Interpreter języka W++

Piotr Gdowski

1 Składnia języka W++

		Składnia abstrakcyjna	*Składnia konkretna*	*Opis*
p	::=	program(dt*, df*, dv*, c)	dt* df* vars dv* in c	deklaracja typów, funkcji oraz zmiennych
dt	::=	newtype(t, τ)	Type $t = \tau$;	deklaracja typu
df	::=	function(f, $(x : \tau)$*, dv*, c, e)	function($(x : \tau)$*) = vars dv* in c return e;	deklaracja funkcji
dv	::=	newvar(x, e)	$x := e$	deklaracja zmiennej
		newpointer(x, e)	$x :=$ *new* e	deklaracja wskaźnika
		typednewvar(τ, x, e)	$\tau\ x := e$	deklaracja zmiennej z własną adnotacją typową
		typednewpointer(τ, x, e)	$\tau\ x :=$ *new* e	deklaracja wskaźnika z własną adnotacją typową
c	::=	*skip*	*skip*	pusta instrukcja
		compose(c_1, c_2)	c; c	złożenie instrukcji
		newvars(dv*, c)	vars dv* in c	deklaracja lokalnych zmiennych i wskaźników
		if(e, c_1, c_2)	if e then c_1 else c_2	instrukcja if
		while(e, c)	while e do c	instrukcja while
		varassign(x, e)	$x := e$	przypisanie wartości do zmiennej
		pointerassign(x, e)	$*x := e$	przypisanie wartości do wartości pod wskaźnikiem
		callassign(x, f, $\vec{x}$)	$x :=$ f($\vec{x}$)	przypisanie wartości zwróconej przez funkcję
e	::=	$\underline{n}$	n	liczba
		x	x	wartość zmiennej
		$*x$	$*x$	wartość zmiennej wskazywanej przez wskaźnik
		plus(e_1, e_2)	$e + e$	dodawanie
		mult(e_1, e_2)	$e * e$	mnożenie
		minus(e)	-e	minus
		tuple($\vec{e}$)	tuple(e_1, ... , e_n)	krotka
		indexer($e, \underline{n}$)	$e[n]$	wyjęcie z krotki wskazanej wartości
		injection(l, e)	$l.e$	włożenie
		case($e, (l.x \to e)^+$)	case e $\{l_1.x \to e_1, ..., l_n.x \to e_n, \}$	case
τ	::=	int	int	typ liczb całkowitych
		TTuple($\vec{\tau}$)	Tuple(τ_1, ..., τ_n)	typ krotki
		TSum($(l \to \tau)^+$)	Sum($l_1 \to \tau_1, ..., l_n \to \tau_n$)	typ sum rozłącznych
		TPtr(τ)	Ptr(τ)	typ wskaźnikowy
		TAbs(t)	t	typ abstrakcyjny

2 System typów języka W++

- Γ jest listą zawierającą zmienne wraz z ich typami
- F jest listą zawierającą funkcje wraz z ich definicjami i zwracanymi wyrażeniami
- T jest listą zawierającą nazwy predefiniowanych typów wraz z jego prawdziwymi typami

2.1 Typowanie wyrażeń

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}$$

$$\frac{}{\Gamma \vdash \underline{n} : int}$$

$$\frac{\Gamma \vdash e : TPtr(\tau)}{\Gamma \vdash *e : \tau}$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash plus(e_1, e_2) : int}$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash mult(e_1, e_2) : int}$$

$$\frac{\Gamma \vdash e : int}{\Gamma \vdash -e : int}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad ... \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash tuple(e_1, ..., e_n) : TTuple(\tau_1, ..., \tau_n)}$$

$$\frac{\Gamma \vdash e : TTuple(\tau_1, ..., \tau_n) \quad \underline{i} : int}{\Gamma \vdash e[i] : \tau_i}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash l.e : TSum(l \to \tau)}$$

$$\frac{\Gamma \vdash e : TSum(l_1 \to \tau_1, ..., l_n \to \tau_n) \quad \Gamma, x : \tau_1 \vdash e_1 : \tau \quad ... \quad \Gamma, x : \tau_n \vdash e_n : \tau}{\Gamma \vdash case(e, l_1.x \to e_1, ..., l_n.x \to e_n) : \tau}$$

