Pamiętam moje wczesne próby na Atari z Kyan Pascalem. Była to trudna znajomość. Częściowo ze względu na nieprzyjazny edytor, a częściowo mój młody wiek i brak instrukcji, bo dyskietka była oczywiście “niezupełnie oryginalna”. Potem miałem trochę przygód z Turbo Pascalem Borlanda, ale to już na PC. Troszkę też poobijałem się o Delphi, pisząc kiedyś ambitny program do obsługi stanów magazynowych, ale na tym moja przygoda z Pascalem się skończyła, a było to ponad 15 lat temu. I pewnie tak by pozostało do dziś, gdybym nie mój powrót do kodowania na Atari i odkrycie Mad-Pascala. To naprawdę świetny kompilator Pascala na małe Atari, autorstwa Tebego. Udało mi się już napisać w nim kilka programów i postanowiłem podzielić się z Wami moimi wrażeniami i zdobytą wiedzą. Może ktoś jeszcze pochyli się wraz ze mną nad klawiaturą i popełni jakiś ciekawy kawałek kodu?

Na początek kilka słów o samym Pascalu. Pomimo że dziś może wydawać się nam już językiem niemal archaicznym - pojawił się dosyć późno, bo dopiero na początku lat siedemdziesiątych dwudziestego wieku. Składniowo przypominający nieco język Algol, Pascal od początku był projektowany jako język nieco wyższego poziomu niż C. Miał umożliwiać proste operowanie złożonymi strukturami danych i wyższy poziom abstrakcji, co poniekąd mu się udaje. Jednocześnie nadal pozwala zejść nieco “na ziemię” i dopuszcza operacje na wskaźnikach, bezpośrednie odwołania do pamięci oraz nawet wstawki w asemblerze.

Ważną cechą Pascala jest to, że podobnie jak C jest językiem silnie typowanym. Oznacza to, że każda zmienna, której użyjemy w programie, musi mieć jasno określony typ. To wymusza pewną dyscyplinę, ale również pozwala często uniknąć błędów. Przykładowo próba przypisania wartości liczbowej do zmiennej tekstowej typu *string* zakończy się błędem już na poziomie kompilacji programu - czyli jeszcze zanim w ogóle spróbujemy go uruchomić.

Kolejną ważną cechą Pascala jest to, że umożliwia podział kodu na procedury i funkcje, oraz grupowanie ich w większe biblioteki, nazywane po polsku modułami (a z angielska units). Kilka takich gotowych modułów dostajemy do dyspozycji już na starcie, co daje nam sporą garść narzędzi, ale to zobaczycie sami, jak już przejdziemy do zasadniczej części kursu.

Mad-Pascal o którym tu będzie mowa, jest kompilatorem skrośnym. Kompilatory skrośne (*cross-compilers*) to programy zdolne do generowania kodu maszynowego na architekturę procesora inną niż ta na której są uruchomione. A co to dla nas oznacza? Dokładnie to, że kompilator nie jest uruchamiany na Atari, tylko na PC, ale “wypluwa” nam plik wykonywalny w postaci pliku binarnego xex (lub obx). Taki plik możemy sobie następnie przerzucić na Atari, lub wykonać jeszcze na PC w ulubionym emulatorze.

Puryści pewnie kręcą nosem i zgrzytają zębami, ale takie rozwiązanie ma wiele zalet. Niech wymienię tylko kilka z nich:

- środowisko programistyczne uruchomione na Atari zabiera nam cenny RAM
- kompilacja na PC jest baaaardzo szybka
- możemy błyskawicznie uruchomić skompilowany program na emulatorze
- możemy dobrać sobie dowolny/ulubiony edytor i korzystać z jego udogodnień

