**ПРОГРАММА ДЛЯ ЭВМ**

# ЭТАЛОННАЯ РЕАЛИЗАЦИЯ ЯЗЫКА TEMPLET

Произведение
на 17 листах

Автор (правообладатель):
_____/Востокин С.В./


Самара 2014

```
////////////////////////////////////////////////////////////
//  Эталонная реализация языка TEMPLET                    //
//  Автор: Востокин Сергей Владимирович (easts@mail.ru)   //
//  Copyright 2014                                        //
//......................................................//
//  Файл templet.h                                        //
//  Объявления структур, заполняемых при разборе          //
//  кода на языке TEMPLET                                 //
////////////////////////////////////////////////////////////
#pragma once

////////////////////////////////////////////////////////////
//  Определение синтаксиса языка TEMPLET с использованием //
//  расширенных форм Бекуса-Наура.                        //
////////////////////////////////////////////////////////////

/*
	channel = '~' ident [params] ['=' state {';' state}] '.'.
	state = ['+'] ident [ ('?'|'!')  [rules] ].
	rules = rule { ',' rule }.
	rule = ident { ',' ident } '->' ident.
	process = '*' ident [params] ['=' ((ports [';' actions]) | actions) ] '.'.
	ports = port {';' port}.
	port  = ident ':' ident ('?'|'!')[(rules ['|' '->' ident])|( '->' ident)].
	actions = action {';' action}.
	action  = ['+'] [ident ':'] disjunction ['->' ([ident] '|' ident) | ident].
	disjunction = conjunction { '|' conjunction}.
	conjunction = call {'&' call}.
	call = ident '(' [args] ')'.
	args = ident ('?'|'!') ident {',' ident ('?'|'!') ident}.
	params = '<' ident {',' ident} '>'
*/

////////////////////////////////////////////////////////////
//  В пространстве имен TEMPLET определены структуры,     //
//  заполняемые при разборе кода на языке TEMPLET.        //
//  struct channel содержит описания объектов-каналов.    //
//  struct process содержит описание объектов-процессов.  //
//  Примеры заполненных вручную структур - в templet.cpp. //
////////////////////////////////////////////////////////////

#include <string>
#include <list>

using namespace std;

namespace TEMPLET{

	struct message;

	struct channel{
		string name;         // ~name.
		list<string> params; // ~name<par1,par2,parN>.
		string init;         // ~name = +init.

		list<string> states; // включает тройки 'from ? message -> to;' в которых
состояние - это склейка 'from_message_to'
		list<message> messages; // смотри структуру message ниже
	};

	struct message{
		string name;               // from ? name -> to;
```

```
		list<string>  cli_read_states; // состояния в виде 'from_message_to' в
которых клиент может прочитать сообщение 'name'
		list<string>  srv_read_states; // тоже самое для сервера
		list<pair<string,string>> cli_write_states; // first в паре - состояние в
котором клиент может записать 'name, second - новое состояние после отправки сообщения
		list<pair<string,string>> srv_write_states; // тоже самое для сервера
	};

	struct port;
	struct call;
	struct method;
	struct param;

	struct process{
		string name;		// *name.
		list<string> params; // *name<par1,par2,parN>.
		string init;		// *name = +init().

		list<port> ports;		// *name = port : type ?.
		list<call> calls;		// *name = call().