2.1.1 Relacja $\preceq$ na typach (podtypowanie)

Rozważmy instrukcję $x := l.5$. Przy pomocy powyższych reguł typowania możemy wyinferować typ wyrażenia po lewej stronie - jest to $Sum(l \to int)$. Oznacza to, że również x powinien być dokładnie takiego typu, aby instrukcja mogła się poprawnie otypować. Wówczas każdy typ sumy miałby dokładnie jeden konstruktor, nie byłoby bowiem możliwości 'zgadnięcia' pozostałych konstruktorów. Z tego powodu dodałem możliwość zdefiniowania w adnotacji typowej wszystkich konstruktorów. Tym sposobem, mamy możliwość przypisywania do zmiennych 'większych' typów wyrażenia 'mniejszych' typów. Intuicyjnie, $T \vdash Sum(l \to int) \preceq Sum(l \to int|r \to int)$, gdzie w T trzymamy predefiniowane typy. Pełna definicja relacji:

$$\frac{}{T \vdash int \preceq int}$$

$$\frac{T \vdash \tau \preceq \tau_2}{T, x : \tau \vdash TAbs(x) \preceq \tau_2}$$

$$\frac{T \vdash \tau_1 \preceq \tau}{T, x : \tau \vdash \tau_1 \preceq TAbs(x)}$$

$$\frac{T \vdash \tau_1 \preceq \tau_2}{T \vdash TPtr(\tau_1) \preceq TPtr(\tau_2)}$$

$$\frac{T \vdash \tau_1 \preceq \tau_1' \quad ... \quad T \vdash \tau_n \preceq \tau_n'}{T \vdash TTuple(\tau_1, ..., \tau_n) \preceq TTuple(\tau_1', ..., \tau_n')}$$

$$\frac{\forall_i \exists_j \;\; l_i = k_j \implies \tau_i \preceq \tau_j'}{T \vdash TSum(l_1 \to \tau_1, ..., l_n \to \tau_n) \preceq TSum(k_1 \to \tau_1', ..., k_m \to \tau_m')}$$

2.2 Poprawność sformowania instrukcji

$$\frac{}{T, F, \Gamma \vdash skip\ \ cwf}$$

$$\frac{T, F, \Gamma \vdash c_1\ \ cwf \quad T, F, \Gamma \vdash c_1\ \ cwf}{T, F, \Gamma \vdash compose(c_1, c_2)\ \ cwf}$$

$$\frac{T, F, \Gamma \vdash c\ \ cwf}{T, F, \Gamma \vdash newvars([], c)\ \ cwf}$$

$$\frac{T, F, \Gamma \vdash e : \tau \quad T, F, \Gamma, x : \tau \vdash newvars(declariations, c)\ \ cwf}{T, F, \Gamma \vdash newvars(newvar(x, e) :: declarations, c)\ \ cwf}$$

$$\frac{T, F, \Gamma \vdash e : \tau \quad T, F, \Gamma, x : Ptr(\tau) \vdash newvars(declariations, c)\ \ cwf}{T, F, \Gamma \vdash newvars(newpointer(x, e) :: declarations, c)\ \ cwf}$$

$$\frac{T, F, \Gamma \vdash e : \tau' \quad T \vdash \tau' \preceq \tau \quad T, F, \Gamma, x : \tau \vdash newvars(declariations, c)\ \ cwf}{T, F, \Gamma \vdash newvars(typednewvar(\tau, x, e) :: declarations, c)\ \ cwf}$$

$$\frac{T, F, \Gamma \vdash e : \tau' \quad T \vdash \tau' \preceq \quad T, F, \Gamma, x : Ptr(\tau) \vdash newvars(declariations, c)\ \ cwf}{T, F, \Gamma \vdash newvars(typednewpointer(\tau, x, e) :: declarations, c)\ \ cwf}$$

$$\frac{\Gamma \vdash e : int \quad T, F, \Gamma \vdash c_1\ \ cwf \quad T, F, \Gamma \vdash c_2\ \ cwf}{T, F, \Gamma \vdash if(e, c_1, c_2)\ \ cwf}$$

