PROGC Lab03 Modulare Programmierung und Präprozessor

Inhalt

1 Einführung

In diesem Praktikum üben Sie modulare Programmierung indem Sie ein Java Programm (bestehend aus drei Java Files) in ein entsprechendes C Programm aus drei Modulen (aus je einem Header- und Implementations-File) übersetzen.

Als zweites implementieren Sie verschiedenen Präprozessor Makros um bedingtes Tracing umzusetzen.

2 Lernziele

- Sie kennen die Grundstruktur eines Header-Files.
- Sie deklarieren im Header-File die öffentlichen Makros, Typen und Funktionen eines Moduls.
- Sie deklarieren die nicht-öffentlichen Makros, Typen und Funktionen eines Moduls im Implementations-File.
- Sie geben die nötigen Definitionen im Implementations-File an.
- Sie kennen die Grundsätze der Präprozessor Makro Definition und deren Anwendung.

3 Hintergrund Informationen

3.1 Tests

Die Tests werden zu Beginn alle brechen. Ihre Aufgabe ist es, das Praktikumsprogramm so zu implementieren dass die Tests alle den Status „passed“ haben ohne den Test-Code oder deren Stimulus und erwarteten Resultat Daten zu manipulieren.

3.2 Verwendete zusätzliche Sprach Elemente

Sprach Element	Beschreibung
`fprintf(stderr, "v=%d", v)`	Formatierte Ausgabe auf den Standard Error Stream. Siehe ***man 3 stderr*** und ***man 3 fprintf***.

`#define TRACE(MSG) \`
`  do { \`
`  fprintf(stderr, MSG); \`
`  } while(0)`
`...`
`TRACE("Calling f(abc)");`
	Dieses Macro zeigt idiomatisches C: was soll ein Macro erfüllen:
	- Ein Macro muss auf einer Zeile angegeben werden Line-Continuation falls die Zeile im Source Code zu lange wird.
	- Ein Funktions-Macro soll wie eine Funktion aufgerufen werden können, d.h. mit einem Strich-Punkt am Ende. Oft wird das mittels eines ***do { ... } while(0)*** umgesetzt (ohne terminierenden Strich-Punkt!).

`#define TRACE(MSG) \`
`  do { \`
`  fprintf(stderr, \`
`  "Trace: " MSG); \`
`  } while(0)`
`...`
`TRACE("done"); // --> Trace: done`
	Obiges Macro erweitert:
	C erlaubt es, String Literale (***"..."***) hintereinander zu schreiben. Die String Literale werden zu einem einzigen zusammengefügt.

`#define TRACE(MSG, ...) \`
`  do { \`
`  fprintf(stderr, \`
`  "Trace: " MSG, \`
`  ##__VA_ARGS__); \`
`  } while(0)`
`...`
`TRACE("%d", v);// --> Trace: 123`
	Obiges Macro nochmals erweitert:
	Variadic Macros: Macros können mit variabler Anzahl Argumente definiert werden.
	In der Definition ***...*** angeben und verwenden mit ***##__VA_ARGS__***.

4 Aufgaben

4.1 Modulares Programm

Ergänzen Sie in **`lab03-modular/src`** den Code in **`triangle.c`**, **`read.h`**, **`read.c`**, **`rectang.h`** und **`rectang.c`** so dass die Tests erfolgreich durchlaufen. Die C Implementation soll dieselbe Funktionalität haben wie die unten gegebenen Java Files. Lehnen Sie sich so nahe wie möglich an die Java Files an.

1) In den Header-Files implementieren Sie den Include-Guard und deklarieren Sie die öffentlichen Funktionen und gegebenenfalls **`#define`**s.
2) In den Implementations-Files implementieren Sie die Funktionen.

Die drei Java Files liegen in **`lab03-modular/java`**

Tipps:

- Implementieren Sie die alles gross geschriebenen Symbole als **`#define`**.
- **`EOF`** kommt schon aus **`stdio.h`** und sollte deshalb nicht mehr definiert werden.

