

Computing Laboratory

Review of C

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

September, 2023

1 Structures

2 Preprocessor Directives

3 Header Files

4 Problems

Structures

Definition

A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling.

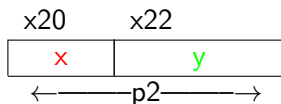
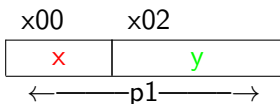
Example:

```
struct point{
    int x; // An integer field
    float y; // A floating point field
}p1, p2;
```

```
struct triangle{
    struct point a, b, c;
}t;
```

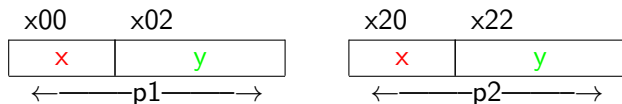
Structures

```
struct point{  
    int x; // An integer field  
    float y; // A floating point field  
}p1, p2;
```

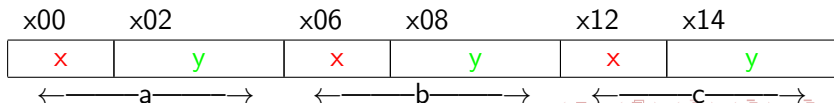


Structures

```
struct point{
    int x; // An integer field
    float y; // A floating point field
}p1, p2;
```



```
struct triangle{
    struct point a, b, c;
}t;
```



Operations on structures

- Assignment to members / fields
`p1.x = 1; p1.y = 2.0; t.a.x = 0; t.a.y = 0.5;`
- Assignment / copying of structure variables
`struct triangle t1, t2; ... ; t2 = t1;`
- Structures may be passed to functions, and returned by functions.

Note: Comparison operators (`==`, `!=`) do not work on structures.

Structure initialization

■ Initialization

```
struct point{
    int x; // An integer field
    float y; // A floating point field
}p1, p2;
```

```
struct triangle{
    struct point a, b, c;
}t = {{1, 1.0},
      {-1, 1.0},
      {1, -1.0}};
```

Typedefs

```
typedef unsigned int UI;
UI i, j;           // A pair of unsigned integers
UI myArray[10];   // An array of unsigned integers

typedef char *STR;
STR p;            // A single string
STR myStrings[128]; // An array of strings

p = (STR) malloc(100);
UI strcmp(STR, STR); // Prototype declaration
```

Typedefs

```
typedef struct {  
    int x;  
    float y;  
}POINT;  
POINT p1, p2;
```

```
typedef struct {  
    struct point a, b, c;  
}TRIANGLE;  
TRIANGLE t;
```

Pointers to structures

```
struct point *pp; // Old scheme (before typedef)
POINT *pp;       // New scheme (after typedef)

(*pp).x = 10; (*pp).y = 1.414; // Syntax 1
pp->x = 10; pp->y = 1.414;     // Syntax 2
```

Memory allocation

```
TRIANGLE *tp;
```

```
tp = (TRIANGLE *)malloc(num_triangles*sizeof(TRIANGLE));
```

Basics

Definition (What is a preprocessor?)

Before a C program is compiled by a compiler, the source code is processed by a program called preprocessor. This process is called preprocessing.

Definition (What are preprocessor directives?)

The commands used in preprocessor are defined as preprocessor directives and they begin with '#' symbol.

Predefined macros

```
#include<stdio.h>
int main(){
    printf("Current date: %s\n", __DATE__);
    printf("Current time: %s\n", __TIME__);
    printf("\# Lines in the code: %d\n", __LINE__);
    printf("File name: %s\n", __FILE__ );
    printf("C Standard: %d\n", __STDC_VERSION__);
    printf("Compilation was successful: %d\n", __STDC__);
}
```

Predefined macros

```
#include<stdio.h>
int main(){
    printf("Current date: %s\n", __DATE__);
    printf("Current time: %s\n", __TIME__);
    printf("\# Lines in the code: %d\n", __LINE__);
    printf("File name: %s\n", __FILE__ );
    printf("C Standard: %d\n", __STDC_VERSION__);
    printf("Compilation was successful: %d\n", __STDC__);
}
```

Note: The `__STDC_VERSION__` value 199409L signifies the C89 standard, 199901L signifies the C99 standard, and 201112L signifies the C11 standard.

