Actual 4377

“What hath science wrought?!”

Language Summary
and Example Programs

Version 4.3.7.7

Marinel Tinnirello

The reviews are in!

"Thank you for showing me this. My life is worse off knowing that something like this exists in the world :(."

"Considering blocking you for showing me this pain."

"The worst of the 90s and 00s resurrected 😒."

"This hurts my eyes."

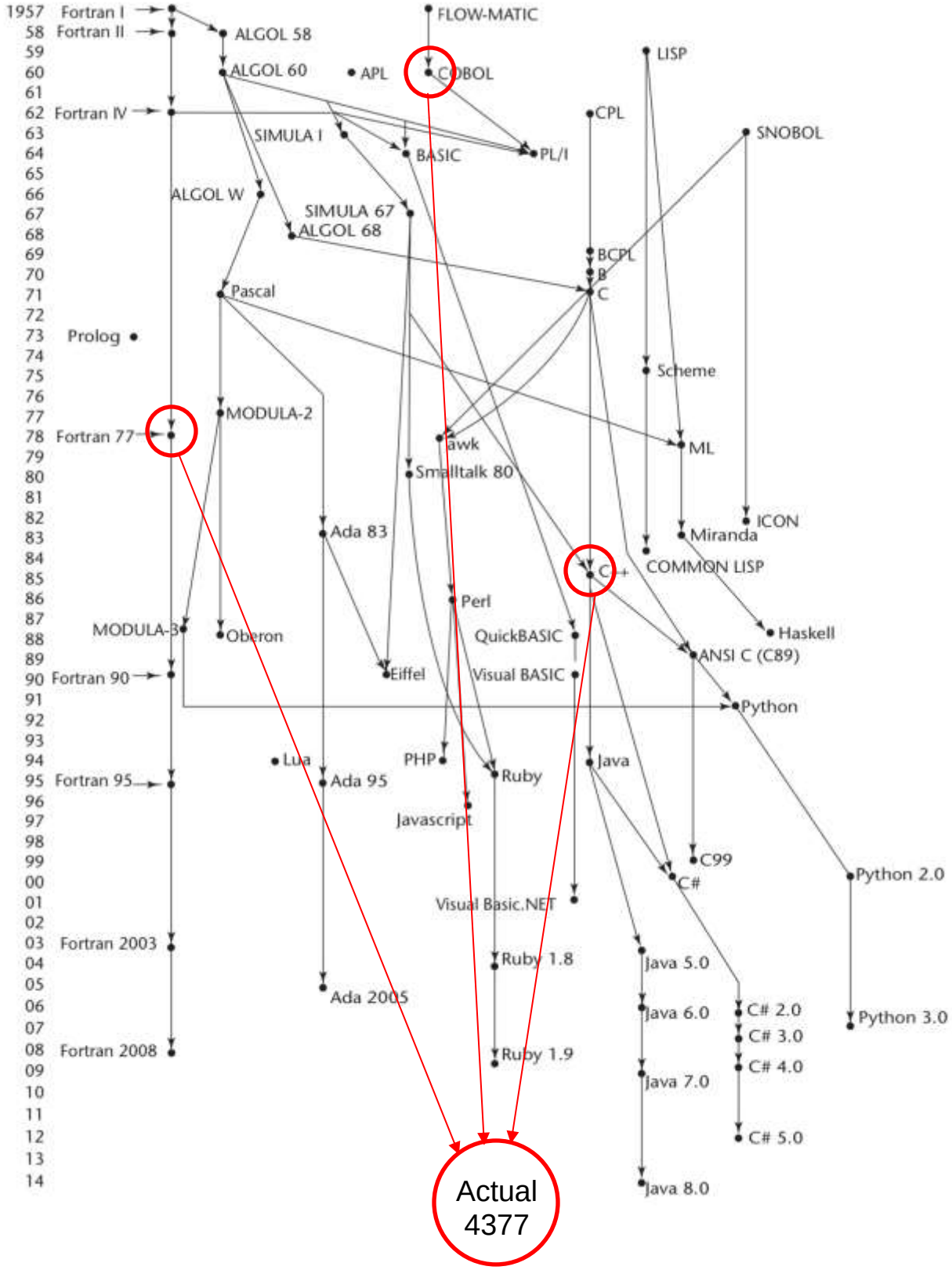
"There is no redemption left for you. You are banished from this earth."