- Jene **`#define`**s welche von andern Modulen verwendet werden kommen ins Header-File, die andern ins Implementations-File.
- Ein Grossteil des Java Codes aus den Methoden Bodies kann eins-zu-eins in C übernommen werden.
  Listen Sie auf welche Unterschiede es gib. Z.B.

Java	C
`byte`	`char`
`boolean`	`bool`
`true`	`true`
`false`	`false`
`System.out.print(…)`	`printf(…)`
`System.out.println(…)`	`printf(...\n)`
`System.in.read()`	`scanf(…)`
`byte[] buffer = new byte[BUFFERSIZE];`	`char[] buffer = {BUFFERSIZE}`

`public class rectang {`
`  public boolean Rectangular(…)`
`  { … }`
`}`
	`class rectang {`
	`  Char rectangular(…)`
	`  {…}`
	`}`

`public class read {`
`  public int getInt(…)`
`  throws java.io.IOException`
`  { … }`
`}`
	`class read {`
	`  int getInt() //Es gibt keine Exceptions in`
	`  {…}  //in C`

`class triangle {`
`  public static void main(String[] args)`
`  throws java.io.IOException`
`  { … }`
`}`
	`class triangle {…}`
	`int main(void){ //Keine Exceptions`
	`{…}`

`read ReadInt = new read();`
`…`
`word = ReadInt.getInt(MAX_NUMBER);`
	`struct read r;`
	`…`
	`int word = getInt(MAX_NUMBER);`

`rectang Rect = new rectang();`
`…`
`if (Rect.Rectangular(a, b, c) == true) { … }`
	`int rectang(int, int, int){…}`
	`…`
	`if (rectang(a, b, c) == true) {…}`

`System.out.println(`
`  "-> Dreieck " + a + "-" + b + "-" + c`
`  + " ist rechtwinklig");`
	`printf(“-> Dreieck %d - %d - %d`
	`ist rechtwinklig\n”,a,b,c);`

triangle.java triangle.c

```
/*****************************************************************************
    Funktion:       Die drei Seiten eines Dreiecks einlesen und bestimmen ob
                    das Dreieck rechtwinklig ist.
    Returns:        Nichts.
*****************************************************************************/

class triangle {

    public static void main(String[] args)
        throws java.io.IOException
    {
        int READ_ERROR = -2;
        int MAX_NUMBER = 1000;

        read ReadInt = new read();
        rectang Rect = new rectang();

        while (true) {
            System.out.println("\nDreiecksbestimmung (CTRL-C: Abbruch)\n");

            int word = 0;
            int a = 0;
            int b = 0;
            int c = 0;

            do {
                System.out.print("Seite a: ");
                word = ReadInt.getInt(MAX_NUMBER);
            }
            while ((word < 0) && (word != READ_ERROR));
            if (word >= 0)
                a = word;
            else
                break;

            do {
                System.out.print("Seite b: ");
                word = ReadInt.getInt(MAX_NUMBER);
            }
            while ((word < 0) && (word != READ_ERROR));
            if (word >= 0)
                b = word;
            else
                break;

            do {
                System.out.print("Seite c: ");
                word = ReadInt.getInt(MAX_NUMBER);
            }
            while ((word < 0) && (word != READ_ERROR));
            if (word >= 0)
                c = word;
            else
                break;

            if (Rect.Rectangular(a, b, c) == true)
                System.out.println("-> Dreieck " + a + "-" + b + "-" + c
                                   + " ist rechtwinklig");
            else
                System.out.println("-> Dreieck " + a + "-" + b + "-" + c
                                   + " ist nicht rechtwinklig");
            System.out.println("\n");
        }
        System.out.println("\n\nbye bye\n");
    }
}
```

