integers
- Provide mnemonic names for “magic” numbers
  - Passing and returning encoded values
  - Creating bit values, which may be or’ed together

```c
enum tag { enumeration_list } variable_list;
```

- `tag` is optional and follows the rules for creating a legal identifier
- `variable_list` is optional and often omitted; if present, it defines variables of type `enum tag`
- `enumeration_list` is a comma separated list of legal identifiers
  - The first identifier is assigned the value 0
  - The value of one or more identifiers may be specified
  - Unless specified, the value of an identifier is always 1 greater than the last

```
enum examples

enum { uread = 1, uwrite = 2, uexe = 4, gread = 8, gwrite = 16,
      gexe = 32, oread = 64, owrite = 128, oexe = 256
      };
```

```c
enum commands command = getcommand( );
switch(command)
{ case e_Quit : exit(0);
  case e_Search : search(personFile, indexFile);
  break;
  case e_Input : input();
  break;
  case e_Import : import(personFile, indexFile);
  break;
  case e_Dump : dump(personFile);
  break;
  case e_Help  : help();
  break;
  default : fprintf(stderr, "UNKNOWN command
");
  break;
}
```

typedef Statements

Creating type aliases

- A `typedef` statement creates an alias or new name for an existing type (i.e., does not create a new type)
  - A “shorthand” notation
  - More mnemonic or self documenting
  - Improve portability by creating machine-dependent types (see `sys/types.h`)

```c
typedef old_name new_name;
```

```
typedef unsigned long ulong;
typedef int boolean;
typedef unsigned int size_t;
ulong counter;
size_t length;
```
Recursion Example

(Print the digits of an integer one at a time)

```c
void forward(int number)
{ if (number != 0)
    { forward(number / 10);
      printf("%d", number % 10);
    }
}
```

```c
void reverse(int number)
{ if (number != 0)
    { printf("%d", number % 10);
      reverse(number / 10);
    }
}
```

Graphical Representation

(Activation records and statement sequencing)

<table>
<thead>
<tr>
<th>4th call</th>
<th>3rd call</th>
<th>2nd call</th>
<th>1st call</th>
</tr>
</thead>
<tbody>
<tr>
<td>number == 0</td>
<td>number == 1</td>
<td>number == 12</td>
<td>number == 123</td>
</tr>
<tr>
<td>3 forward(0)</td>
<td>4 print(1)</td>
<td>2 forward(1)</td>
<td>1 forward(12)</td>
</tr>
<tr>
<td>5 print(1)</td>
<td>6 reverse(0)</td>
<td>3 print(2)</td>
<td>6 print(3)</td>
</tr>
</tbody>
</table>

prints digits forward prints digits reversed