		list<method> methods;	// для двух вызовов в '*name = label1 : call();
label2 : call().' метод - это 'call'
	};

	struct port{
		string name; // name : type ? message -> label;
		string type;
		bool asking; // истина, если 'name : type ?;' - ложь, если 'name : type !;'

		list<pair<string,string>> rules; // name : type ? message1 -> label1 |
message2 -> label2 | messsageN -> labelN;
		string if_default; // name : type ? -> default;
	};

	struct call{
		string label; // label : method(p1?m1,p2!m2,pN?mN) -> if_true | if_false;
		string method;
		list<param> params;
		string if_true;
		string if_false;
	};

	struct param{
		param(bool ap,char*p,bool
a,char*m):asking_port(ap),port(string(p)),asking(a),message(string(m)){}
		bool asking_port;
		string port; // method (port?message)
		bool asking; // port?message - true, port!message false;
		string message;

	};

	struct method{
		string name; // name() - method
		list<pair<string,string>> types;//
name(channel.message1,channel.message2,channel.messageN);
	};
}
```

```
////////////////////////////////////////////////////////////
//  Эталонная реализация языка TEMPLET                    //
//  Автор: Востокин Сергей Владимирович (easts@mail.ru)   //
//  Copyright 2014                                        //
//......................................................//
//  Файл templet.cpp                                      //
//  Примеры кода и функции генерации кода по структурам   //
//  из пространства имен TEMPLET                          //
////////////////////////////////////////////////////////////

#include "templet.h"

#include <iostream>
#include <vector>

using namespace std;

////////////////////////////////////////////////////////////
//  Имитационный runtime для отладки TEMPLET программ.    //
//  Реализует возможные варианты  исполения               //
//  параллельных программ на последовательной ЭВМ         //
//  с использованием генератора случайных чисел.          //
////////////////////////////////////////////////////////////

struct chan;
struct engine;

struct proc{virtual void recv(chan*c)=0;};
// assert(c->p==this && !sending)
// ограничение на доступ к каналам из процесса

struct chan{
	chan(engine&_e){e=&_e;sending=false;p=0;_cli=_srv=0;}
	void send(proc*_p);
	bool sending;

	proc* p;
	engine* e;
	proc *_cli,*_srv;
	//int mes;
};

struct engine{
	void run();
	vector<chan*> ready;
};

void chan::send(proc*_p)
{
	if(sending) return;
	p=_p; e->ready.push_back(this);    sending=true;
}

void engine::run()
{
	size_t rsize;
	while(rsize=ready.size()){
		int n=rand()%rsize;  auto it=ready.begin()+n;
		chan*c=*it;   ready.erase(it); c->sending=false;
		c->p->recv(c);
	}
}
```

```
////////////////////////////////////////////////////////////
//  Пример проверки тождества sin2(x)+cos2(x)=1           //
//  с использованием трех процессов, связанных            //
//  двумя каналами типа Link.                             //
//  Процесс типа Parent передает значения x               //
//  в два процесса типа Child, которые вычисляют          //
//  sin2(x) и cos2(x) соответсвенно. Результаты           //
//  передаются в Parent, который затем вычисляет сумму.   //
//  Предполагается возможность параллельного вычисления   //
//  значений  sin2(x) и cos2(x) в процессах типа Child    //
////////////////////////////////////////////////////////////
/*
	~Link =
		+BEGIN ? argCos -> CALCCOS | argSin -> CALCSIN;
		 CALCCOS ! cos2 -> END; CALCSIN ! sin2 -> END.