#include

This preprocessor directive is used to include system-defined and user-defined files into the current file. There are two different ways of doing this as shown below.

```
#include <filename>
```

This is used to include system header files. It first searches for a file in a list of directories specified by you, then in a standard list of system directories.

```
#include "filename"
```

This is used include user-defined contents. It first searches for a file in the current directory, then in a standard list of system directories.

#define

There are two different types of macros as listed below.

1 Object-like macros:

```
#define e 2.71828
```

2 Function-like macros:

```
#define COMPARE(x, y) (((x) > (y)) - ((x) < (y)))
```

#define

What is the output of the following C program?

```
#include<stdio.h>
#define RECIPROCAL(x) 1/x
int main(){
    int x = 1, y = 2, z;
    z = (x+y) * RECIPROCAL(x+y);
    printf("%d", z);
    return 0;
}
```

#define (also known as macros)

Multi-line macros can be accommodated by inserting line separators at the end of each line.

```
#define ASCII(character) {\n    printf("The ASCII value of ");\n    printf("%c", character);\n    printf(" is ");\n    printf("%d", character);\n}
```

#undef

The definition of a macro can be dropped as follows.

```
#undef COMPARE(x, y)
```

#ifdef

We can check whether a macro is already defined as follows.

```
#ifdef COMPARE(x, y)
    // Code segment
#endif
```

Note: If the definition exists, it will execute the code segment.

#ifndef

We can check whether a macro is not yet defined as follows.

```
#ifndef COMPARE(x, y)
    // code segment
#endif
```

Note: If the definition does not exist, it will execute the code segment.

#if and #endif

```
#if <expression>  
    // code segment  
#endif
```

#else

```
#if <expression>
    // if code segment
#else
    // else code segment
#endif
```

#elif

```
#if expression
    // if code segment
#elif expression
    // elif code segment
#else
    // else code segment
#endif
```

#error

```
#include<stdio.h>
#ifdef __MATH_H
#error Remember to include math.h!!!
#else
void main(){
    float f = 0.7;
    printf("%.2f", pow(f, 2));
}
#endif
```

Header files

Contents:

- Pre-processor directives and macros
- Constant declarations
- Type declarations (enum, typedef, struct, union, etc.)
- Function prototype declarations
- Global variable declarations
- Static function definitions (may contain)

Header files

Contents:

- Pre-processor directives and macros
- Constant declarations
- Type declarations (enum, typedef, struct, union, etc.)
- Function prototype declarations
- Global variable declarations
- Static function definitions (may contain)

Example: HelloWorld.h

```
#ifndef _HELLOWORLD_H_
#define _HELLOWORLD_H_
typedef unsigned int my_uint_t;
void printHelloWorld();
int iMyGlobalVar;
...
#endif
```

Looking into stdio.h (GNU)

```
#ifndef _STDIO_H_
#ifdef __cplusplus
extern "C" {
#endif
#define _STDIO_H_
#define _FSTDIO /* ‘function stdio’ */
#define __need_size_t
#include <stddef.h>
#define __need__va_list
#include <stdarg.h>
struct __sFile
{
    int unused;
};
typedef struct __sFILE FILE;
```

Link: www.gnu.org/software/m68hc11/examples/stdio_8h-source.html

Looking into stdio.h (GNU)

```
#define __SLBF 0x0001    /* line buffered */
#define __SNBF 0x0002    /* unbuffered */
#define __SRD 0x0004     /* OK to read */
#define __SWR 0x0008     /* OK to write */
/* RD and WR are never simultaneously asserted */
#define __SRW 0x0010     /* open for reading & writing */
#define __SEOF 0x0020    /* found EOF */
#define __SERR 0x0040    /* found error */
#define __SMBF 0x0080    /* _buf is from malloc */
...
```

Link: www.gnu.org/software/m68hc11/examples/stdio_8h-source.html

Looking into stdio.h (GNU)

```
...
int    _EXFUN(printf, (const char *, ...));
int    _EXFUN(scanf, (const char *, ...));
int    _EXFUN(sscanf, (const char *, const char *, ...));
int    _EXFUN(vfprintf, (FILE *, const char *, __VALIST));
int    _EXFUN(vprintf, (const char *, __VALIST));
int    _EXFUN(vsprintf, (char *, const char *, __VALIST));
...
```

Link: www.gnu.org/software/m68hc11/examples/stdio_8h-source.html

Problems – Day 9

1 ...