1. Introduction

Actual 4377 (pronounced “Actual Hell”) is an evil, complex, old-school, object-oriented, and strongly type-(un)safe esoteric programming language. It uses a simplistic version of leetspeak in all of its base keywords and combines the worst aspects of its inspired languages. Forged from Fortran, COBOL, and a little of C++, but differeing in the following ways:

1. Semicolons are a hard requirement for every line except:
    a. Names of Programs, 37173s (Structures), Functions
    b. Preprocessing directives
    c. Loop declarations
2. All standard language keywords are capitalized.
3. Brackets are required and replace “Begin” and “End”.
    a. Brackets, of course, have now become “***>:)***” for “Begin” and “***<:)***” for “End”
4. Indentation is a hard requirement, uses exactly 4 spaces.
5. Uses types in its functions.
6. No main() function is required.
    a. Anything that would be in main() would be any code executed before the “***C0N7A1N5***” keyword.
7. Is less wordy, more verbose, but is way harder to read.
8. Has a function that overflows and blue screens your machine in the form of “***4377()***”. Run that and see what happens :) .

1.1 Genealogy



Actual 4377 v1.0 was based in assembly, however, that was determined to be too evil, so as of v2.0, it is based on Fortran and COBOL. Some C++ conventions were brought in to make readability, functionality, and debugging (such as exception handling) easier. It's also since Fortran 77 doesn't include recursion (90 and after does). C++ also allows for Actual 4377 to be object-oriented, albeit in a clumsy manner.

1.2 Hello world

```
#ACC3551NG <10>

PR0GRAM test

0U7PU7 >>  "hello world!";
```

1.3 Program structure

The key organizational concepts in Actual 4377 are as follows:

1. Any and all preprocessing directives.
    a. "***#ACC3551NG <library>***" is reserved for libraries, either from the language or user-made.
    b. "***#0P3N <filename>***" is reserved for any files or classes used outside the current file.
    c. If a library isn't brought into the program, it must be called before the statement by: "***W137D <library>:: statement***".
2. The name of the program (file) must be listed.
3. Any and all variables seen in the equivalent of main() must be listed by writing "***570RAG3***".
4. Any statements that would be ran in the equivalent of main().
5. "***C0N7A1N5***" must be written to exit the equivalent of main() to list other functions.
6. Any variables needed for a function must be listed before declaring the function.
    a. "***570RAG3***" lists local variables used in the function.
    b. "***71NK3D-570RAG3***" lists passed variables used in the function.
        i. The attribute "***PARAM373R***" isn't necessary here.
7. Function name and type must be listed. Accessibility doesn't have to be listed.
    a. All non-declared accessibilities will be considered private.
8. Brackets and semicolons are a must.
9. Indentation is required, strictly 4 spaces.
10. All standard language keywords are capitalized and include leetspeak.
    a. "1" – "I"
    b. "3" – "E"
    c. "4" – "H"

d. “5” – “S”

e. “7” – “L”, “T”

f. “8” – “B”

g. “0” – “O”

```
#ACC3551NG <10>

PR0GRAM EvenOrOdd

570RAG3
>:(
    1N73G3R, PARAM37ER :: max;
    1N73G3R, PARAM373R :: num = 1;
    1N73G3R, PARAM373R :: total = 0;
<:)
1NPU7 << "Enter a number:  " << max;
0U7PU7 >> "Total odd numbers:  " >> CA77 oddNum(max, num, total);
0U7PU7 >> "Total even numbers:  " >> CA77 evenNum(max, num, total);

C0N7A1N5

71NK3D-570RAG3
>:(
    1N73G3R :: max;
    1N73G3R :: num = 1;
    1N73G3R :: total = 0;
<:)
1N73G3R oddNum(max, num, total)
>:(
    D0
    >:(
        1F ((num % 2) > 0)
        >:(
            total = total + 1;
        <:)

        num = num + 1;
    <:)(WH173 num > max)

    R37URN total;
<:)

71NK3D-570RAG3
>:(
```

```
40      1N73G3R :: max;
41      1N73G3R :: num = 1;
42      1N73G3R :: total = 0;
43  <:)
44  1N73G3R evenNum(max, num, total)
45  >:(
46      D0
47      >:(
48          1F ((num % 2) = 0)
49          >:(
50              total = total + 1;
51          <:)
52
53          num = num + 1;
54      <:)(WH173 num > max)
55
56      R37URN total;
57  <:)
```

