JĘZYK PROGRAMOWANIA ROBOTA PROGRAMOWEGO „CHRZĄSZCZ” DOKUMENTACJA DLA UŻYTKOWNIKA

1. Podstawowe informacje

„CHRZĄSZCZ” jest językiem programowania umożliwiającym obsługę prostego robota programowego. Program może być narzędziem do uczenia (w szczególności najmłodszych) niskopoziomowego programowania strukturalnego. Składnia języka jest w języku polskim, aby umożliwić młodszym dzieciom nieznającym jeszcze innych języków naukę. Dzięki wbudowanej planszy oraz konsoli można oglądać co dzieje się z robotem jednocześnie obserwując jakie instrukcje są za to odpowiedzialne.

2. Uruchamianie

Aby uruchomić symulację należy uruchomić w środowisku wykonawczym Javy 8 program Main z katalogu z klasami programu i jako argument wywołania podać ścieżkę do pliku *.chrz lub *.schrz (zapisany stan programu uprzednio wykonywanego).

3. Okno symulacji

Wewnątrz okienka pojawiającego się po rozpoczęciu symulacji widzimy planszę z polami w różnych kolorach. Biały oznacza puste pole, zielony pole z robotem, niebieski przedmiot, który można usunąć lub postawić przy pomocy robota, natomiast czerwony niemożliwą do przejścia ścianę. Przed uruchomieniem symulacji poprzez kliknięcie na którymś z pól można ustawić tam przedmiot, a po drugim kliknięciu pojawi się tam ściana (nie należy ustawiać przeszkód na polu z robotem), gdy chcemy utworzony przedmiot/ścianę usunąć wystarczy kliknąć trzeci raz. Na górze dodatkowo znajdują się dwa przyciski:

- **Rozpocznij symulację** – blokuje możliwość układania przeszkód na planszy (bez użycia robota), ustawia robota zwróconego na północ na polu (0,0) i rozpoczyna właściwe działanie uruchomionego programu.
- **Zapisz stan programu** – zapisuje w lokalizacji uruchomionego programu plik nazwaProgramu.schrz , który można później uruchomić, co spowoduje wykonywanie się programu od momentu zapisania.

4. Pisanie programu

Pisanie programu w języku „CHRZĄSZCZ” nie wymaga specjalnego edytora, może być pisany w dowolnym edytorze tekstowym.

5. Składnia

Język „CHRZĄSZCZ” jest czuły na wielkość liter. Nazwy procedur i zmiennych powinny być ciągami znaków zawierającymi litery,cyfry i znak _. Znaki białe są pomijane dlatego
{};
jest równoważne
{
};
co umożliwia pisanie według swoich preferencji formatowania kodu.

a) słowa kluczowe – nie należy ich używać podczas definiowania własnych procedur i zmiennych
procedura, jeżeli, dopóki, dopóty, powtórz, zmienna, powrót

b) Procedury
Każda nowa procedura ma składnię:

procedura nazwaProcedury
{
    kod procedury
}$

Każda procedura kończy się słowem kluczowym „powrót” i średnikiem, gdy interpreter napotka tą instrukcję wraca z procedury do miejsca jej wywołania. Główna procedura, od której zacznie się wykonywanie programu musi mieć nazwę „główna”. Nie można definiować procedur wewnątrz innej procedury. Aby wywołać procedurę wewnątrz programu wystarczy zapisać jej nazwę zakończoną średnikiem np.

procedura nieskonczonaRekurencja
{
    nieskonczonaRekurencja;
    powrót;
}$

c) Zmienne

Przed użyciem każdą zmienną należy zadeklarować, deklaracja ma następującą składnię:

zmienna nazwaZmiennej;

Po zadeklarowaniu zmienna przyjmuje automatycznie wartość 0. Aby w programie odwołać się do zmiennej wystarczy napisać jej nazwę np.

zmienna = zmienna + 1;

Zmienne mogą mieć wartości całkowitoliczbowe. Na zmiennych można wykonywać operacje matematyczno-logiczne oraz przypisywać im wartości. Wszystkie zmienne w „CHRZĄSZCZ-u" są globalne, zatem można je wykorzystać m.in. do zwracania wartości z procedur tworząc tak jakby funkcje np.

```
procedura główna
{
        zmienna a;
        a = 5;
        inkrementujA;
        powrót;
}$
procedura inkrementujA
{
        a = a + 1;
        powrót;
}$
```

e) Operatory matematyczno-logiczne

Język „CHRZĄSZCZ" pozwala na wykonywanie prostych obliczeń, służą do tego następujące operatory:

=	przypisuje zmiennej po lewej stronie wartość zmiennej lub stałej z prawej strony, zawsze zwraca wartość 1
+	zwraca wynik dodawania
-	zwraca wynik odejmowania
*	zwraca wynik mnożenia
/	zwraca wynik dzielenia całkowitoliczbowego
==	zwraca 1 gdy elementy są równe, w przeciwnym wypadku 0
!=	zwraca 0 gdy elementy są równe, w przeciwnym wypadku 1
> i >=	zwraca 1 gdy element po lewej stronie jest większy/większy lub równy, w przeciwnym wypadku 0
< i <=	zwraca 1 gdy element po lewej stronie jest mniejszy/mniejszy lub równy, w przeciwnym wypadku 0

=! przypisuje zmiennej wartość odwrotną (0 gdy zmienna !=0 , 1 gdy zmienna ==1), dodatkowo zwraca przypisywaną wartość

Operacje wykonuje się w następujący sposób:
np. *a =! a + b;* zapisze do zmiennej a jedynkę gdy wynik dodawania a i b jest zerem, lub zero, gdy nim nie będzie.
Operacje matematyczne wykonywane są kolejno od prawej strony, nie można używać nawiasów, dlatego jeśli chcemy wykonać bardziej skomplikowane działanie należy użyć do tego dodatkowych zmiennych np. zamiast

*a = b + ((d > c) * c);*

piszemy następujący ciąg instrukcji:

zmienna tmp;
tmp = d > c;
*tmp = tmp * c;*
a = b + tmp;

d) Komentarze
Aby część kodu wyłączyć z interpretacji wystarczy ją objąć znakami „#”. To co znajduje się pomiędzy dwoma takimi zostanie pominięte np.

procedura główna # { powrót; jakiś losowy tekst } #
{ powrót; }$

e) Wbudowane komendy
Robot posiada kilka wbudowanych komend, z których można korzystać tak jak z innych procedur z tą różnicą, że nie musimy ich definiować. Dostępne mamy komendy:

- *idź* – porusza robota o jedno miejsce do przodu

- *skręćWPrawo* – zmienia kierunek przodu robota w prawo w stosunku do obecnego

- *skręćWLewo* – analogicznie w lewo

- *połóżCegłę* – kładzie Cegłę (przedmiot) na polu na którym się znajduje

- *zabierzCegłę* – usuwa Cegłę (przedmiot) z pola na którym robot się znajduje

- *sprawdźOdległość* – aktualizuje wartość zmiennej „odległość” (jest ona automatycznie zdefiniowana, nie musimy tego robić), tak aby znalazła się w niej ilość pól do najbliższej przeszkody (lub końca planszy jeśli nie ma przeszkód) w

kierunku przodu robota. Zmiennej „odległość” możemy używać jak zwykłej zmiennej.

- wypisz – wypisuje na konsolę wartość zmiennej „out”. Zmienną taką musimy wcześniej utworzyć. W przypadku braku zmiennej „out” wypisany zostanie stosowny komunikat.

f) Instrukcje warunkowe
Instrukcja warunkowa ma składnię:

```
jeżeli( warunek )
{
        kod który wykona się gdy warunek jest prawdziwy
};
```

Warunek może być liczbą lub zmienną, 0 jest traktowane jako fałsz, każda inna wartość jako prawda.

e) Pętla z warunkiem
składnia:

```
dopóki ( warunek ) dopóty
{
        kod wykonywany dopóty dopóki warunek jest spełniony
};
```

f) Pętla iteracyjna
Aby wykonać dany blok instrukcji n razy, należy zapisać

```
powtórz ( n )
{
        instrukcje;
};
```

gdzie n jest liczbą całkowitą.

6. Przykładowy program
Robot jedzie pięć pól na wschód, następnie skręca i jedzie tyle samo pól na południe i kładzie tam cegłę.

```
procedura główna
{
```

```
        zmienna a;
        zmienna b;
        b = 1;

        skręćWPrawo;
        powtórz(5)
        {
                idź;
                inkrementujA;
        };
        skręćWPrawo;
        dopóki ( a > 0 ) dopóty
        {
                idź;
                a = a – b;
        };
        połóżCegłę;
        powrót;

}$

procedura inkrementujA{ a = a + 1; powrót; }$
```