*/
class Link: public chan{
public:
	Link(engine&_e):chan(_e){
		state=BEGIN;
		//user code
	}
	~Link(){
		//user code
	}
	enum STATE{
		BEGIN,
		BEGIN_argCos_CALCCOS,
		BEGIN_argSin_CALCSIN,
		CALCCOS_cos2_END,
		CALCSIN_sin2_END,
	};

	struct argCos{
		//user code
		double x;
	} _argCos;

	bool rs_argCos(){
		if(sending)return false;
		return state==BEGIN_argCos_CALCCOS;
	}

	bool wc_argCos(){
		if(sending)return false;
		return state==BEGIN||false;
	}

	void csend_argCos(){
		if(state==BEGIN){
			state=BEGIN_argCos_CALCCOS;
			send(_srv);
		}
	}

	struct argSin{
		//user code
		double x;
	} _argSin;
```

```
	bool rs_argSin(){
		if(sending)return false;
		return state==BEGIN_argSin_CALCSIN;
	}
	bool wc_argSin(){
		if(sending)return false;
		return state==BEGIN||false;
	}

	void csend_argSin(){
		if(state==BEGIN){
			state=BEGIN_argSin_CALCSIN;
			send(_srv);
		}
	}

	struct cos2{
		//user code
		double c2;
	} _cos2;

	bool rc_cos2(){
		if(sending)return false;
		return state==CALCCOS_cos2_END;
	}

	bool ws_cos2(){
		if(sending)return false;
		return state==BEGIN_argCos_CALCCOS;
	}

	void ssend_cos2(){
		if(state==BEGIN_argCos_CALCCOS){
			state=CALCCOS_cos2_END;
			send(_cli);
		}
	}
	struct sin2{
		//user code
		double s2;
	} _sin2;

	bool rc_sin2(){
		if(sending)return false;
		return state==CALCSIN_sin2_END;
	}

	bool ws_sin2(){
		if(sending)return false;
		return state==BEGIN_argSin_CALCSIN;
	}

	void ssend_sin2(){
		if(state==BEGIN_argSin_CALCSIN){
			state=CALCSIN_sin2_END;
			send(_cli);
		}
	}
	//user code
private:
	STATE state;
};
```

```
/*
	*Parent =
		p1 : Link ! sin2 -> join; p2 : Link ! cos2 -> join;
		+fork(p1!argSin,p2!argCos); join(p1?sin2,p2?cos2).
*/
class Parent: public proc{
public:
	Parent(){
		_p1=_p2=0;
		//user code
		x=0;
		sin2x_plus_cos2x=0;
	}
	~Parent(){
		//user code
	}

	bool p_p1(Link&p){_p1=&p;if(p._cli==0){p._cli=this;return true;} else return
false;}
	bool p_p2(Link&p){_p2=&p;if(p._cli==0){p._cli=this;return true;} else return
false;}

	enum LABEL{
		PORT_p1,
		PORT_p2,
		CALL_fork,
		CALL_join,
	};

	void init(){recv(0);}

	void recv(chan*c){
		LABEL pos;
		bool res;

		if(c==0)pos=CALL_fork;
		else if(c==_p1)pos=PORT_p1;
		else if(c==_p2)pos=PORT_p2;
		else exit(-1);

		for(;;)switch(pos){
			case PORT_p1:
				if(_p1->rc_sin2())pos=CALL_join;
				else exit(-1);
				break;

			case PORT_p2:
				if(_p2->rc_cos2())pos=CALL_join;
				else exit(-1);
				break;

			case CALL_fork:
				res=_p1->wc_argSin() && _p2->wc_argCos();
				if(res)res=fork(&_p1->_argSin,&_p2->_argCos);
				if(res){_p1->csend_argSin();_p2->csend_argCos();return;}else
{return;}

			case CALL_join:
				res=_p1->rc_sin2() && _p2->rc_cos2();
				if(res)res=join(&_p1->_sin2,&_p2->_cos2);
				if(res){return;}else{return;}
```

```
		}
	}

	bool fork(Link::argSin*a1,Link::argCos*a2){
		// user code
		a1->x=a2->x=x;
		return true;
	}

	bool join(Link::sin2*a1,Link::cos2*a2){
		// user code
		sin2x_plus_cos2x=a1->s2+a2->c2;
		return true;
	}

	// user code
	double x;
	double sin2x_plus_cos2x;

private:
	Link* _p1;
	Link* _p2;
};

/*
	*Child =
		p : Link ? argSin -> sin | argCos -> cos;
		sin2(p?argSin,p!sin2); cos2(p?argCos,p!cos2).
