Symbolic Data & Functions

Hour 18

Objectives

- Symbolic constants with enumerations
- Type aliases with the typedef statement
- Recursion

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
- `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 if 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

"Magic numbers" and bit-patterns

```c
enum commands {
    e_Unknown, /* unknown command entered */
    e_Quit, /* close files, quite database */
    e_Search, /* search for a person */
    e_Input, /* input a new person */
    e_Import, /* import data from another file */
    e_Dump, /* dump data base in ascii/text */
    e_Help /* show valid commands list */
};
```

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

### enum Examples

Continued

```c
enum commands command = getcommand();
```

```c
switch(command)
{
    case e_Quit : exit(0);
    case e_Search : search(personFile, indexFile);
    case e_Input : input();
    case e_Import : import(personFile, indexFile);
    case e_Dump : dump(personFile);
    case e_Help : help();
    default : fprintf(stderr, "UNKNOWN command\n");
}
```
**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>)

**General format:**
```c
typedef old_name new_name;
```
- `typedef unsigned long ulong;`
- `typedef int boolean;`
- `typedef unsigned int size_t;`
- `ulong counter;`
- `size_t length;`

---

**Recursive Functions**

Essential elements of recursion

- **Direct recursion**: a function calls itself
- **Indirect recursion**: A calls B, B calls C, ..., Y calls Z, Z calls A
- **Must** have at least two conditional parts:
  - Recursive call
  - Termination (i.e., non-recursive) condition (may be implicit rather than explicit)
- Input must change with each call
- Theoretically, may be written as a loop
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);
    }
}

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

Graphical Representation

Activation records and statement sequencing

```
4th call
  number == 0

3rd call
  number == 1
  3 forward(0)
  4 print(1)

2nd call
  number == 12
  2 forward(1)
  5 print(2)

1st call
  number == 123
  1 forward(12)
  6 print(3)
```

prints digits forward

```
4th call
  number == 0

3rd call
  number == 1
  5 print(1)
  6 reverse(0)

2nd call
  number == 12
  3 print(2)
  4 reverse(1)

1st call
  number == 123
  1 print(3)
  2 reverse(12)
```

prints digits reversed