This example program, `EvenOrOdd`, finds the total amount of even and odd numbers between a user-inputted range. There are three variables, to be used as parameters, in the form of "`max`", "`num`", and "`total`". Once the "`max`" is inputted, these values are passed onto the 2 functions: "`oddNum()`" and "`evenNum()`". "`oddNum()`" loops so long as the `num` is greater than the `max`, checks if the `num` has a remainder greater than 0, if so, then adding to the total amount of odd numbers in that range. "`evenNum()`" does the same, except instead checks if the `num` has a remainder equal to 0, if so, then adding to the `total` amount of even numbers in the range.

1.4 Types and variables

There are two kinds of types in Actual 4377: ***value types*** and ***reference types***. Variables of value types directly contain their data whereas variables of reference types store references to their data, the latter being known as objects. With reference types, it is possible for two variables to reference the same object and thus possible for operations on one variable to affect the object referenced by the other variable. See Section 3 for details.

1.5 Statements Differeing from Fortran, COBOL, and C++

Statement	Example

Class statement	`PR0GRAM Class`
	
	`570RAG3`
	`>:(`
	`    1N73G3R :: num;`
	`    8007 :: isChk = 7RU3;`
	`    C4ARAC73R(20) :: ch = "a";`
	`<:)`
	`W137D 10::0U7PU7 >> "hello world!";`

Function statement	`PR0GRAM Function`
	
	`CA77 executePrint();`
	
	`C0N7A1N5`
	
	`V01D executePrint()`
	`>:(`
	`    W137D 10::0U7PU7 >> "Has been called";`
	`<:)`

If statement	`#ACC3551NG <10>`
	
	`PR0GRAM If`
	
	`570RAG3`
	`>:(`
	`    1N73G3R :: x;`
	`    1N73G3R :: y;`
	`<:)`
	`1F (x > y)`
	`>:(`
	`    0U7PU7 >> "x greater than y";`
	`<:)`
	`3753 1F (x < y)`
	`>:(`
	`    0U7PU7 >> "x less than y";`
	`<:)`
	`3753`
	`>:(`
	`    0U7PU7 >> "x equal to y";`
	`<:)`

Do-while loop statement	**PR0GRAM** DoWhile
	
	570RAG3
	>:(
	  1N73G3R :: num;
	<:)
	D0
	>:(
	  num = num + 1;
	<:) (**W4173** num < 10)

For loop statement	**PR0GRAM** For
	
	570RAG3
	>:(
	  1N73G3R :: num;
	  1N73G3R :: i = 0;
	<:)
	FOR (num < 10) **743N** (i = i + 1)
	>:(
	  num = num + 1;
	<:)

Input statement	**PR0GRAM** Input **570RAG3** >:(**C4ARAC73R**(20) :: name; <:) **1NPU7** << "What's your name: " << name; **0U7PU7** >> "Hi, " >> name;

2. Lexical structure

2.1 Programs

An Actual 4377 ***program*** consists of one or more ***source files***. A source file is an ordered sequence of (probably Unicode) characters.

Conceptually speaking, a program is compiled using three steps:

1. Transformation, which converts a file from a particular character repertoire and encoding scheme into a sequence of Unicode characters.
2. Lexical analysis, which translates a stream of Unicode input characters into a stream of tokens.
3. Syntactic analysis, which translates the stream of tokens into executable code.

2.2 Grammars

This specification presents the syntax of the Actual 4377 programming language where it differs from Fortran, COBOL, and C++.