*/
class Child: public proc{
public:
	Child(){
		_p=0;
		//user code
	}
	~Child(){
		//user code
	}

	bool p_p(Link&p){_p=&p;if(p._srv==0){p._srv=this;return true;} else return false;}

	enum LABEL{
		PORT_p,
		CALL_sin2,
		CALL_cos2,
	};

	void recv(chan*c){
		LABEL pos;
		bool res;

		if(c==0)exit(-1);
		else if(c==_p)pos=PORT_p;
		else exit(-1);

		for(;;)switch(pos){
			case PORT_p:
				if(_p->rs_argSin())pos=CALL_sin2;
				else if(_p->rs_argCos())pos=CALL_cos2;
				else exit(-1);
				break;
```

```
                        case CALL_sin2:
                                res=_p->rs_argSin() && _p->ws_sin2();
                                if(res)res=sin2(&_p->_argSin,&_p->_sin2);
                                if(res){_p->ssend_sin2();return;}else {return;}

                        case CALL_cos2:
                                res=_p->rs_argCos() && _p->ws_cos2();
                                if(res)res=cos2(&_p->_argCos,&_p->_cos2);
                                if(res){_p->ssend_cos2();return;}else{return;}

                }
        }

        bool sin2(Link::argSin*a1,Link::sin2*a2){
                // user code
                a2->s2=sin(a1->x)*sin(a1->x);
                return true;
        }

        bool cos2(Link::argCos*a1,Link::cos2*a2){
                // user code
                a2->c2=cos(a1->x)*cos(a1->x);
                return true;
        }

        // user code

private:
        Link* _p;
};

////////////////////////////////////////////////////////////////
// Процедура генерации кода канала на языке C++               //
// по структуре TEMPLET::channel.                             //
////////////////////////////////////////////////////////////////

void channel2cpp(TEMPLET::channel&c,ostream&f)
{
        f<<
        "class "<<c.name<<": public chan{\n"
        "public:\n"
        "       "<<c.name<<"(engine&_e):chan(_e){\n"
        "               state="<<c.init<<";\n"
        "               //user code\n"
        "       }\n"
        "       ~"<<c.name<<"(){\n"
        "               //user code\n"
        "       }\n\n"

        "       enum STATE{\n";
        for(auto it=c.states.begin();it!=c.states.end();it++){
                f<<
                "               "<<*it<<",\n";
        }
        f<<
        "       };\n\n";

        for(auto it=c.messages.begin();it!=c.messages.end();it++){
                f<<
                "       struct "<<(*it).name<<"{\n"
                "               //user code\n"
                "       } _"<<(*it).name<<";\n\n";
```

```
		if(!(*it).cli_read_states.empty()){
			f<<
			"	bool rc_"<<(*it).name<<"(){\n"
			"		if(sending)return false;\n"
			"		return ";

			int count=1,size=(*it).cli_read_states.size();
			for(auto s=(*it).cli_read_states.begin();s!
=(*it).cli_read_states.end();s++,count++){
				if(count<size)f<< "state==" << *s << "||";
				else f<< "state==" << *s <<";\n";
			}
			f<<
			"	}\n\n";
		}

		if(!(*it).srv_read_states.empty()){
			f<<
			"	bool rs_"<<(*it).name<<"(){\n"
			"		if(sending)return false;\n"
			"		return ";

			int count=1,size=(*it).srv_read_states.size();
			for(auto s=(*it).srv_read_states.begin();s!
=(*it).srv_read_states.end();s++,count++){
				if(count<size)f<< "state==" << *s << "||";
				else f<< "state==" << *s <<";\n";
			}
			f<<
			"	}\n\n";
		}

		if(!(*it).cli_write_states.empty()){
			f<<
			"	bool wc_"<<(*it).name<<"(){\n"
			"		if(sending)return false;\n"
			"		return ";

			int count=1,size=(*it).cli_write_states.size();
			for(auto s=(*it).cli_write_states.begin();s!
=(*it).cli_write_states.end();s++,count++){
				if(count<size)f<< "state==" << (*s).first << "||";
				else f<< "state==" << (*s).first <<";\n";
			}
			f<<
			"	}\n\n";
		}

		if(!(*it).srv_write_states.empty()){
			f<<
			"	bool ws_"<<(*it).name<<"(){\n"
			"		if(sending)return false;\n"
			"		return ";

			int count=1,size=(*it).srv_write_states.size();
			for(auto s=(*it).srv_write_states.begin();s!
