Computer programming

"Education is a progressive discovery of our own ignorance."

Will Durant

Outline

- Structure, union, enumeration and user defined data types
  - Definitions
  - User defined types
  - Example programs

Structures

- Structures = collections of related variables (aggregates) under one name
  - Can contain variables of different data types
  - Similar to record data type of PASCAL
  - Commonly used to define records to be stored in files
  - Combined with pointers, can create linked lists, stacks, queues, and trees

Structures

- Definition in BNF:

```
structure_definition ::= struct⌈name_identifier⌋
                        {
                             member_list;
                        } ⌈variable_identifiers⌋;
member_list::=type member_identifier, {,
   member_identifier};
   { type member_identifier, {, member_identifier};}
variable_identifiers::=identifier{, identifier}
```

- Note that in this definition name_identifier and variable_identifiers cannot both be missing;
- A structure variable of type name_identifier can be later declared using:

```
structure_variable_declaration::=⌈struct⌋ name_identifier
   variable_identifier;
```

Structures. Equivalent examples

a.

```
struct material
{
   long code;
   char name[30];
   char uom[10];
   float amount;
   float ppu;
} fabric, paper, engine;
```

b.

```
struct material
{
   long code;
   char name[30];
   char uom[10];
   float amount;
   float ppu;
};
struct material fabric, paper, engine;
```

or

```
material fabric, paper, engine;
```

c.

```
struct
{
     long code;
     char name[30];
     char uom[10];
     float amount;
     float ppu;
} fabric, paper, engine;
```

Structures

- `struct` information
  - A `struct` cannot contain an instance of itself
  - Can contain a member that is a pointer to the same structure type
  - A structure definition does not reserve space in memory
    - Instead creates a new data type used to define structure variables
- Valid Operations
  - Assigning a structure to a structure of the same type
  - Taking the address (&) of a structure
  - Accessing the members of a structure
  - Using the `sizeof` operator to determine the size of a structure

Structures

- Access to members – by qualification
- Example:

```
    fabric.name
    paper.amount
```

- Access to members using pointers:

```
 pointer_to_structure->
```

- Example:

```
    struct material *p;
    p=&fabric;
    p->amount = 20.5;  or
    (*p).amont = 20.5;
```

- Structure variables can be allocated like the other variables (simple variables and arrays)

Structure initialization

- Initialization is similar to array initialization

```
struct type_name variable_identifier=
{
    value_of_member_1,      value_of_member_2, ...,
    value_of_member_n
};
```

- Example:

```
struct material fabric = {1234L, "Wool fabric",
"meter", 25.5, 45.3};
```

- Members of
  - global and static structures are initialized to zero by default
  - automatic structures have undefined values

Structures

- Notes
  - In C++ one can assign structure variables of the same type, e.g.

    ```
    material alpha, beta;
    alpha=beta;
    ```

  - Can define recursive structures (heavily used to generate dynamic lists, trees, etc), e.g.

```
struct nodeType
{
  long code;
  char name[30];
  struct nodeType *next;
} Node;
```

Using structures with functions

- Passing structures to functions
  - **In C one can pass a structure only by its address**
  - In C++ pass entire structure, e.g. `void f(material p, ...)`
    - Or, pass individual members, e.g. `void f(float p.amount, ...)`
    - Both pass call by value
  - In C++ to pass structures call-by-reference
    - Pass its address: `void f(material *p, ...)`
    - Pass reference to it: `void f(material& p, ...)`
  - In C++ to pass arrays call-by-value
    - Create a structure with the array as a member
    - Pass the structure

Bit Fields (I)

- Bit field
  - Member of a structure whose size (in bits) has been specified
  - Enable better memory utilization
  - Must be defined as `int` or `unsigned`
  - Cannot access individual bits

Bit Fields (II)

- Defining bit fields
  - Follow `unsigned` or `int` member with a colon (`:`) and an integer constant representing the width of the field
  - Example:

```
struct BitCard {
   unsigned face : 4;
   unsigned suit : 2;
   unsigned color : 1;
};
```

Bit Fields (III)

- Unnamed bit field
  - Field used as padding in the structure
  - Nothing may be stored in the bits

```
struct Example {
   unsigned a : 13;
   unsigned   : 3;
   unsigned b : 4;
}
```

  - Unnamed bit field with zero width aligns next bit field to a new storage unit boundary
  - Demos

Unions

- **`union`**
  - Memory that contains a variety of objects over time
  - Only contains one data member at a time
  - Members of a `union` share space
  - Conserves storage
  - Only the last data member defined can be accessed
- **`union`** definitions
  - Same as struct

```
union Number
{
  int x;
  float y;
};
union Number value;
```