$$\frac{\Gamma \vdash e : int \quad T, F, \Gamma \vdash c\ \ cwf}{T, F, \Gamma \vdash while(e, c)\ \ cwf}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash x : \tau}{T, F, \Gamma \vdash varassign(x, e)\ \ cwf}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash x : TPtr(\tau)}{T, F, \Gamma \vdash pointerassign(x, e)\ \ cwf}$$

$$\frac{\Gamma \vdash x : \tau \quad \Gamma \vdash e_1 : \tau_1 \quad ... \quad \Gamma \vdash e_n : \tau_n}{T, F, (f, x_1 : \tau_1, ..., x_n : \tau_n, e : \tau), \Gamma \vdash callassign(x, f, e_1, ..., e_n)\ \ cwf}$$

2.3 Poprawność sformowania programu

2.3.1 Poprawność sformowania typów

Na potrzeby typów definiowanych przez użytkownika, potrzebujemy nowego jugdementu sprawdzającego poprawność sformowania typu przy pewnym zbiorze typów T: $T \vdash \tau\ \ twf$. Reguły są następujące:

$$\frac{}{T \vdash int\ \ twf}$$

$$\frac{}{T, x : \tau \vdash TAbs(x)\ \ twf}$$

$$\frac{T \vdash \tau\ \ twf}{T \vdash TPtr(\tau)\ \ twf}$$

$$\frac{T \vdash \tau_1\ \ twf \quad ... \quad T \vdash \tau_n\ \ twf}{T \vdash TTuple(\tau_1, ..., \tau_n)\ \ twf}$$

$$\frac{T \vdash \tau_1\ \ twf \quad ... \quad T \vdash \tau_n\ \ twf}{T \vdash TSum(l_1 \to \tau_1, ..., l_n \to \tau_n)\ \ twf}$$

2.3.2 Poprawność sformowania funkcji

Deklaracja funkcji $function(f, x_1 : \tau_1, ..., x_n : \tau_n, dvs', c, e)$ zawiera nazwę funkcji, nazwy argumentów wraz z ich typami, deklarację zmiennych globalnych w tej funkcji, ciało oraz wyrażenie zwracane. Zwracane wyrażenie ma dostęp do zmiennych globalnych oraz argumentów funkcji. Z ciała funkcji można wywołać samego siebie, bądź funkcję zdefiniowaną wyżej w kodzie. Reguły poprawnego sformowania funkcji:

$$\frac{\begin{array}{r} T \vdash \tau_1\ \ twf \quad ... \quad T \vdash \tau_n\ \ twf \quad T, F, (x_1 : \tau_1, ..., x_n : \tau_n) \vdash newvars(dvs', newvar(x', e))\ \ cwf \quad (x'\ \ fresh) \\ T, F, (f, x_1 : \tau_1, ..., x_n : \tau_n, e : \tau), (x_1 : \tau_1, ..., x_n : \tau_n) \vdash newvars(dvs', c)\ \ cwf \end{array}}{T, F \vdash function(f, x_1 : \tau_1, ..., x_n : \tau_n, dvs', c, e)\ \ fwf}$$

2.3.3 Właściwe reguły

$$\frac{\emptyset, \emptyset, \emptyset \vdash program(dts, dfs, dvs, c)\ \ pwf}{program(dts, dfs, dvs, c)\ \ pwf}$$

$$\frac{T \vdash \tau\ \ twf \quad T, (t : \tau), F, \Gamma \vdash program(dts, dfs, dvs, c)\ \ pwf}{T, F, \Gamma \vdash program(newtype(t, \tau) :: dts, dfs, dvs, c)\ \ pwf}$$

$$\frac{T, F \vdash function(f, x_1 : \tau_1, ..., x_n : \tau_n, dvs', c, e)\ \ fwf \quad T, F, (f, x_1 : \tau_1, ..., x_n : \tau_n, e : \tau), \Gamma \vdash program([], dfs, dvs, c)\ \ pwf}{T, F, \Gamma \vdash program([], function(f, x_1 : \tau_1, ..., x_n : \tau_n, dvs', c, e) :: dfs, dvs, c)\ \ pwf}$$

$$\frac{T, F, \Gamma \vdash newvars(dvs, c)\ \ cwf}{T, F, \Gamma \vdash program([], [], dvs, c)\ \ pwf}$$

