Set Lang (Compiles to C or C++)

Types:

int – like in C

str – String variable

set – Defines a set of (**unique**) int values

collection – Defines a set of (**unique**) str strings

Literals:

dddd	- integer value
" …. "	- String literal ("" – Empty string)
[dddd, dddd, …]	-- Set literal ([] – Empty set)
{ "….", "….", … }	-- Collection literal ({} – Empty collection)

Variable Definitions:

int	<var> [, <var> …]	-- Define one or more int variables
str	<var> [, <var> …]	-- Define one or more str variables
set	<var> [, <var> …]	-- Define one or more set variables
collection	<var> [, <var> …]	-- Define one or more collection variables

Operations:

+, - , *, /	– for int, as in C
+, -, &, |…|	– for set and collection
()	– parenthesis, as in C
=	– Assignment to variable

Set/Collection Operation definitions

Operation	Action	Result type
<set> + <set> <collection> + <collection>	Union	set collection
<set> - <set> <collection> - <collection>	Difference	Set collection
<set> & <set> <collection> & <collection>	Intersection	set collection
|<set>| |<collection>|	Size of (# of elements)	int
<set> + int	Insert	<set>

<set> - int	Extract	<set>
<collection> + "..."	Insert	<collection>
<collection> - "..."	Extract	<collection>

Conditions:

>, <, >=, <=, ==	-- for int: numeric compare; for string lexicographic compare
==	-- for set and collection
<set>	
<collection>	-- Empty ➔ False; otherwise ➔ True
!	-- Not

<expression> :== <literal>
<var>
<var> <op> <var> [... <op> <var>]

Commands and Blocks

<expression>;	Sentence

{
 <expression>;
 <expression>;
 :
 <expression>;
 }
	Sentence Block

Control

if (<condition>) <sentence$_t$> / <block$_t$> [else <sentence$_f$> / <block$_f$>]	Execute <sentence$_t$> or <block$_t$> if <condition> is true. Otherwise, (optional) execute <sentence$_f$> or <block$_f$> instead
while (<condition) <sentence> / <block>	Execute <sentence> or <block> repeatedly, while <condition> is true.
for (<var> : <set> / <collection>) <sentence> / <block>	Iterator: execute <sentence> or <block> for each element in <set> or <collection>

Input/Output

input <prompt-value> <var>;	Output the <prompt-value> and input reply into variable <var>. If <var> is a <set> or <collection>, accept a comma-separated list (if just hit "Enter" ➔ empty set/collection)
output "<string>" [<expression>];	Output <string> and then (optionaly) Evaluate and Output <expression>

Example Program 1:

```
collection class, highGradeStudents, lowGradeStudents, avgGradeStudents;
set grades, gradesHigh;
int grd;
str student;
class = {"Rafi_Suisa", "Tamar_Even", "Avi_Maoz", "Eli_Kamer", "Shlomit_Raz",
"Haim_Mizrachi", "Moshe_Samocha", "Tali_Raban", "Sharon_Tal", "Gal_Elbaz"};
gradesHigh = [];
highGradeStudents = {};


for (student:class)
{
       output "Grade for:" student;
       input  ">" grd;
       grades = grades + grd;
       if (grd >= 90])
       {
              gradesHigh = gradesHigh + grd;
              highGradeStudents = highGradeStudents + student;
       }
}

if (gradesHigh)
{
       output "Number of top grades:" |gradesHigh|;
       output "Top Grades are:" gradesHigh;
       output "High Grade Students are:" highGradeStudents
}

input "Low-grade students >", lowGradeStudents;
for (student : lowGradeStudents)
              output "Get better next time:" student;

avgGradeStudents = class – highGradeStudents – lowGradeStudents;
output "Students with good grades:" avgGradeStudents;
```

```
Program Run:

Grade for: Rafi_Suisa
> 70
Grade for: Tamar_Even
> 95
Grade for: Avi_Maoz
> 72
Grade for: Eli_Kamer
> 55
Grade for: Shlomit_Raz
> 95
Grade for: Haim_Mizrachi
> 80
Grade for: Moshe_Samocha
> 85
Grade for: Tali_Raban
> 42
Grade for: Sharon_Tal
> 100
Grade for: Gal_Elbaz
> 88
Number of top grades: 2
Top Grades are: [95, 100]
High Grade Students are: {Tamar_Even, Shlomit_Raz, Sharon_Tal}
Low-grade students > Eli_Kemer, Tali_Raban
Get better next time: Eli_Kemer
Get better next time: Tali_Raban
Students with good grades: {Rafi_Suisa, Avi_Maoz, Haim_Mizrachi, Moshe_Samocha, Gal_Elbaz}
```

Example Program 2:

```
collection highTech, gaming;
collection software, hardware, industrial;
highTech = {"Apple", "Google", "Microsoft", "Nvidia", "Adobe", "Oracle", "Sap"};
highTech = highTech + {"PayPal", "Nice", "Amdocs", "OpenAI", "Ford", "Mercedes"};
gaming = {"Sony", "Apple", "Microsoft", "Google", "Nintendo", "Playtika"};
software = {"Apple", "Microsoft", "Oracle", "Google", "Sap", "PayPal", "Playtika", "Amdocs", "OpenAI"};
hardware = {"Apple", "Nice", "Sony", "Google", "Cummins", "Nucor", "Microsoft", "Nvidia"};
industrial = {"Caterpillar", "Cummins", "Nucor"};

output "Companies that sell hardware & software:" software & hardware;
collection highSW;
highSW = software & highTech;
if (highSW == software)
        output "All software companies are high-tech companies:" highSW;
else
        output "Not all software companies are high-tech companies:" highSW;

highSW = highSW + "Playtika"
if (highSW == software)
        output "Now all software companies are high-tech companies:" highSW;
else
        output "Not all software companies are high-tech companies:" highSW;

output "Companies that do software or hardware:" software + hardware;
if (industrial & software == {})
        output "No industrial companies sell software"

output "Companies that sell Hardware but not Gaming Software:" hardware – (software & gaming)
```

```
Program Run
Companies that sell hardware & software: {Microsoft, Apple, Google}

Not all software companies are high-tech companies: {Apple, Oracle, Microsoft, Amdocs, Google,
PayPal, OpenAI, Sap}

Now all software companies are high-tech companies: {Apple, Oracle, Microsoft, Playtika, Amdocs,
Google, PayPal, OpenAI, Sap}

Companies that do software or hardware: {PayPal, Google, Playtika, Nice, Apple, Oracle, Nucor,
Microsoft, Amdocs, Nvidia, Cummins, Sony, OpenAI, Sap}

No industrial companies sell software

Companies that sell Hardware but not Gaming Software: {Sony, Nucor, Nvidia, Cummins, Nice}
```