Data Types Structure, Union, Enumerations

1. Overview

The learning objective of this lab session is to:

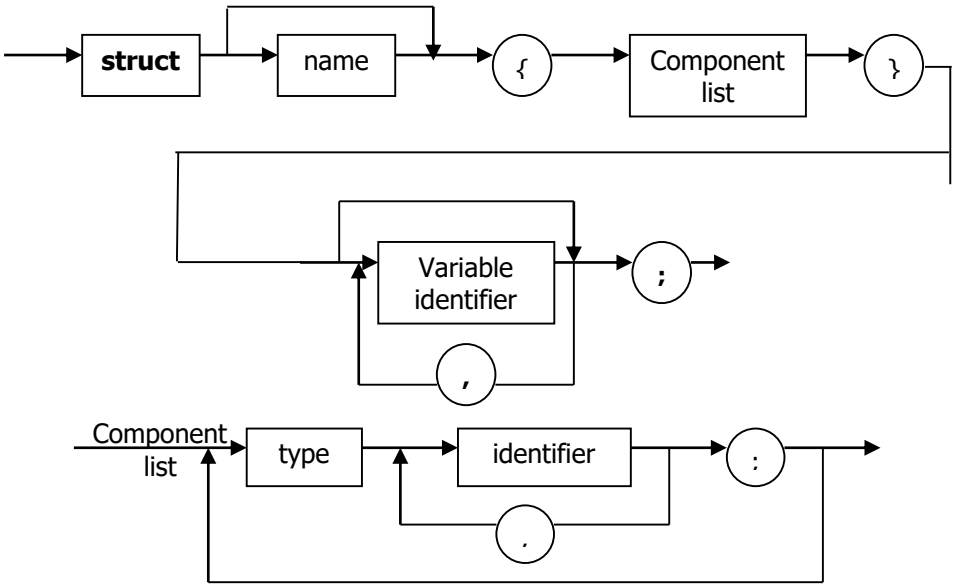
- Understand the user-defined types structure, union, and enumeration
- Learn how to access the components of such types
- Learn how to assign names for these types
- Acquire skills in using such type in programs.

2. Brief theory reminder

2.1. The "struct" data type

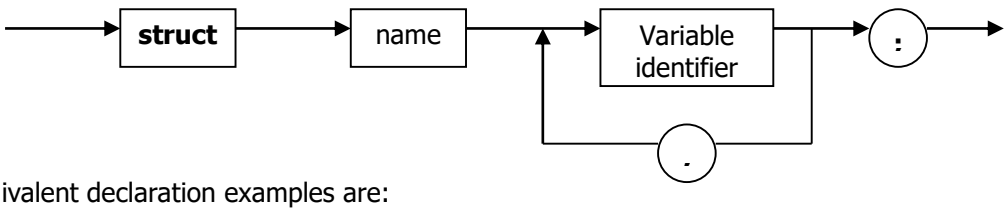
A **structure** contains a number of components of different types (predefined or user defined), grouped according to a hierarchy.

A structure is declared according to the following pattern:



Note. In this declaration **name** and **identifier** cannot both be missing.

A structure variable of type **name** can be later declared as the diagram below illustrates. In C++ the **struct** keyword may be missing.



Equivalent declaration examples are:

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

b)
```
struct item {
        long code;
        char name[30];
        char uom[10];
        float amount;
        float unit_price;
};
```

struct item fabric, paper, engine;

---

c) struct {
long code;
char den[30];
char uom[10];
float amount;
float unit_price;
} fabric, paper, engine;

Access to the components of a structure can be achieved by **qualification**:

variable_identifier.field_identifier;

Examples: **fabric.name**
paper.amount

In C, if a structure is used as a parameter in a function call, then this can be done by passing the address (a pointer to) the structure variable. For example, for the function defined as:

```
void f (struct item *p, ...)
{
        ...
}
```

a call should be like:

f(&fabric, ...);

In the body of function **f** above, selection of a field is done like this:

p->name
p->amount

Using (*fabric) instead of fabric-> , as below:

(*p).name
(*p).amount

In C++ a structure can be passed as a parameter in three ways:

- directly: **void f (item p, ..)**
- pointer to structure: **void f (item *p, ...)**
- reference to structure: **void f (item &p, ...)**

A variable of type structure can be assigned to another if they are both of the same type, like in the example below:

```
struct item alpha, beta;
alpha=beta; /* correct */
```

2.2. The "union" data type

At different times during execution, the same memory area can hold different types. This is useful for saving memory or for other reasons. This can be achieved by grouping all data which will be

allocated the same memory area. The user type obtained in this way is called **union**. The syntax for **union** is identical to that used for **struct**, the only difference being the keyword used.

We must mention that the memory area which is reserved has a size equal to the size needed to store the component which takes the biggest amount of memory.