3 Semantyka operacyjna języka W++

3.1 Definicje wartości i postaci końcowych

$$\frac{}{\underline{n}\ \ val}$$

$$\frac{}{EPtr(e)\ \ val}$$

$$\frac{e_1\ \ val \quad ... \quad e_n\ \ val}{ETuple(e_1, ..., e_n)\ \ val}$$

$$\frac{e\ \ val}{l.e\ \ val}$$

$EPtr(e)$ jest sztucznym wyrażeniem, służy jako wartość dla zmiennej typu wskaźnikowego

$$\frac{}{skip\ \ cfinal}$$

$$\frac{}{Program(skip)\ \ pfinal}$$

3.2 Semantyka małych kroków

- M jest pamięcią programu, przechowującą wartości zmiennych.
- H jest stertą, pamiętającą wartości wskazywane przez wskaźniki.
- F jest listą zdefiniowanych funkcji.

3.2.1 Wyrażenia

$$\frac{}{H, (M, x \hookrightarrow v)\ \ x \longmapsto H, (M, x \hookrightarrow v)\ \ v}$$

$$\frac{H, M\ \ e_1 \longmapsto H, M\ \ e_1'}{H, M\ \ e_1 + e_2 \longmapsto H, M\ \ e_1' + e_2}$$

$$\frac{e_1\ \ val \quad H, M\ \ e_2 \longmapsto H, M\ \ e_2'}{H, M\ \ e_1 + e_2 \longmapsto H, M\ \ e_1 + e_2'}$$

$$\frac{H, M\ \ e_1 \longmapsto H, M\ \ e_1'}{H, M\ \ e_1 + e_2 \longmapsto H, M\ \ e_1' * e_2}$$

$$\frac{H, M\ \ e \longmapsto H, M\ \ e'}{H, M\ \ -e \longmapsto H, M\ \ -e'}$$

$$\frac{}{H, M\ \ -\underline{n} \longmapsto H, M\ \ \underline{-n}}$$

$$\frac{e_1\ \ val \quad ... \quad e_{i-1}\ \ val \quad H, M\ \ e_i \longmapsto H, M\ \ e_i'}{H, M\ \ ETuple(e_1, ..., e_i, ..., e_n) \longmapsto H, M\ \ ETuple(e_1, ..., e_i', ..., e_n)}$$

$$\frac{H, M\ \ t \longmapsto H, M\ \ t'}{H, M\ \ t[\underline{m}] \longmapsto H, M\ \ t'[\underline{m}]}$$

$$\frac{ETuple(e_1, ..., e_i, ..., e_n)\ \ val}{H, M\ \ ETuple(e_1, ..., e_i, ..., e_n)[\underline{m}] \longmapsto H, M\ \ e_m}$$

$$\frac{H, M\ \ e \longmapsto H, M\ \ e'}{H, M\ \ l.e \longmapsto H, M\ \ l.e'}$$

$$\frac{H, M\ \ e \mapsto H, M\ \ e'}{H, M\ \ case\ \ e\ \ of\{l_1.x \to e_1, ..., l_n.x \to e_n\} \longmapsto H, M\ \ case\ \ e'\ \ of\{l_1.x \to e_1, ..., l_n.x \to e_n\}}$$

$$\frac{l_i.e\ \ val}{H, M\ \ case\ \ l_i.e\ \ of\{l_1.x \to e_1, ..., l_i.x \to e_i, ..., l_n.x \to e_n\} \longmapsto H, M\ \ [e/x]e_i}$$

$$\frac{H, M\ \ e \longmapsto H, M\ \ e'}{H, M\ \ *e \longmapsto H, M\ \ *e'}$$

$$\frac{}{(H, x \hookrightarrow v), M\ \ *EPtr(x) \longmapsto (H, x \hookrightarrow v), M\ \ v}$$