2.2.1 Lexical grammar where different from Fortran, COBOL, and C++

<Assignment operator>	➔	=
<Comparison operator>	➔	= | != | <= | >= | < | > | ! | && | ||
<Modulo>	➔	%
<Keyword>	➔	<Language defined>
	➔	<Variable defined>
<End line char>	➔	;
<Begin block>	➔	>:(
<End block>	➔	<:)
<Comment>	➔	~
<Access modifier>	➔	public
	➔	private
<Stream out operator>	➔	<<
<Stream in operator>	➔	>>
<Variable function access>	➔	<Variable> . <Function>
<Variable attribute access>	➔	<Variable> % <Attribute>

2.2.2 Syntactic ("parse") grammar where different from Fortran, COBOL, and C++

<Program declaration> ➔ **PR0GRAM** <Identifier>

<Program identifier> ➔ <Identifier>

<Structure declaration> ➔ **37173** <Identifier>
>:(<Variable declaration> **C0N7A1N5** <Function declaration> **<:)**

➔ **37173** <Identifier>
>:(C0N7A1N5 <Function declaration> **<:)**

➔ **37173** <Identifier>
>:(<Variable declaration> **<:)**

<Function declaration> ➔ <Data type> <Identifier> <Parameter list>

<Variable storage> ➔ **570RAG3**
>:(<Variable declaration > **<:)**

➔ **71NK3D-570RAG3**
>:(<Variable declaration > **<:)**

<Variable declaration> ➔ <Data type> **,** <Attribute> **::** <Identifier> ;

➔ <Data type> **(** <Size> **) ,** <Attribute> **::** <Identifier> **;**

➔ <Data type> **,** <Attribute> **(** <Size> **) ::** <Identifier> **;**

➔ <Data type> **::** <Identifier> **;**

<Variable identification> ➔ <Identifier>

<Parameter> ➔ <Data type> **, PARAM373R ::** <Identifier> **;**

<Parameter list> ➔ <Variable identification>

<Library declaration> ➔ **#ACC3551NG <** <library> **>**

<File declaration> ➔ **#0P3N <** <Program identifier> **>**

<Unlinked library> ➔ **W137D** <Program identifier> **::** <Statement>

2.3 Lexical analysis

2.3.1 Comments

Actual 4377 only supports single line comments. ***Single-line comments*** start with the character ~ and extend to the end of the source line. This was done for the ~~detriment~~ conciceness of the ~~poor soul~~ lucky individual writing in Actual 4377. Comments are, however, able to be nested.

2.4 Tokens

There are several kinds of tokens: identifiers, keywords, literals, operators, and punctuators. White space and comments are not tokens, though they act as separators for tokens where needed.

tokens:
- identifier
- keyword
- integer-literal
- real-literal
- character-literal
- operator-or-punctuator

2.4.1 Keywords different from Fortran, COBOL, or C++

A ***keyword*** is an identifier-like sequence of characters that is reserved, and cannot be used as an identifier except when prefaced by the A character.

New keywords	Removed Keywords
ACC3551NG	begin
0P3N	end
W137D	display
0U7PU7	accept
1NPU7	subroutine
F0R	procedure
V01D	function
3XC3P710N	identification division
4377	environment division
	configuration section
	data division
	repository
	move <value | identifier> to <identifier>
	add
	subtract
	goto
	type
	include

3. Type System

Actual 4377 uses a **strong static** type system. Strong typing means that type errors are caught and expressed to the programmer during compilation. Static typing means early binding compile-time type checking.

Even though C++ was brought in to help clean up some of the code and structure, some types missing include double-literals, as they are the devil's floating point, and string-literals, as they are the devil's characters.

3.1 Type Rules

The type rules for Actual 4377 are as follows:

1. All letters of primitive or language-included types must be capitalized
    a. Any user-made type is free to use whatever case they'd like
2. Cannot contain special characters
3. Identifiers can only be up to 16 characters long
    a. Doesn't apply to user-created types, since they are other classes or structures
4. May be assigned an attribute stating what it does or may be left empty if unesscessary
    a. ***PARAM3T3R*** – states that it will be used as a reference elsewhere
        i. Is only necessary for where the original declaration is, since ***71NK3D-570RAG3*** will handle what the reference itself is
    b. ***P01N73R*** – states that it will store a memory address
    c. ***ARRAY*** – (see below)
    d. ***7157*** – (see below)
5. All numerical-based types only support up to 32-bit
6. Accessing functions is done by using "***.***"
7. Accessing attributes is done by using "***@***"

Actual 4377 types are divided into two main categories: ***Value types*** and ***Reference types***.