=(*it).srv_write_states.end();s++,count++){
				if(count<size)f<< "state==" << (*s).first << "||";
				else f<< "state==" << (*s).first <<";\n";
			}
			f<<
```

```
                        "       }\n\n";
                }

                if(!(*it).cli_write_states.empty()){
                        f<<
                        "       void csend_"<<(*it).name<<"(){\n";

                        for(auto s=(*it).cli_write_states.begin();s!
=(*it).cli_write_states.end();s++){
                                f<<
                                "               if(state==" << (*s).first << "){\n"
                                "                       state==" << (*s).second <<";\n"
                                "                       send(_srv);\n"
                                "               }\n";
                        }

                        f<<"    }\n\n";
                }

                if(!(*it).srv_write_states.empty()){
                        f<<
                        "       void ssend_"<<(*it).name<<"(){\n";

                        for(auto s=(*it).srv_write_states.begin();s!
=(*it).srv_write_states.end();s++){
                                f<<
                                "               if(state==" << (*s).first << "){\n"
                                "                       state==" << (*s).second <<";\n"
                                "                       send(_cli);\n"
                                "               }\n";
                        }

                        f<<"    }\n\n";
                }
        }
        f<<
        "       //user code\n\n"
        "private:\n"
        "       STATE state;\n"
        "};\n\n";

}

////////////////////////////////////////////////////////////
// Процедура генерации кода процесса на языке C++         //
// по структуре TEMPLET::process.                         //
////////////////////////////////////////////////////////////

void process2cpp(TEMPLET::process&p,ostream&f)
{
        f<<
        "class "<<p.name<<": public proc{\n"
        "public:\n"
        "       "<<p.name<<"(){\n";

        if(!p.ports.empty()){
                f<<"            ";
                for(auto it=p.ports.begin();it!=p.ports.end();it++){
                        f<<"_"<<(*it).name<<"=";
                }
                f<<"0;\n";
        }
```

```
	f<<
	"		//user code\n"
	"	}\n"
	"	~"<<p.name<<"(){\n"
	"		//user code\n"
	"	}\n\n";

	for(auto it=p.ports.begin();it!=p.ports.end();it++){
		if((*it).asking) f<<"	bool p_"<<(*it).name<<"(Link&p)
{_"<<(*it).name<<"=&p;if(p._srv==0){p._srv=this;return true;} else return false;}\n";
		else f<<"	bool p_"<<(*it).name<<"(Link&p)
{_"<<(*it).name<<"=&p;if(p._cli==0){p._cli=this;return true;} else return false;}\n";

	}
	f<<"\n";

	f<<
	"	enum LABEL{\n";
	for(auto it=p.ports.begin();it!=p.ports.end();it++){
		f<<	"		PORT_"<<(*it).name<<",\n";
	}
	for(auto it=p.calls.begin();it!=p.calls.end();it++){
		f<<	"		CALL_"<<(*it).label<<",\n";
	}

	f<<
	"	};\n\n";

	if(!p.init.empty())
		f<<"	void init(){recv(0);}\n\n";

	f<<
	"	void recv(chan*c){\n"
	"		LABEL pos;\n"
	"		bool res;\n"
	"\n";
	if(p.init.empty())	f<<"		if(c==0)exit(-1);\n";
	else			f<<"		if(c==0)pos=CALL_"<<p.init<<";\n";

	for(auto it=p.ports.begin();it!=p.ports.end();it++){
		f<<"		else if(c==_"<<(*it).name<<")pos=PORT_"<<(*it).name<<";\n";
	}

	f<<
	"		else exit(-1);\n\n";