The syntax for accessing a component is identical with the one used for structures.

Note. The programmer must at all times be aware which data is actually store in the union.

Example:

```
union alpha
{
        char c[5];      /* uses 5 octets */
        int i;          /* uses 2(4) octets depending on implementation */
        long j;         /* uses 4 bytes */
};
union alpha x;
strcpy(x.c, "ABCD");
```

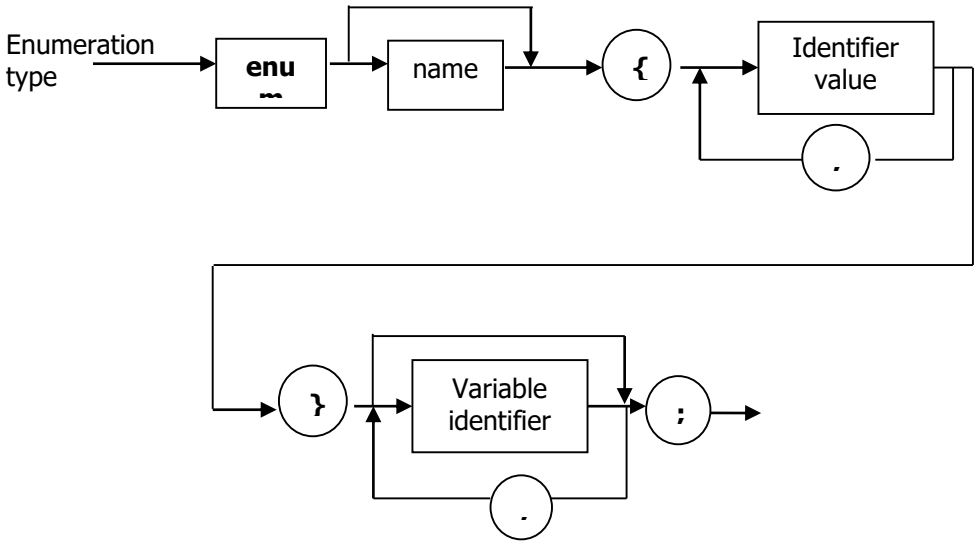
The memory area used to store variable **x** is, in hexadecimal:

41	42	43	44	00

If component **x.i** is accessed, then this component will have the hex value 4241, i.e.16961 decimal if the **int** type is represented on 2 bytes, or 44434241 hex, 1145258561 decimal, same as **x.j** if the **int** type uses 4 bytes (same as long).

2.3. The "enum" data type

The enumeration type enables the programmer to use suggestive names for numeric values. The syntax diagram for enumeration is similar with the one for **struct** and **union**. The difference consists in the fact that the component list contains only integer-valued identifiers:



Equivalent examples are:

a) enum week {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};
enum week week_holiday;

b) enum week {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
week_holiday;

c) enum {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday} week_holiday;

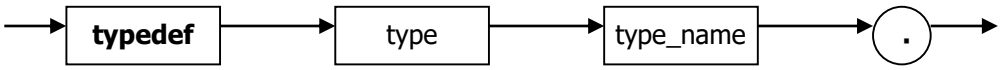
One can assign:

week_holiday=Friday;

The identifiers are assigned values Monday=0, Tuesday=1, …

2.4. Declaring data types using symbolic names

In C/C++ one can assign a symbolic name to a predefined type, or to a user-defined type. The syntax diagram for assigning a symbolic name "type_name" to a predefined or user-defined type is the following:



Examples:

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

b)
```
typedef struct
{
    float re;
    float im;
} 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;
```