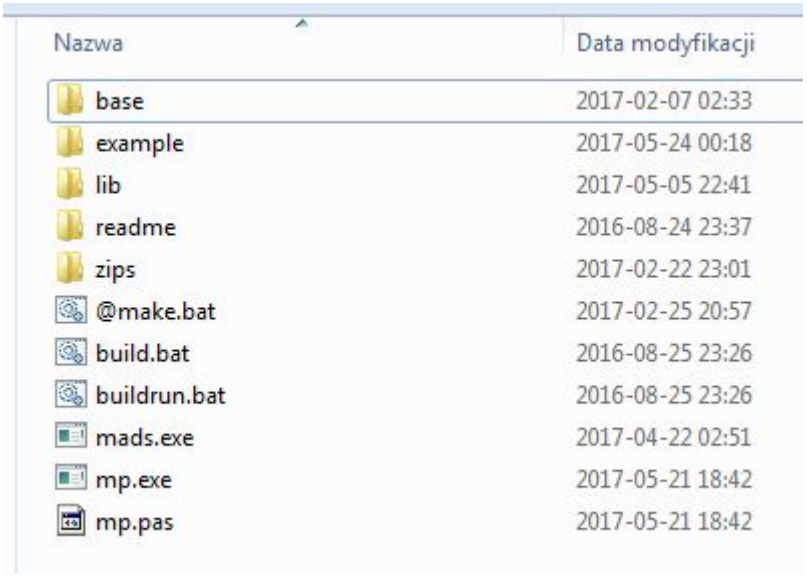
- mamy multitasking i w drugim okienku możemy uruchomić sobie niniejszy poradnik
- programując na PC oszczędzamy klawiaturę naszego Atari!

Kolejną cechą Mad-Pascala o której warto wspomnieć jest fakt, że kompilacja przebiega dwuetapowo. W pierwszym kroku kod jest optymalizowany i tłumaczony na plik asemblera składniowo zgodny z crossassemblerem MADS - również autorstwa Tebego. W drugim kroku plik ten jest asemblowany do kodu maszynowego właśnie za pomocą MADSa. Umożliwia to dodatkowo przeróżnym dziwnym zapaleńcom przeprowadzanie optymalizacji na kodzie “przejściowym”, lub głębsze analizowanie oraz debugowanie napotkanych problemów.

Ale przejdźmy w końcu do konkretów. W pierwszej części tego poradnika postanowiłem, że prócz powyższego przydługiego wstępu, pokażę jak skonfigurować sobie przykładowe środowisko pracy i jak skompilować nasz pierwszy program w Mad-Pascalu. Zatem do dzieła!

Przede wszystkim musimy pobrać sobie i skonfigurować nasz kompilator. Odsyłam do strony autora gdzie znajdziemy pliki binarne dla platformy Windows w najnowszych wersjach. W chwili pisania tego tekstu jest to wersja 1.4.7. Jeżeli akurat nie pracujemy na systemach z Redmond, a jesteśmy dumnymi posiadaczami linuksa, to zmuszeni jesteśmy własnoręcznie skompilować sobie pliki wykonywalne dla naszego systemu. Nie jest to trudne, na stronie autora znajdziemy dokładną instrukcję jak to zrobić.

Zakładam, że mamy już pobrane pliki ze strony oraz plik wykonywalny odpowiedni dla naszej platformy. Umieszczamy sobie powyższe pliki w jakimś ulubionym miejscu. U mnie jest to katalog: e:\atari\MadPascal i jego zawartość wygląda tak:



Pewnie zauważyliście dwa dodatkowe pliki których nie ma w paczce ze strony, build.bat i buildrun.bat?

Przygotowałem je na bazie pliku @make.bat, ale uprzednio dopasowałem pod mój system. Zaraz pomogę Wam “skroić” podobne pliki pod Wasz komputer, co przyda się wam w dalszej części tego poradnika.

Najpierw zaglądniemy do pliku build.bat:

```
@setlocal
@set PATH=%PATH%;e:\atari\MadPascal
mp.exe %1.pas -o
mads.exe %1.a65 -x -i:e:\atari\MadPascal\base -o:%1.xex
pause
```

Polecenie @setlocal w pierwszej linii nie jest konieczne, ale powoduje że wszystkie zmienne środowiskowe ustawione w tym pliku będą działały tylko lokalnie (nie zanieczyszcza globalnych zmiennych środowiskowych).