	f<<
	"		for(;;)switch(pos){\n";
	for(auto it=p.ports.begin();it!=p.ports.end();it++){
		f<<	"			case PORT_"<<(*it).name<<":\n";
		bool first=true;
		for(auto rule=(*it).rules.begin();rule!=(*it).rules.end();rule++){
			if(first){first=false;
				f<<"				if(_"<<(*it).name<<"-
>r"<<((*it).asking?"s":"c")<<"_"<<(*rule).first<<"())pos=CALL_"<<(*rule).second<<";\n";
			}
			else{
				f<<"				else if(_"<<(*it).name<<"-
>r"<<((*it).asking?"s":"c")<<"_"<<(*rule).first<<"())pos=CALL_"<<(*rule).second<<";\n";
			}
		}
```

```
                if(!(*it).if_default.empty())
                        f<<"                        else pos=CALL_"<<(*it).if_default<<";\n";
                else
                        f<<"                        else exit(-1);\n";
                f<<"                        break;\n\n";
        }
        for(auto it=p.calls.begin();it!=p.calls.end();it++){
                f<<     "                   case CALL_"<<(*it).label<<":\n";

                auto params=(*it).params;
                int count=1,size=params.size();
                f<<"                        res=";
                for(auto param=params.begin();param!=params.end();param++,count++){
                        if(count<size)      f<<"_"<<(*param).port<<"->"<<((*param).asking?"r":"w")<<((*param).asking_port?"s":"c")<<"_"<<(*param).message<<"() && ";
                        else f<<"_"<<(*param).port<<"->"<<((*param).asking?"r":"w")<<((*param).asking_port?"s":"c")<<"_"<<(*param).message<<"();\n";
                }
                count=1;
                f<<"                        if(res)res="<<(*it).method<<"(";
                for(auto param=params.begin();param!=params.end();param++,count++){
                        if(count<size)      f<<"&_"<<(*param).port<<"->_"<<(*param).message<<",";
                        else f<<"&_"<<(*param).port<<"->_"<<(*param).message<<");\n";
                }

                f<<"                        if(res){";
                for(auto param=params.begin();param!=params.end();param++){
                        if(!(*param).asking)
                        f<<"_"<<(*param).port<<"->"<<((*param).asking_port?"s":"c")<<"send_"<<(*param).message<<"();";
                }

                if(!(*it).if_true.empty())f<<"pos=CALL_"<<(*it).if_true<<";break;}";
                else f<<"return;}";
                if(!(*it).if_false.empty())f<<"else {pos=CALL_"<<(*it).if_false<<";break;}\n";
                else f<<"else {return;}\n";
        }
        f<<
        "               }\n";

        f<<
        "       }\n\n";

        for(auto it=p.methods.begin();it!=p.methods.end();it++){
                auto types=(*it).types;
                int a=1,size=types.size();
                f<<
                "       bool "<<(*it).name<<"(";
                for(auto type=types.begin();type!=types.end();type++,a++){
                        f<<(*type).first<<"::"<<(*type).second<<"*a"<<a<<(a<size?",":"");
                }
                f<<
                "){\n"
                "               // user code\n"
                "               return true;\n"
                "       }\n\n";
        }
```

```
	f<<
	"	// user code\n"
	"private:\n";

	for(auto it=p.ports.begin();it!=p.ports.end();it++){
		f<<"	"<<(*it).type<<"* _"<<(*it).name<<";\n";
	}

	f<<"}\n\n";
}

int main(int argc, char* argv[])
{