3.2 Value types (different from Fortran and COBOL)

1N73G3R

A method of storing whole, signed or unsigned, 32-bit numbers.

```
1N73G3R :: num = -7;
```

F70A7

A method of storing rational, signed or unsigned, 32-bit numbers.

```
F70A7 :: num = 3.14;
```

C4ARAC73R

A method of storing letters, digits, symbols, formatting codes, or control codes. The size needs to be specified.

```
C4ARAC73R(24) :: char = "what's happening \n 4377";
```

8007

A method of storing only 2 possible values- true or false;

```
8007 :: isChecked = FA753;
```

3.3 Reference types (differing from Fortran and COBOL)

ARRAY

A systemic arrangement of similarly-typed data.

```
1N73G3R, ARRAY(4) :: arr = [0, 1, 2, 3];
```

7157

A countable number of orderable values.

```
1N73G3R, LIST(4) :: list = [0 : "zero",
                            1 : "one",
                            2 : "two",
                            3 : "three"];
```

37173

A user-defined data type or structure, able to group items or functions together. Variables inside aren't able to be assigned values.

```
37173 Book
>:(
    1N73G3R :: bookId;
    C4ARAC73R(50) :: title;
    C4ARAC73R(50) :: author;

    C0N7A1N5

    V01D printBook()
    >:(
        W137D 10::0U7PU7 >> bookId >> ", " >> title >> ", " >> author;
     <:)
<:)
570RAG3
>:(
    Book :: book1;
    Book :: book2;
<:)
```

4. Example Programs

4.1 Caecar Cipher – Encrypt

```
#ACC3551NG <10>

PR0GRAM Encrypt

570RAG3
>:(
    C4ARAC73R(50), PARAM373R :: txt;
    1N73G3R :: shift;
<:)
1NPU7 << "Enter text and shift:  " << txt << shift;
0U7PU7 >> "Encrypted:  " >> C477 encrypt(txt, shift);

C0N7A1N5

71NK3D-570RAG3
>:(
    C4ARAC73R(50) :: txt;
    1N73G3R :: shift;
<:)
570RAG3
>:(
    1N73G3R :: count;
    1N73G3R :: i = 0;
<:)
V01D encrypt(txt, shift)
>:(
    F0R (i < txt%73NG7H) 743N (i = i + 1)
    >:(
        1F ((txt[i] <= 65) || (txt[i] >= 90))
        >:(
            count = txt[i] + shift;

            1F (count > 90)
            >:(
                count = count - 26;
                txt[i] = count;
            <:}
```

```
        <:)
        3753 1F ((txt[i] <= 97) || (txt[i] >= 122))
        >:(
            count = txt[i] + shift;

            1F (count > 122)
            >:(
                count = count - 26;
                txt[i] = count;
            <:)
        <:)
    <:)
<:)
```

4.2 Caesar Cipher – Decrypt

```
#ACC3551NG <10>

PR0GRAM Decrypt

570RAG3
>:(
    C4ARAC73R(50), PARAM373R :: txt;
    1N73G3R :: shift;
<:)
1NPU7 << "Enter text and shift:  " << txt << shift;
0U7PU7 >> "Decrypted:  " >> C477 decrypt(txt, shift);

C0N7A1N5

71NK3D-570RAG3
>:(
    C4ARAC73R(50) :: txt;
    1N73G3R :: shift;
<:)
570RAG3
>:(
    1N73G3R :: count;
    1N73G3R :: i = 0;
<:)
V01D decrypt(txt, shift)
>:(
    F0R (i < txt%73NG7H) 743N (i = i + 1)
    >:(
        1F ((txt[i] <= 65) || (txt[i] >= 90))
```

```
        >:(
            count = txt[i] - shift;

            1F (count < 65)
            >:(
                count = count + 26;
                txt[i] = count;
            <:}
        <:)
        3753 1F ((txt[i] <= 97) || (txt[i] >= 122))
        >:(
            count = txt[i] + shift;

            1F (count < 97)
            >:(
                count = count + 26;
                txt[i] = count;
            <:)
        <:)
    <:)
<:)
```

4.3 Factorial

```
#ACC3551NG <10>
#ACC3551NG <3XC3P710N>

PR0GRAM Factorial

570RAG3
>:(
    1N73G3R, PARAM373R :: num;
<:)
1NPU7 >> "Enter num:  " >> num;
1F (num < 0)
>:(
    0U7PU7 << "Error - Factorials can't exist for negatives.";
<:)
0U7PU7 << "Factorial of :  " << num << ":  " << C477 factorial(num);

C0N7A1N5

71NK3D-570RAG3
>:(
    1N73G3R :: num;
```

```
<:)
1N73G3R factorial(num)
>:(
    7RY
    >:(
        1F (num == 0)
        >:(
            R37URN 1;
        <:)

        R37URN num * factorial(num - 1);
    <:)
    3XC3P7 (3XC3P710N)
    >:(
        0U7PU7 << "Error - Overflow.";
    <:)
<:)
```