3.3 Instrukcje

$$\frac{H, M\ \ e \longmapsto H, M\ \ e'}{F, H, M\ \ x := e \longmapsto F, H, M\ \ x := e'}$$

$$\frac{e\ \ val}{F, H, (M, x \hookrightarrow \_)\ \ x := e \longmapsto F, H, (M, x \hookrightarrow e)\ \ skip}$$

$$\frac{F, H, M\ \ c_1 \longmapsto F, H', M'\ \ c_1'}{F, H, M\ \ (c_1; c_2) \longmapsto F, H', M'\ \ (c_1'; c_2)}$$

$$\frac{}{F, H, M\ \ (skip; c_2) \longmapsto F, H, M\ \ c_2}$$

$$\frac{H, M\ \ e \longmapsto H, M\ \ e'}{F, H, M\ \ IfThenElse(e, c_1, c_2) \longmapsto F, H, M\ \ IfThenElse(e', c_1, c_2)}$$

$$\frac{e = \underline{0}}{F, H, M\ \ IfThenElse(e, c_1, c_2) \longmapsto F, H, M\ \ c_1}$$

$$\frac{e \neq \underline{0}}{F, H, M\ \ IfThenElse(e, c_1, c_2) \longmapsto F, H, M\ \ c_2}$$

$$\frac{}{F, H, M\ \ While(e, c) \longmapsto F, H, M\ \ IfThenElse(e, skip, (c; While(e, c)))}$$

$$\frac{H, M\ \ e \longmapsto H, M\ \ e'}{F, H, M\ \ newvar(x, e, c) \longmapsto F, H, M\ \ newvar(x, e', c)}$$

$$\frac{e\ \ val \quad x \notin M \quad F, H, (M, x \hookrightarrow e)\ \ c \longmapsto F, H', (M', x \hookrightarrow e')\ \ c'}{F, H, M\ \ newvar(x, e, c) \longmapsto F, H, M\ \ newvar(x, e, c')}$$

$$\frac{e\ \ val \quad x \in M \quad F, H, M\ \ c \longmapsto F, H', M'\ \ c'}{F, H, M\ \ newvar(x, e, c) \longmapsto F, H, M\ \ newvar(x, e, c')}$$

$$\frac{c\ \ cfinal}{F, H, (M, x \hookrightarrow \_)\ \ newvar(x, e, c) \longmapsto F, H, M\ \ skip}$$

$$\frac{}{F, H, M\ \ typednewvar(\tau, x, e, c) \longmapsto F, H, M\ \ newvar(x, e, c)}$$

$$\frac{H, M\ \ e \longmapsto H, M\ \ e'}{F, H, M\ \ newpointer(x, e, c) \longmapsto F, H, M\ \ newpointer(x, e', c)}$$

$$\frac{e\ \ val \quad x \notin M \quad F, (H, x \hookrightarrow e), (M, x \hookrightarrow EPtr(x))\ \ c \longmapsto F, (H', x \hookrightarrow e'), (M', x \hookrightarrow EPtr(x'))\ \ c'}{F, H, M\ \ newpointer(x, e, c) \longmapsto F, H, M\ \ newpointer(x, e, c')}$$

$$\frac{e\ \ val \quad x \in M \quad F, H, M\ \ c \longmapsto F, H', M'\ \ c'}{F, H, M\ \ newpointer(x, e, c) \longmapsto F, H, M\ \ newpointer(x, e, c')}$$

$$\frac{c\ \ cfinal}{F, H, (M, x \hookrightarrow \_)\ \ newpointer(x, e, c) \longmapsto F, H, M\ \ skip}$$

$$\frac{}{F, H, M\ \ typednewpointer(\tau, x, e, c) \longmapsto F, H, M\ \ newpointer(x, e, c)}$$

$$\frac{e_1\ \ val \quad ... \quad e_{i-1}\ \ val \quad H, M\ \ e_i \longmapsto H, M\ \ e_i'}{F, H, M\ \ x := f(e_1, ..., e_i, ..., e_n) \longmapsto F, H, M\ \ x := f(e_1, ..., e_i', ..., e_n)}$$

$$\frac{e_1 \ val \quad ... \quad e_n \ val}{(F,(f;x_1,...,x_n;c;e)),H,M \ \ x := f(e_1,...,e_i,...,e_n) \longmapsto F,H,M \ \ x := [F,H,(x_1 \hookrightarrow e_1,...,x_n \hookrightarrow e_n) \ \ (c;e)]}$$