Unions

- Valid `union` operations
  - Assignment to `union` of same type: =
  - Taking address: &
  - Accessing union members: .
  - Accessing members using pointers: ->
- Initialization:
  - **In C: not allowed**
  - In C++: first member can be initialized

Enumeration

- Enumeration
  - Set of integer constants represented by identifiers
  - Enumeration constants are like symbolic constants whose values are automatically set
    - Values start at `0` and are incremented by `1`
    - Values can be set explicitly with `=`
    - Need unique constant names
  - Example:

    ```
    enum Months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG,
      SEP, OCT, NOV, DEC};
    ```

    - Creates a new type `enum Months` in which the identifiers are set to the integers `1` to `12`
  - Enumeration variables can only assume their enumeration constant values (not the integer representations)

Enumeration

- BNF definition:

```
enum_definition::=
    enum⌈name_identifier⌋
    { enumerator_list} ⌈variable identifiers⌋;
enumerator_list::=
    enumerator_identifier {, enumerator_identifier}
variable_identifiers::=identifier{, identifier}
```

- Equivalent examples:

a.
```
enum week {Sunday, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday};
enum week holiday_week;
```

b.
```
enum week {Sunday, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday} holiday_week;
```

c.
```
enum {Sunday, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday} holiday_week;
```

- Possible assignments:

```
holiday_week=Friday; ... holiday_week=Sunday;
```

User-defined data types

- Type definition uses keyword `typedef`

  `type_definition::=typedef type type_identifier`

  - Examples:

a.
```
typedef struct
{
  int i;
  float j;
  double x;
} AlphaT;
AlphaT y, z;
```

b.
```
typedef struct complex
{
  float re;
  float im;
  struct complex *next;
} ComplexT;
ComplexT x, y;
```

c.
```
typedef union
{
  char x[10];
  long code;
} BetaT;
BetaT u, v;
```

d.
```
typedef enum {false, true}
BooleanT;
BooleanT k, l;
```

Example programs. Structures. Operations on complex numbers

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {float re, im; }
   ComplexT;
/* parameters passed by pointers
   */
void  add(ComplexT *a, ComplexT
   *b,           ComplexT *c)
{
   c->re = a->re + b->re;
   c->im = a->im + b->im;
}
void  subtract(ComplexT a,
   ComplexT b, ComplexT *c)
/* input parameters passed by
   value, allowed in C++ only,
    result passed by pointer */
{
   c->re = a.re - b.re;
   c->im = a.im - b.im;
}
```

```
/* input parameters passed by value,
    result passed by reference, all
    allowed in C++ only */

void multiply(ComplexT a, ComplexT b,
    ComplexT& c)

{
    c.re = a.re * b.re - a.im * b.im;
    c.im = a.im * b.re + a.re * b.im;
}
/* parameters passed by pointers */
void  divide(ComplexT *a, ComplexT *b,
                 ComplexT *c)
{
    float x;
    x = b->re * b->re + b->im * b->im;
    if ( x == 0 )
    {
        puts("\nDivision by zero");
        exit(1);
    }
    else
    {
        c->re = (a->re * b->re +
                 a->im * b->im) / x;
        c->im = (a->im * b->re -
                 a->re * b->im) / x;
    }
}
```

Example programs. Structures. Operations on complex numbers

```
void  readComplex(const char *msg, const char *nbName, ComplexT *nb)
{
    printf("\nPlease input the %s complex number\n\t%s.re=",
               msg, nbName);
    scanf("%f%*c", &(nb->re));
    printf("\t%s.im=", nbName);
    scanf("%f%*c", &(nb->im));
}

int main(int argc, char *argv[])
{
    ComplexT a, b, c;
    char ch;

    ch = 'Y';
    while (ch == 'Y' || ch == 'y')
    {
        readComplex("first", "a", &a);
        readComplex("second", "b", &b);
```

Example programs. Structures. Operations on complex numbers

```
        add(&a, &b, &c);
        printf("\na + b = (%f+j*%f) + (%f+j*%f) = (%f+j*%f)",
              a.re, a.im, b.re, b.im, c.re, c.im);
        subtract(a, b, &c);
        printf("\na - b = (%f+j*%f) - (%f+j*%f) = (%f+j*%f)",
                a.re, a.im, b.re, b.im, c.re, c.im);
        multiply(a, b, c);
        printf("\na * b = (%f+j*%f) * (%f+j*%f) = (%f+j*%f)",
                a.re, a.im, b.re, b.im, c.re, c.im);
        divide(&a, &b, &c);
        printf("\na / b = (%f+j*%f) / (%f+j*%f) = (%f+j*%f)",
                a.re, a.im, b.re, b.im, c.re, c.im);
        printf("\nContinue [Y/N]? ");
        ch = getchar();
    }
    return 0;
}
```

Example programs. Unions

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
union
{
    char ch[10];
    short int i;
    long l;
    float f;
    double d;
} all;
int main(void)
{
    strcpy(all.ch, "ABCDEFGHIJ");
    printf("\nThe size of the union is %u bytes\n", sizeof(all));
    printf("Area contents:\n\t- as a character string: %s\n", all.ch);
    printf("\t- as an integer of type short int: %d\n", all.i);
    printf("\t- as an integer of type long: %ld\n", all.l);
    printf("\t- as a real of type float: %f\n", all.f);
    printf("\t- as a real of type double: %g\n", all.d);
    system("pause");
    return 0;
}
```