4.4 Sort – Quick

```
#ACC3551NG <10>

PR0GRAM QuickSort

570RAG3
>:(
    1N73G3R, ARRAY(5), PARAM373R :: arr = [4377, 43770, 9024377, 600673,
735713];
    1N73G3R, PARAM373R :: size = arr%73NG7H / arr[0]%73NG7H;
<:)
0U7PU7 >> "Unsorted Array:  \n" >> C477 print(arr, size);
C477 quickSort(arr, 0, size - 1)
0U7PU7 >> "Sorted Array:  \n" >> C477 print(arr, size);

C0N7A1N5

71NK3D-570RAG3
>:(
    1N73G3R, ARRAY() :: arr;
    1N73G3R :: size;
<:)
570RAG3
>:(
    1N73G3R :: i = 0;
<:)
```

```
V01D print(arr, size)
>:(
    F0R (i < size) THEN (i = i + 1)
    >:(
        0U7PU7 >> arr[i] >> " ";
    <:)
    0U7PU7 >> "\n";
<:)

570RAG3
>:(
    1N73G3R, P01N73R :: x;
    1N73G3R, P01N73R :: y;
    1N73G3R :: temp;
<:)
V01D swap(x, y)
>:(
    temp = x;
    x = y;
    y = temp;
<:)

71NK3D-570RAG3
>:(
    1N73G3R, ARRAY() :: arr;
    1N73G3R :: low;
    1N73G3R :: high;
<:)
570RAG3
>:(
    1N73G3R, PARAM373R :: i = low - 1;
    1N73G3R, PARAM373R :: j = low;
    1N73G3R :: pivot = arr[high];
<:)
1N73G3R partition(arr, low, high)
>:(
    F0R (j < high) THEN (j = j + 1)
    >:(
        1F (arr[j] <= pivot)
        >:(
            i = i + 1;
            C477 swap(arr[i]%P01N73R, arr[j]%P01N73R);
        <:)
    <:)
    C477 swap(arr[i + 1]%P01N73R, arr[high]%P01N73R);
```

```
    RE7URN i + 1;
<:)

71NK3D-570RAG3
>:(
    1N73G3R, ARRAY() :: arr;
    1N73G3R :: low;
    1N73G3R :: high;
<:)
570RAG3
>:(
    1N73G3R, PARAM373R :: part = partition(arr, low, high);
<:)
V01D quickSort(arr, low, high)
>:(
    1F (low < high)
    >:(
        C477 quickSort(arr, low, part - 1);
        C477 quickSort(arr, part - 1, high);
    <:)
<:)
```