read.java read.h/read.c

```
/*****************************************************************************
    Funktion:       Unsigned int Zahl via Bytestream von stdin einlesen.
                    Es wird zuerst eine Zeile eingelesen und dann konvertiert.
                    Die eingelesene Zeile darf nur eine Zahl enthalten
                    (mit optionalen Leerzeichen vor/nach der Zahl).
    Returns:        Die konvertierte Zahl
                    oder -1 (PARSE_ERROR) wenn keine Zahl oder zu gross
                    oder -2 (READ_ERROR) wenn Fehler beim einlesen.
*****************************************************************************/

public class read {
    public int getInt(int maxResult)
        throws java.io.IOException
    {
        // end of input
        int EOF   = -1; // end of file
        int EOL   = 10; // end of line
        // abnormal return values
        int PARSE_ERROR = -1;
        int READ_ERROR  = -2;
        // ASCII Codes
        int ASCII_SPACE   = 32; // ' '
        int ASCII_DIGIT_0 = 48; // '0'
        int ASCII_DIGIT_9 = 57; // '9'

        // conversion buffer
        int NO_POS = -1;
        int BUFFERSIZE = 10;
        byte[] buffer = new byte[BUFFERSIZE];

        int result = 0;

        // read line: up to EOL or EOF (i.e. error while reading)
        int bytes = 0;
        int input = System.in.read();
        while ((input != EOL) && (input != EOF)) { // read whole line
            if (bytes < BUFFERSIZE) { // only buffer first n characters
                buffer[bytes] = (byte)input;
                bytes++;
            } else {
                result = PARSE_ERROR; // exceed buffer size, continue read line
            }
            input = System.in.read();
        }
        if (input == EOF) {
            result = READ_ERROR;
        }
        // check for numbers: skip leading and trailing spaces
        // (i.e. this includes all control chars below the space ASCII code)
        int pos = 0;
        while((pos < bytes) && (buffer[pos] <= ASCII_SPACE)) pos++; // skip SP
        int posOfFirstDigit = pos;
        int posOfLastDigit = NO_POS;
        while ((pos < bytes)
               && (buffer[pos] >= ASCII_DIGIT_0)
               && (buffer[pos] <= ASCII_DIGIT_9))
        {
            posOfLastDigit = pos;
            pos++;
        }
        while((pos < bytes) && (buffer[pos] <= ASCII_SPACE)) pos++; // skip SP
        // produce return value
        if (result != 0) {
            // previously detected read or parse error given
        } else if ((pos != bytes) || (posOfLastDigit == NO_POS)) {
            result = PARSE_ERROR;
        } else { // convert number
            for(int i = posOfFirstDigit; i <= posOfLastDigit; i++) {
                result = result * 10 + (buffer[i] - ASCII_DIGIT_0);
                if (result > maxResult) {
                    result = PARSE_ERROR;
                    break;
                }
            }
        }
        return result;
    }
}
```

rectang.java rectang.h/rectang.c

```
/*************************************************************************
    Funktion:       Bestimmt, ob Dreieck rechtwinklig ist.
    Returns:        true wenn rechtwinklig, sonst false.
*************************************************************************/

public class rectang {

    public boolean Rectangular(int a, int b, int c) {

        int aS = a*a;
        int bS = b*b;
        int cS = c*c;

        boolean isRightAngled;
        if ((a == 0) && (b == 0) && (c == 0))
            isRightAngled = false;
        else if ((aS + bS) == cS)
            isRightAngled = true;
        else if ((aS + cS) == bS)
            isRightAngled = true;
        else if ((bS + cS) == aS)
            isRightAngled = true;
        else
            isRightAngled = false;

        return isRightAngled;
    }
}
```

4.2 Präprozessor Makros

Ergänzen Sie das Header-File **`trace.h`** mit einem variadic Trace Macro (siehe 3.2).

Vergessen Sie den Include Guard nicht.

Rufen Sie dieses **TRACE** Macro zu Beginn jeder Funktionsimplementation auf um den Funktionsnamen und die Parameter auszugeben.