$$\frac{F,H',M' \ \ c \longmapsto F,H'',M''c'}{F,H,M \ \ x := [F,H',M' \ \ (c;e)] \longmapsto F,H'',M \ \ x := [F,H'',M'' \ \ (c';e)]}$$

$$\frac{c \ cfinal \quad H,M' \ \ e \longmapsto H,M' \ \ e'}{F,H,M \ \ x := [F,H,M' \ \ (c;e)] \longmapsto F,H,M \ \ x := [F,H,M' \ \ (c;e')]}$$

$$\frac{c \ cfinal \quad e \ val}{F,H,M \ \ x := [F,H,M' \ \ (c;e)] \longmapsto F,H,M \ \ x := e}$$

$$\frac{H,M \ \ e \longmapsto H,M \ \ e'}{F,H,M \ \ *x := e \longmapsto F,H,M \ \ *x := e'}$$

$$\frac{e \ val}{F,(H,y \hookrightarrow \_),(M,x \hookrightarrow EPtr(y)) \ \ *x := e \longmapsto F,(H,y \hookrightarrow e),(M,x \hookrightarrow EPtr(y)) \ \ skip}$$

3.4 Program

$$\frac{}{p \longmapsto \emptyset,\emptyset,\emptyset \ \ p}$$

$$\frac{}{F,H,M \ \ newtype(\tau,t,p) \longmapsto F,H,M \ \ p}$$

$$\frac{}{F,H,M \ \ newfunc(f;x_1,...,x_n;c;e;p) \longmapsto (F,(f;x_1,...,x_n;c;e)),H,M \ \ p}$$

$$\frac{}{F,H,M \ \ pcommand(c) \longmapsto F,H,M \ \ c}$$

4 Bezpieczeństwo typów

Twierdzenie: Jeśli $p \ \ pwf$ i $p \mapsto p'$ to $p' \ \ pwf$

5 Postęp obliczeń

Twierdzenie: Jeśli $p \ \ pwf$ to $\exists p' \ \ p \mapsto p'$ lub $p \ \ pfinal$

6 Inne uwagi, które nie pasują nigdzie indziej

- Prezentowane reguły typowania i semantyki małych kroków odpowiada dokładnie implementacji.
- Ciąg deklaracji zmiennych i wskaźników (np. $vars(x := 1, y := 3, r := 3)in\{skip\}$ jest w czasie parsowania zamieniany na jedną, zagnieżdżoną instrukcję, tak jak w języku W - $(newvar(x,1,newvar(y,3,newvar(r,3,skip))))$
- Podobnie rozwiązany jest rozwiązany problem nowych funkcji i typów definiowanych przez użytkownika. W pliku $syntax.ml$ znajdują się wszystkie definicje typów.
- Definicja funkcji wymaga podania typu wszystkich argumentów.
- Stos wywołań funkcji w języku $W++$ jest symulowany stosem w języku $OCaml$ (metoda $processFunctionCall$ w pliku $eval.ml$).
- Nazwy zmiennej w ciele instrukcji nigdy nie może zostać przesłonięta.
- Indekser krotki musi zawsze być liczbą naturalną.
- Krotki są niemutowalne - aby zmienić jej jedną współrzędną, należy stworzyć nową.
- Zmienna typu wskaźnikowego w pamięci trzyma etykietę komórki pamięci na stercie.
- Raz zaalokowana wartość na stercie nie może zostać usunięta, co najwyżej można ją ponownie nadpisać.
- W case'ie wszystkie przypadki muszą zostać zdefiniowane.

- Typ abstrakcyjny nie musi być zdefiniowany tylko pod wskaźnikiem.
- Plik *Readme.md* zawiera informacje o komplikowaniu i uruchamianiu kodu.
- Funkcja może zwrócić wskaźnik.
- Jedyny sposób, by zadeklarować zmienną typu $TSum$ z więcej niż jednym konstruktorem, to wpisać cały typ w adnotacji typowej.
- Średnik po ostatniej instrukcji powoduje $SyntaxError$.
- Błędy typowania są w miarę dobrze opisane w ramach wyjątków. Jeśli program się otypował, to powinien się poprawnie wykonać (tj. zakończyć lub zapętlić).
- Katalog *examples/* zawiera przykładowe programy.