////////////////////////////////////////////////////////////
// Пример заполненной структуры channel для канала Link.    //
////////////////////////////////////////////////////////////

	cout<<
		"/*\n"
		"~Link = \n"
		"	+BEGIN ? argCos -> CALCCOS | argSin -> CALCSIN;\n"
		"	 CALCCOS ! cos2 -> END; CALCSIN ! sin2 -> END.\n"
		"*/\n";

	TEMPLET::channel _Link;

	_Link.name="Link";
	_Link.init="BEGIN";

	list<string> states;
	states.push_back(string("BEGIN"));
	states.push_back(string("BEGIN_argCos_CALCCOS"));
	states.push_back(string("BEGIN_argSin_CALCSIN"));
	states.push_back(string("CALCCOS_cos2_END"));
	states.push_back(string("CALCSIN_sin2_END"));
	_Link.states=states;

	TEMPLET::message MES_argCos,MES_argSin,MES_cos2,MES_sin2;

	MES_argCos.name=string("argCos");

	list<string> crs_argCos;
	MES_argCos.cli_read_states=crs_argCos;

	list<string> srs_argCos;
	srs_argCos.push_back(string("BEGIN_argCos_CALCCOS"));
	MES_argCos.srv_read_states=srs_argCos;

	list<pair<string,string>> cws_argCos;

cws_argCos.push_back(pair<string,string>(string("BEGIN"),string("BEGIN_argCos_CALCCOS")))
;
	MES_argCos.cli_write_states=cws_argCos;

	list<pair<string,string>> sws_argCos;
	MES_argCos.srv_write_states=sws_argCos;

	MES_argSin.name=string("argSin");

	list<string> crs_argSin;
	MES_argSin.cli_read_states=crs_argSin;
```

```
        list<string> srs_argSin;
        srs_argSin.push_back(string("BEGIN_argSin_CALCSIN"));
        MES_argSin.srv_read_states=srs_argSin;

        list<pair<string,string>> cws_argSin;

cws_argSin.push_back(pair<string,string>(string("BEGIN"),string("BEGIN_argSin_CALCSIN")))
;
        MES_argSin.cli_write_states=cws_argSin;

        list<pair<string,string>> sws_argSin;
        MES_argSin.srv_write_states=sws_argSin;

        MES_cos2.name=string("cos2");

        list<string> crs_cos2;
        crs_cos2.push_back(string("CALCCOS_cos2_END"));
        MES_cos2.cli_read_states=crs_cos2;

        list<string> srs_cos2;
        MES_cos2.srv_read_states=srs_cos2;

        list<pair<string,string>> cws_cos2;
        MES_cos2.cli_write_states=cws_cos2;

        list<pair<string,string>> sws_cos2;

sws_cos2.push_back(pair<string,string>(string("BEGIN_argCos_CALCCOS"),string("CALCCOS_cos
2_END")));
        MES_cos2.srv_write_states=sws_cos2;

        MES_sin2.name=string("sin2");

        list<string> crs_sin2;
        crs_sin2.push_back(string("CALCSIN_sin2_END"));
        MES_sin2.cli_read_states=crs_sin2;

        list<string> srs_sin2;
        MES_sin2.srv_read_states=srs_sin2;

        list<pair<string,string>> cws_sin2;
        MES_sin2.cli_write_states=cws_sin2;

        list<pair<string,string>> sws_sin2;

sws_sin2.push_back(pair<string,string>(string("BEGIN_argSin_CALCSIN"),string("CALCSIN_sin
2_END")));
        MES_sin2.srv_write_states=sws_sin2;

        list<TEMPLET::message> messages;
        messages.push_back(MES_argCos);
        messages.push_back(MES_argSin);
        messages.push_back(MES_cos2);
        messages.push_back(MES_sin2);

        _Link.messages=messages;
        channel2cpp(_Link,cout);
```

```
////////////////////////////////////////////////////////////
// Пример заполненной структуры process для процесса Parent.//
////////////////////////////////////////////////////////////////

	cout<<
		"/*\n"
		"*Parent = \n"
		"	p1 : Link ! sin2 -> join; p2 : Link ! cos2 -> join;\n"
		"	+fork(p1!argSin,p2!argCos); join(p1?sin2,p2?cos2).\n"