Druga linia powoduje dodanie do głównej ścieżki wyszukiwania systemu naszego katalogu z kompilatorem i modułami. Pozwala nam to (w obrębie tego pliku) wywoływać kompilator i asembler bez podawania ich pełnej ścieżki, niezależnie w którym katalogu nastąpi wywołanie pliku build.bat

Linia 3 to już wywołanie naszego kompilatora mp.exe gdzie %1 to pierwszy parametr wywołania pliku build.bat a opcja -o powoduje wykonanie dodatkowej optymalizacji generującej krótszy i szybszy kod dla 6502. Warto zauważyć, że w wywołaniu doklejamy “ręcznie” domyślne rozszerzenie .pas do nazwy kompilowanego pliku. Czyli prawidłowe wywołanie do skompilowania pliku **program.pas** będzie wyglądało:

```
build.bat program
```

rozszerzenie .pas zostanie doklejone automatycznie. Zdecydowałem się na takie rozwiązanie, bo umożliwia to późniejszą łatwą integrację z wybranym przeze mnie edytorem, ale o tym za chwilę.
Wynikiem wywołania linii trzeciej będzie utworzenie pliku **program.a65** zawierającego skompilowany i zoptymalizowany kod naszego programu w asemblerze.

Linia 4 to wywołanie asemblera, który przekonwertuje nasz plik program.a65 na binarny plik wykonywalny gotowy do uruchomienia. Opcja -x usuwa z kodu nieużywane procedury, -i wskazywać musi na lokalizację katalogu “*base”* Mad-Pascala zawierającego potrzebne do asemblacji biblioteki. Parametr -o powoduje, że nasz plik uruchamialny otrzyma rozszerzenie .xex. Zatem wynikiem wykonania linii czwartej będzie gotowy do uruchomienia plik wykonywalny: **program.xex**

Linia 5 powoduje oczekiwanie na wciśnięcie dowolnego klawisza. Nie jest konieczna, jednak dla swojej wygody dodałem ją, aby móc spokojnie podglądnąć komunikaty kompilatora i asemblera. Jest to szczególnie przydatne w wypadku wystąpienia jakichś błędów.

A teraz plik buildrun.bat, który różni się od poprzedniego tylko ostatnią linią:

```
@setlocal
@set PATH=%PATH%;e:\atari\MadPascal
mp.exe %1.pas -o
mads.exe %1.a65 -x -i:e:\atari\MadPascal\base -o:%1.xex
e:\atari\Altirra\altirra64.exe %1.xex
```

Po kompilacji zamiast oczekiwania na wciśnięcie klawisza wywoływany jest emulator, który uruchamia nasz wynikowy plik. Tutaj oczywiście możemy wstawić inny ulubiony emulator, odpowiednio dopasowując ścieżkę.

Analogiczne pliki dla systemu linuks (ubuntu) wyglądają u mnie tak:

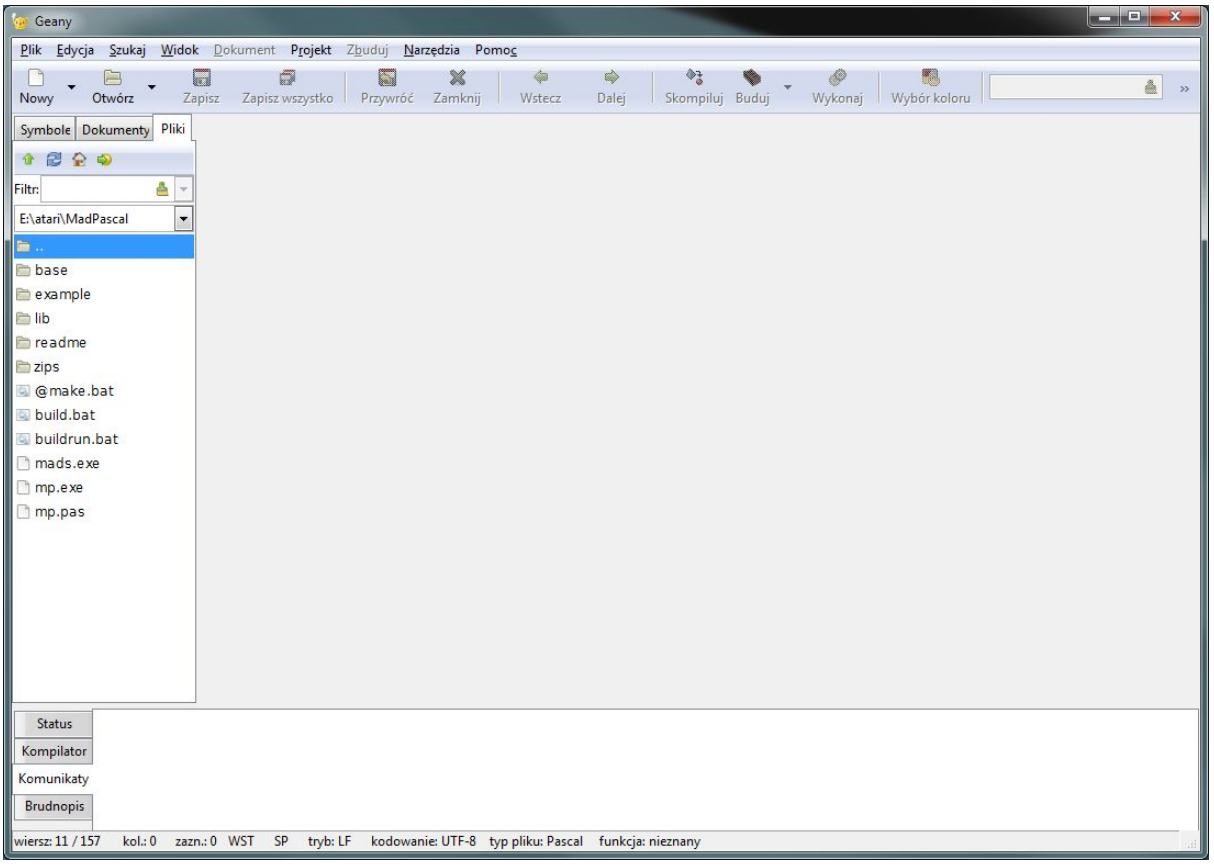
build.sh

```
#!/bin/bash
export PATH="/home/bocianu/madPascal:$PATH"
mp $1.pas -o
mads $1.a65 -x -i:/home/bocianu/madPascal/base -o:$1.xex
```

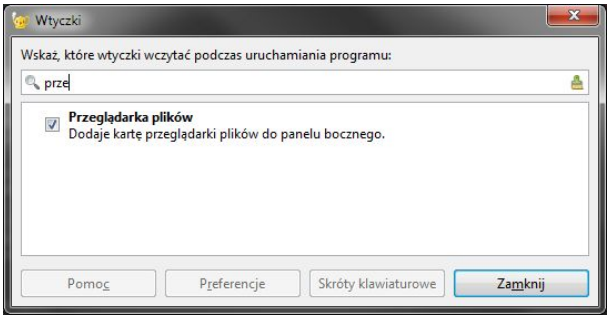
buildrun.sh

```
#!/bin/bash
export PATH="/home/bocianu/madPascal:$PATH"
mp $1.pas -o
mads $1.a65 -x -i:/home/bocianu/madPascal/base -o:$1.xex
atari800 $1.xex
```

Mając gotowe powyższe pliki możemy śmiało zabrać się za przygotowanie edytora. Ja polecam i opiszę jak skonfigurować edytor geany, którego używam z powodzeniem od dłuższego czasu i na windowsie i pod linuksem. Oto on:



Edytor jest darmowy, i można go pobrać ze strony producenta: https://www.geany.org/. Znajduje się także w ogólnodostępnych repozytoriach dla większości dystrybucji linuksa. Geany ma domyślnie podświetlanie składni Pascala i wielu innych języków, więc nie wymaga na starcie prawie żadnych dodatkowych zabiegów, aby działać jak należy. Jedynym dodatkiem który się bardzo przydaje, jest przeglądarka plików w oknie bocznym, którą widać na powyższym obrazku. Możemy ją w prosty sposób dodać z menu Narzędzia -> Menedżer Wtyczek