Example programs. Enumerations

```c
#include <stdio.h>
#include <stdlib.h>
typedef enum {zero, one ,two, three, four, fine} NUM;
int main(void)
{
    NUM x, y;
    int z, w;

    x = two; /* x=2 */
    y = three; /* y = 3 */
    z = x + y;
    w = x * y;
    printf("\nz=%d w=%d\n", z, w);
    system("pause");
    x = 2; y = 3; /* cause warnings */
    z = x + y;
    w = x * y;
    printf("\nz=%d w=%d\n", z, w);
    system("pause");
    return 0;
}
```

The Data Hierarchy

- Bit - smallest data item
  - Value of **0** or **1**
- Byte – 8 bits
  - Used to store a character (decimal digits, letters, and special symbols)
- Field - group of characters conveying meaning
  - Example: an address, a name
- Record – group of related fields
  - Represented a **`struct (C,C++)`** or a **`class (C++)`**
  - Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.
- File – group of related records
  - Example: payroll file
- Database – group of related files

Files

- Text files
  - Characters are (usually) human readable
  - Are divided into lines
  - May contain a special “End-of-file” marker
  - Examples: source files, header files
- Binary files
  - None of the above characteristics
  - Examples: executables, object files, databases, etc.

Error Handling

- System calls set a *global* integer called `errno` on error:
  - `extern int errno;  /* defined in errno.h */`
- The constants that errno may be set to are defined in `<errno.h>`. For example:

`EPERM`	operation not permitted
`ENOENT`	no such file or directory (not there)
`EIO`	I/O error
`EEXIST`	file already exists
`ENODEV`	no such device exists
`EINVAL`	invalid argument passed

```
#include <stdio.h>
void perror(const char * s);
```

- E.G. if `errno==EINVAL`, then `perror("EINVAL: ");` prints:

```
EINVAL: : Invalid argument
```

File status

- `int stat(const char * pathname, struct stat *buf);`
- The `stat()` system call returns a structure (into a buffer you pass in) representing all the stat values for a given filename. This information includes:
  - the file’s mode (permissions)
  - inode number (in Unix)
  - number of hard links
  - user id of owner of file
  - group id of owner of file
  - file size
  - last access, modification, change times
  - see header file `stat.h` and `sys/types.h` (`S_IFMT`, `S_IFCHR`, etc.)

File status example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <time.h>
int main(int argc, char *argv[])
{
    char *fname="main.c";
    struct stat statBuf;

    if (0==stat(fname, &statBuf))
    {
      printf("\nFile: %s", fname);
      if (S_ISDIR (statBuf.st_mode))
         puts("\n\tdirectory");
      if (S_ISCHR(statBuf.st_mode))
         puts("\n\tcharacter special file");
      if (S_ISBLK(statBuf.st_mode))
         puts("\n\tblock special file");
      if (S_ISREG(statBuf.st_mode))
         puts("\n\tregular file");
      if (S_ISFIFO(statBuf.st_mode))
         puts("\n\tFIFO special file, or a pipe");
      printf("\tinode=%u\n\tdevice=%c
:", statBuf.st_ino,
statBuf.st_dev+'A');

      printf("\n\tlinks=%d\n\tuid=%d\
n\tgid=%d\n\tsize=%ld",
          statBuf.st_nlink,
statBuf.st_uid, statBuf.st_gid,
statBuf.st_size);
      printf("\n\taccess
time=%s\tcontents mod
time=%s\tattrib mod time=%s",
          ctime(&statBuf.st_atime),
ctime(&statBuf.st_mtime),
          ctime(&statBuf.st_ctime));
    }
    else perror(0);
  system("PAUSE");
  return 0;
```

File status example

- Program output example:

```
File: main.c
        regular file
        inode=0
        device=D:
        links=1
        uid=0
        gid=0
        size=1007
        access time=Thu Dec 21 08:47:51 2005
        contents mod time=Thu Dec 21 08:47:51 2005
        attrib mod time=Thu Dec 21 08:47:51 2005
```

Reading

- Deitel: chapter 10, chapter 11 sections 11.1 to 11.3
- Prata: chapters 14 & 15
- King: chapters 16 & 20

Summary

- Structure, union, enumeration and user defined data types
  - Definitions
  - User defined types
  - Example programs