		"*/\n";

	TEMPLET::process _Parent;
	_Parent.name="Parent";
	_Parent.init="fork";

	TEMPLET::port _p1,_p2;

	_p1.name="p1";
	_p1.type="Link";
	_p1.asking=false;
	_p1.rules.push_back(pair<string,string>("sin2","join"));
	_Parent.ports.push_back(_p1);

	_p2.name="p2";
	_p2.type="Link";
	_p2.asking=false;
	_p2.rules.push_back(pair<string,string>("cos2","join"));
	_Parent.ports.push_back(_p2);

	TEMPLET::call _fork,_join;

	_fork.label="fork";
	_fork.method="fork";
	_fork.params.push_back(TEMPLET::param(false,"p1",false,"argSin"));
	_fork.params.push_back(TEMPLET::param(false,"p2",false,"argCos"));
	_Parent.calls.push_back(_fork);

	_join.label="join";
	_join.method="join";
	_join.params.push_back(TEMPLET::param(false,"p1",true,"sin2"));
	_join.params.push_back(TEMPLET::param(false,"p2",true,"cos2"));
	_Parent.calls.push_back(_join);

	TEMPLET::method _m_fork,_m_join;

	_m_fork.name="fork";
	_m_fork.types.push_back(pair<string,string>("Link","argSin"));
	_m_fork.types.push_back(pair<string,string>("Link","argCos"));
	_Parent.methods.push_back(_m_fork);

	_m_join.name="join";
	_m_join.types.push_back(pair<string,string>("Link","sin2"));
	_m_join.types.push_back(pair<string,string>("Link","cos2"));
	_Parent.methods.push_back(_m_join);

	process2cpp(_Parent,cout);
```

```
//////////////////////////////////////////////////////////
// Пример заполненной структуры process для процесса Child. //
//////////////////////////////////////////////////////////////
	cout<<
		"/*\n"
		"*Child = \n"
		"	p : Link ? argSin -> sin | argCos -> cos;\n"
		"	sin2(p?argSin,p!sin2); cos2(p?argCos,p!cos2).\n"
		"*/\n";
	TEMPLET::process _Child;
	_Child.name="Child";

	TEMPLET::port _p;
	_p.name="p";
	_p.type="Link";
	_p.asking=true;
	_p.rules.push_back(pair<string,string>("argSin","sin"));
	_p.rules.push_back(pair<string,string>("argCos","cos"));
	_Child.ports.push_back(_p);

	TEMPLET::call _sin2,_cos2;

	_sin2.label="sin2";
	_sin2.method="sin2";
	_sin2.params.push_back(TEMPLET::param(true,"p",true,"argSin"));
	_sin2.params.push_back(TEMPLET::param(true,"p",false,"sin2"));
	_Child.calls.push_back(_sin2);

	_cos2.label="cos2";
	_cos2.method="cos2";
	_cos2.params.push_back(TEMPLET::param(true,"p",true,"argCos"));
	_cos2.params.push_back(TEMPLET::param(true,"p",false,"cos2"));
	_Child.calls.push_back(_cos2);

	TEMPLET::method _m_sin2,_m_cos2;

	_m_sin2.name="sin2";
	_m_sin2.types.push_back(pair<string,string>("Link","argSin"));
	_m_sin2.types.push_back(pair<string,string>("Link","sin2"));
	_Child.methods.push_back(_m_sin2);

	_m_cos2.name="cos2";
	_m_cos2.types.push_back(pair<string,string>("Link","argCos"));
	_m_cos2.types.push_back(pair<string,string>("Link","cos2"));
	_Child.methods.push_back(_m_cos2);
	process2cpp(_Child,cout);
//////////////////////////////////////////////////////////
//  Сборка процессов и каналов в структуру                  //
//  [c1:Child]--l1:Link-->[p:Parent]--l2:Link-->[c1:Child]  //
//  Запуск, сравнение результата с 1.                       //
//////////////////////////////////////////////////////////////
	engine e;
	Link l1(e),l2(e);
	Parent p;
	Child  c1,c2;
	p.p_p1(l1);p.p_p2(l2);
	c1.p_p(l1);c2.p_p(l2);
	cout<<"input x:"; cin>>p.x;
	p.init(); e.run();
	cout<<"sin2(x) + cos2(x) = "<<p.sin2x_plus_cos2x;
	return 0;
}
```

Всего пронумеровано и прошнуровано _17_ листов
исходного текста ПрЭВМ
Правообладатель_____/Востокин С.В./