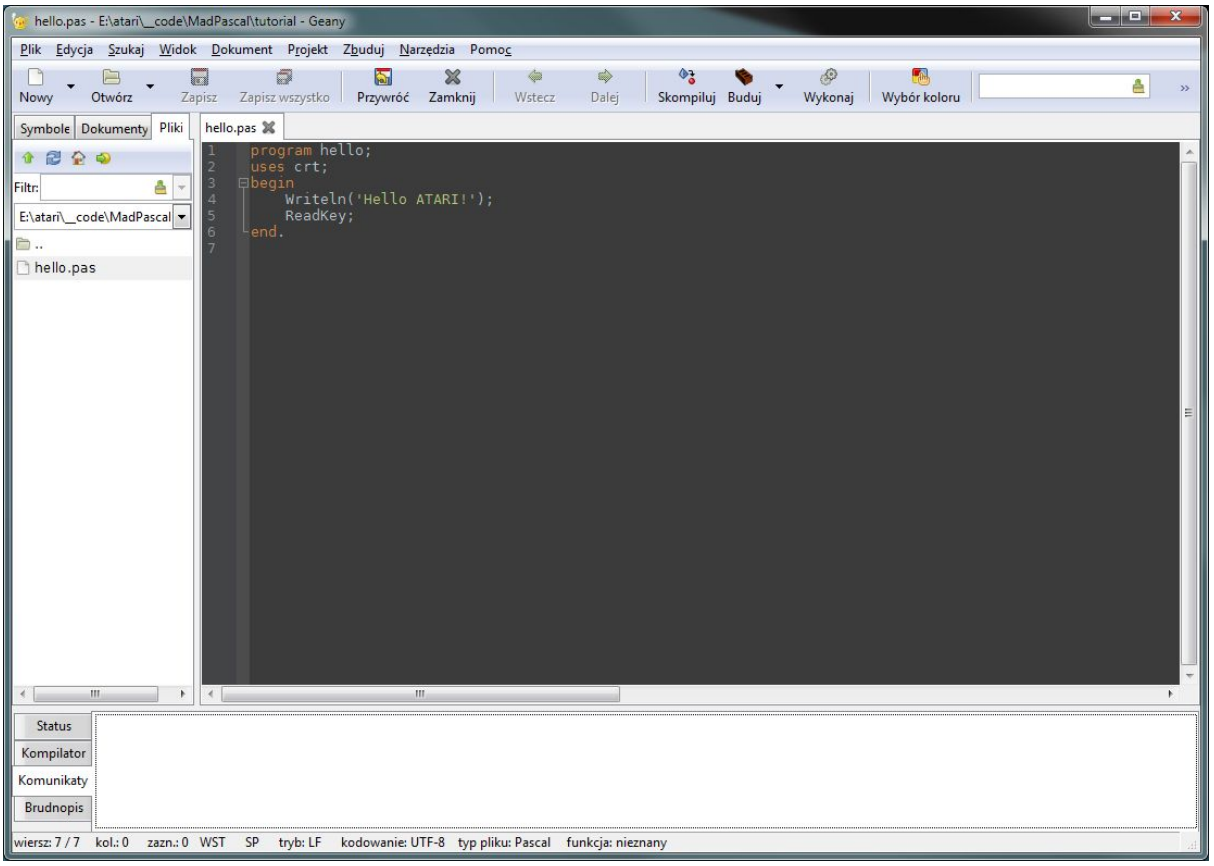


I w zasadzie nie potrzebujemy już nic więcej aby napisać swój pierwszy program w Pascalu, zatem do dzieła! Utwórz nowy plik naciskając przycisk menu lub Ctrl+n. Następnie wprowadź (lub opcja dla leni - przeklej) następujący kod:

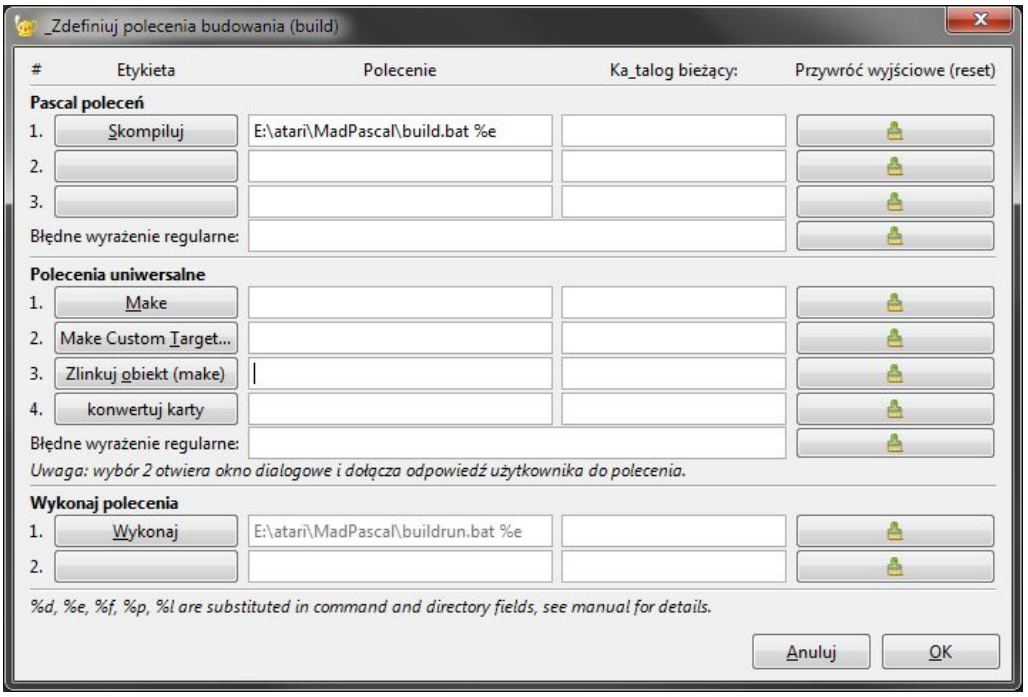
```
program hello;
uses crt;
```

```
begin
    Writeln('Hello ATARI!');
    ReadKey;
end.
```

Teraz zapisz dzieło w jakimś wybranym katalogu (Ctrl+s) pod dowolną nazwą pamiętając o obowiązkowym rozszerzeniu .pas. Na przykład niech będzie to **hello.pas**
U mnie wygląda to tak:

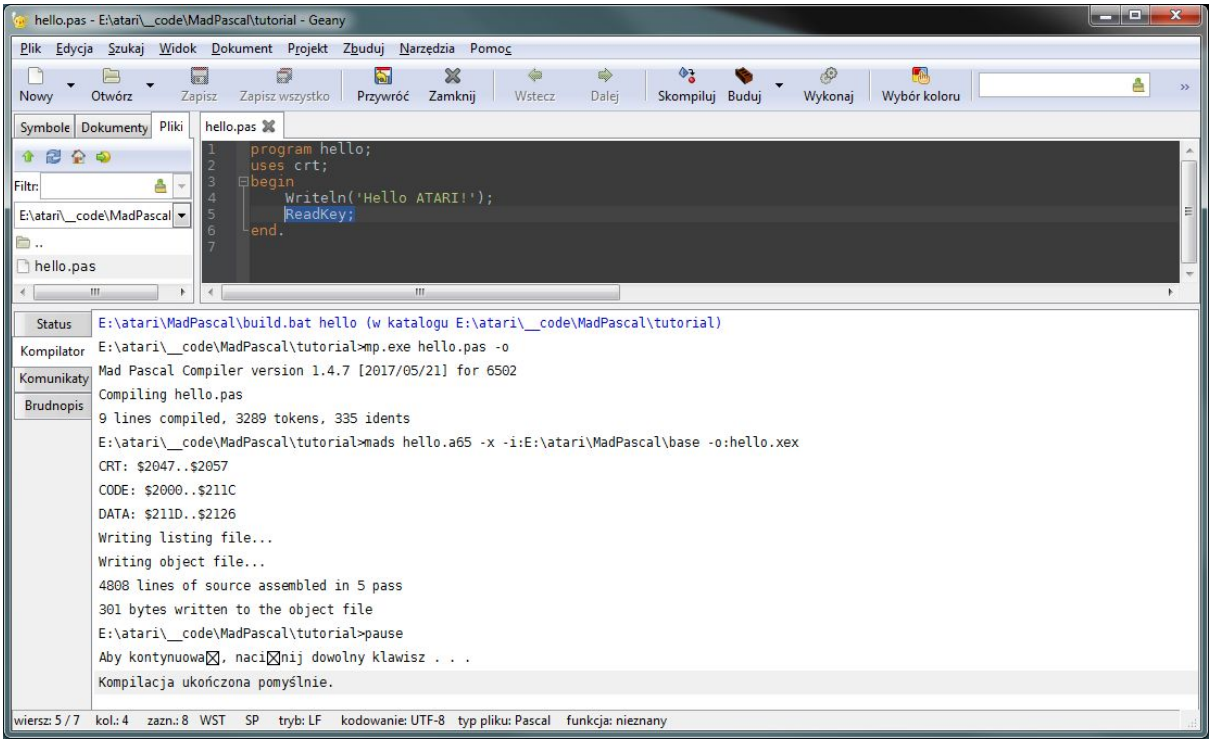


Pewnie już zauważyliście przyciski Skompiluj i Wykonaj? Jak nie, to nietrudno je znaleźć w menu. Prowadzą też do nich łatwe skróty klawiaturowe: Skompiluj - F8, Wykonaj - F5. Polecam zapamiętać - przyda się na przyszłość. Jednak jeżeli już odważyliście się je nacisnąć, to pewnie zauważyliście również, że nic specjalnego się nie stało. Ewentualnie dostaliście w oknie kompilatora komunikaty o błędach. I tutaj przychodzą nam z pomocą przygotowane wcześniej pliki build.bat (skompiluj) buildrun.bat (wykonaj). Wystarczy, że podłączymy je jako akcje do naszych przycisków. A robimy to bardzo łatwo wchodząc w menu Zbuduj -> Zdefiniuj polecenia budowania:

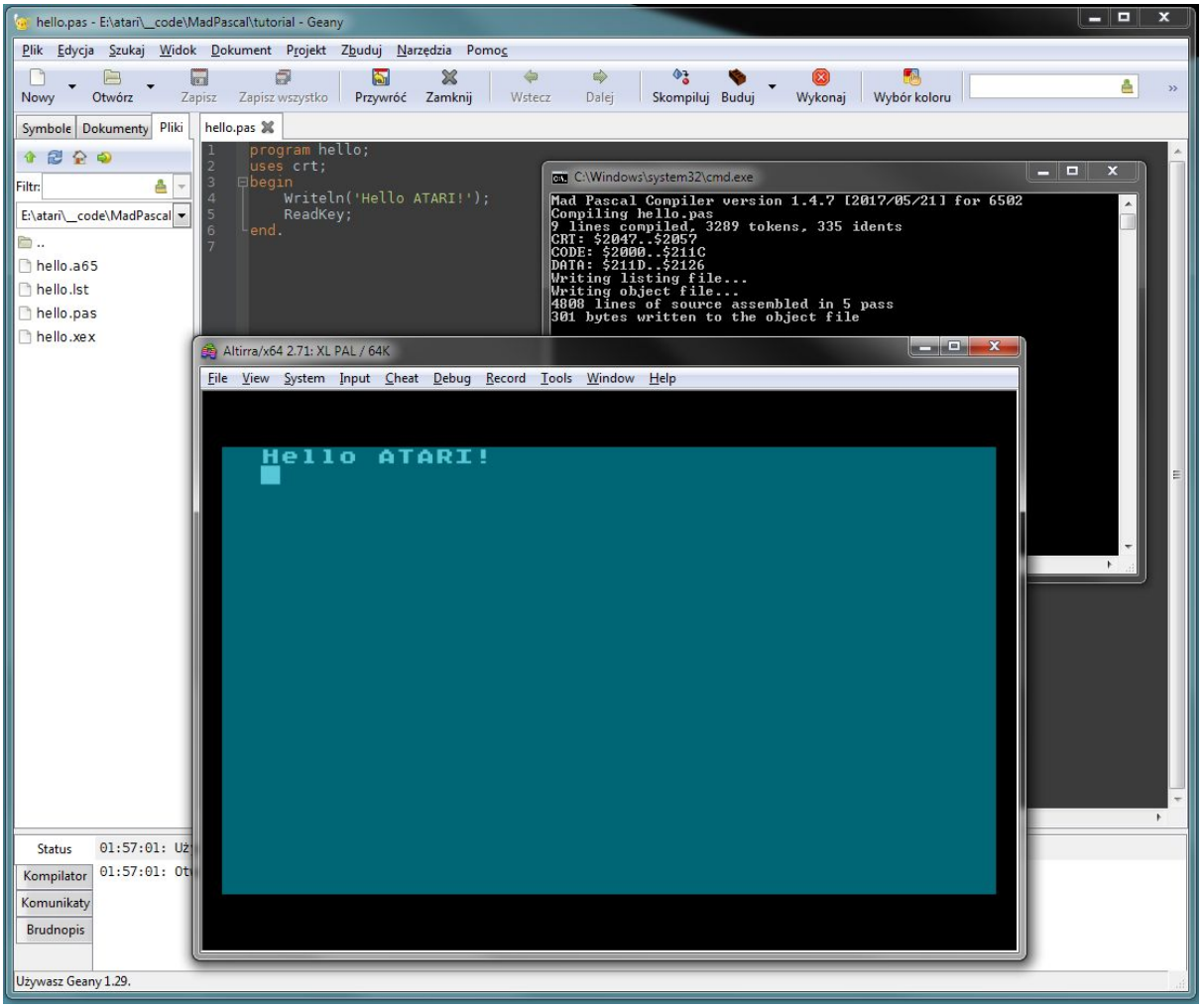


Wpisujemy u siebie w odpowiednie pola (modyfikując odpowiednio ścieżkę) powyższe polecenia. Wartość %e zostanie przez edytor podmieniona na nazwę pliku bez rozszerzenia, czyli tak jak potrzebujemy. Pozostałe pola mogą pozostać puste. Naciskamy OK i gotowe!

Spróbujmy skompilować nasz program naciskając F8. Jeżeli wszystko dobrze skonfigurowaliśmy to, powinniśmy zobaczyć w dolnej zakładce Kompilator przelatujące szybko komunikaty wyglądające mniej więcej tak:



Kompilacja ukończona pomyślnie! Brawo. Pozostało tylko uruchomić nasz program. Naciskamy F5.



Tadam! Napisaliśmy i uruchomiliśmy nasz pierwszy program w Mad-Pascalu! Jeżeli coś poszło nie tak, to sprawdź, czy prawidłowo podałeś ścieżki do katalogu MadPascala i emulatora w plikach .bat, bo to najczęstszy problem.

Teraz masz gotowe środowisko i na tym zakończymy pierwszy rozdział poradnika. W następnej części wytłumaczę dlaczego nasz pierwszy program wygląda tak jak wygląda i spróbujemy napisać coś bardziej wymyślnego. A tymczasem w oczekiwaniu na kolejną część możesz sobie spróbować skompilować i uruchomić przykłady z katalogu MadPascal/examples. Jest tam sporo ciekawych programów i przykładów użycia domyślnych bibliotek. Masz wszystko podane na tacy, tylko otworzyć plik *.pas w Geany i nacisnąć F5 ;)

Miłej zabawy.