2.5. Example programs

The following program illustrates operations on complex numbers, using the struct type. All possibilities for passing structures as parameters are shown.

```
/*Program L9Ex1.cpp */

#include <stdio.h>
#include <process.h>
typedef struct
{
    float re,
    im;
} ComplexT;

void add(ComplexT *a, ComplexT *b, ComplexT *c)
 /* parameters passes as pointers */
{
        c->re=a->re+b->re;
        c->im=a->im+b->im;
};
void subtract(ComplexT a, ComplexT b, ComplexT *c)
```

```
/* parameters passed by values (can be done only in C++) and result is passed as a
    pointer */
{
    c->re=a.re-b.re;
     c->im=a.im-b.im;
}

void multiply(ComplexT a, ComplexT b, ComplexT &c)
/* parameters passed by values and result is passed as a  reference (can be done only in
C++) */
{
    c.re=a.re*b.re-a.im*b.im;
    c.im=a.im*b.re+a.re*b.im;
}

void divide(ComplexT *a, ComplexT *b, ComplexT *c)
/* parameters passed as pointers */
{
     float x;

     x=b->re*b->re+b->im*b->im;
     if (x==0)
     {
         printf("\nDivision by zero!\n");
         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;
    }
}

 /* Operations on ComplexT numbers */
 int main()
{
    ComplexT a, b, c;
         char ch, op;
         ch='Y';
    while ( ch=='Y' || ch=='y' )
    {
         printf("\nInput the first ComplexT number:\n");
         printf("a.re=");
         scanf("%f",&a.re);
         printf("a.im=");
         scanf("%f",&a.im);
         printf("\nInput the second ComplexT number:\n");
         printf("b.re=");scanf("%f",&b.re);
         printf("b.im=");scanf("%f",&b.im);

         add(&a, &b, &c);
         printf("\n(%f+j*%f)+(%f+j*%f)=%f+j*%f\n",
                    a.re, a.im, b.re,b.im, c.re, c.im);

         subtract(a, b, &c);
         printf("\n(%f+j*%f)-(%f+j*%f)=%f+j*%f\n",
                  a.re, a.im, b.re, b.im, c.re, c.im);
```

```
        multiply(a, b, c);
        printf("\n(%f+j*%f)*(%f+j*%f)=%f+j*%f\n",
                    a.re, a.im, b.re, b.im, c.re, c.im);
        divide(&a, &b, &c);
        printf("\n(%f+j*%f)+(%f+j*%f)=%f+j*%f\n",
                    a.re, a.im, b.re, b.im, c.re, c.im);
                printf("\nContinue [Yes=Y/y, No=other character]? " );
                scanf("%*c%c", &ch);
    }
        return 0;
}
```

The following program shows operations on data of type "union":

```
/* Program L9Ex2.c */

#include <stdio.h>
#include <string.h>
/* Example of using type "union" */
int main()
{
   typedef union
   {
      char ch[10];
      int x;
      long y;
      float f;
   } alpha;
   alpha  a;

   strcpy(a.ch, "ABCDEFGHI");
   printf("\nThe size of the memory area occupied by a is %d bytes\n", sizeof a);
   printf("\narea contents:\n");
   printf("\n-character string: %s", a.ch);
   printf("\n-integer of type int: %d(%x in hex)", a.x, a.x);
   printf("\n-integer of type long: %ld(%lx in hex)",a.y, a.y);
   printf("\n-real of type float: %g",a.f);
   getchar();
   return 0;
}
```

The following program shows operations on data of type "enum".

```
/* Program L9Ex3.c */

#include <stdio.h>
/* Example usage of the "enum" type */
int main()
{
    typedef enum{zero, one, two, three, four, five} Nb;
    Nb x, y;
    int z, w;

    x=two;    /* x=2 */
    y=three;    /*x=3*/
    z=x+y;
    w=x*y;
```

```
    printf("\nz=%d w=%d\n", z, w);
    getch();

    x=2; y=3;/* such assignments cause warnings */
    z=x+y; w=x*y;
        printf("\nz=%d w=%d\n", z, w);
        getchar();
        return 0;
}
```

3. Lab Tasks

3.1. Using the structure type to store a date (year, month, day), write a program to display the number of the day in a year, and the number of days to the end of that year.

3.2. Write a program which should print the week day (Monday, Tuesday, etc.) of your birth, based on your birth date and the current year week day of your anniversary.

3.3. Write a modular program which reads student related data for a group, namely: name, date of birth, residence address, and prints them in lexicographic order.

3.4. Write a program to compute the value of a polynomial of degree ***n*** with complex coefficients for a given complex value. Use functions o implement computations.

3.5. Construct a new type, **rational** type, as a structure composed of a numerator (integer) and a denominator (integer), used to represent fractions. Write functions to perform the operations: simplification, addition, subtraction, division, power for this type.

3.6. Using the union type containing the elements needed to represent circles, rectangles, squares, and triangles write a function to compute the area of the corresponding geometric figure.

3.7. Using the **enum** type, add the Boolean type. Then write a function to sort a string in ascending order using bubble sort. And a Boolean semaphore.

3.8. Read a character string composed of only letters and digits. Find the occurrence frequencies of the characters of the string using a list, ordered alphabetically (a node contains a character, the occurrence frequency and the address of the next node).

3.9. A lorry can carry at most ***m*** kilograms. The name of the materials, the amounts in kilograms, and the price per kilo are known. Find a load composition such a way the value of the load is maximum.

3.10. A **sparse polynomial** is a polynomial whose number of zero monomials is higher than the one of non-zero monomials. Write a program which stores a sparse polynomial of degree ***m*** , displays that polynomial, and finds its value for a given variable value.

3.11. A **sparse matrix** is a matrix where the number of elements which are zero is bigger than the number of elements which are not zero. Find an effective way to store sparse matrices, and write the functions to add, subtract, and multiply pairs of such matrices.

3.12. Write a program to implement the operations: insert a node, delete a node, and show the contents stored in all nodes of a **singly-linked dynamic list**. At each node, an integer value (i.e. the contents) and a pointer to the next node in the list will be stored.