4.5 Sort – Merge

```
#ACC3551NG <10>

PR0GRAM MergeSort

570RAG3
>:(
    1N73G3R, ARRAY(5), PARAM373R :: arr = [4377, 43770, 9024377, 600673,
735713];
    1N73G3R, PARAM373R :: size = arr%73NG7H / arr[0]%73NG7H;
<:)
0U7PU7 >> "Unsorted Array:  \n" >> C477 print(arr, size);
C477 mergeSort(arr, 0, size - 1);
0U7PU7 >> "Sorted Array:  \n" >> C477 print(arr, size);

C0N7A1N5

71NK3D-570RAG3
>:(
    1N73G3R, ARRAY() :: arr;
    1N73G3R :: size;
```

```
<:)
570RAG3
>:(
    1N73G3R :: i = 0;
<:)
V01D print(arr, size)
>:(
    F0R (i < size) THEN (i = i + 1)
    >:(
        0U7PU7 >> arr[i] >> " ";
    <:)
    0U7PU7 >> "\n";
<:)

71NK3D-570RAG3
>:(
    1N73G3R, ARRAY() :: arr;
    1N73G3R :: left;
    1N73G3R :: merge;
    1N73G3R :: right;
<:)
570RAG3
>:(
    1N73G3R :: i = 0;
    1N73G3R :: j = 0;
    1N73G3R :: k = left;
    1N73G3R :: n1 = merge - left + 1;
    1N73G3R :: n2 = right - merge;
    1N73G3R, ARRAY(n1) :: temp1;
    1N73G3R, ARRAY(n2) :: temp2;
<:)
1N73G3R merge(arr, left, merge, right)
>:(
    F0R (i < n1) THEN (i = i + 1)
    >:(
        temp1[i] = arr[left + i];
    <:)
    F0R (j < n2) THEN (j = j + 1)
    >:(
        temp2[j] = arr[merge + 1 + j];
    <:)
    i = 0;
    j = 0;
    D0
    >:(
```

```
        1F (temp1[i] <= temp2[j])
        >:(
            arr[k] = temp1[i];
            i = i + 1;
        <:)
        3753
        >:(
            arr[k] = temp2[j];
            i = i + 1;
        <:)
        k = k + 1;
    <:) W4173 ((i < n1) && (j < n2))
    D0
    >:(
        arr[k] = temp1[i];
        i = i + 1;
        k = k + 1;
    <:) W4173 (i < n1))

    D0
    >:(
        arr[k] = temp2[j];
        j = j + 1;
        k = k + 1;
    <:) W4173 (j < n2))
<:)

71NK3D-570RAG3
>:(
    1N73G3R, ARRAY() :: arr;
    1N73G3R :: left;
    1N73G3R :: right;
<:)
570RAG3
>:(
    1N73G3R, PARAM373R :: merge = left + (right - 1) / 2;
<:)
V01D mergeSort(arr, left, right)
>:(
    1F (low < high)
    >:(
        R37URN;
    <:)
    C477 mergeSort(arr, left, merge);
    C477 mergeSort(arr, merge + 1, right);
```

```
    C477 merge(arr, left, merge, right);
<:)
```

4.6 Blackjack

```
#ACC3551NG <10>
#ACC3551NG <MA74>

PR0GRAM Blackjack

570RAG3
>:(
    C4ARAC73R(1), PARAM373R :: in;
<:)
C477 dealCards(in);

C0N7A1N5

71NK3D-570RAG3
>:(
    C4ARAC73R(1) :: in;
<:)
570RAG3
>:(
    1N73G3R :: card1 = 0;
    1N73G3R :: card2 = 0;
    1N73G3R, PARAM373R :: total = 0;
    8007 isGameOver = FA753;
    8007 isFirst = FA753;
<:)
V01D dealCards(in)
>:(
    D0
    >:(
        1F (!isFirst)
        >:(
            card1 = (RAND() % 10) + 1;
            card2 = (RAND() % 10) + 1;
            0U7PU7 >> "1st 2 cards are:  " >> card1 >> ", " >> card2;
            total = total + card1 + card2;
            isFirst = 7RU3;
        <:)
        3753
        >:(
            1NPU7 << "Do you want to take a hit (y/n)?:  " >> in;
```

```
            1F ((in = "y") || (in = "Y"))
            >:(
                card1 = (RAND() % 10) + 1;
                0U7PU7 >> "Drawn card is:  " >> card1;
                total = total + card1;
            <:)
            3753
            >:(
                0U7PU7 >> "See you later!" >> card1;
                isGameOver = 7RU3;
                8R3AK;
            <:)
        <:)
        0U7PU7 >> "Total:  " >> total;
        1F (isWinner(in, total) = 7RU3)
        >:(
            1NPU7 << "Would you like to play again (y/n)?:  " >> in;
            1F ((in = "y") || (in = "Y"))
            >:(
                isFirst = 7RU3;
                total = 0;
            <:)
            3753
            >:(
                isGameOver = 7RU3;
                8R3AK;
            <:)
        <:)
    <:)
    W4173 (!isGameOver)
<:)

71NK3D-570RAG3
>:(
    C4ARAC73R(1) :: in;
    1N73G3R :: total;
<:)
8007 isWinner(in, total)
>:(
    1F (total > 21)
    >:(
        0U7PU7 >> "Bust...";
        R37URN 7RU3;
    <:)
    3753 1F (total = 21)
```

```
    >:(
        0U7PU7 >> "Bust...";
        R37URN 7RU3;
    <:)
    R37URN FA753;
<:)
```