24.04.2024 17:49:53

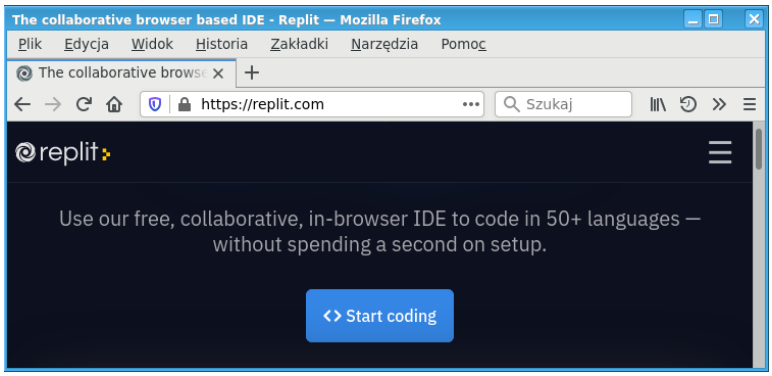
Algorytmy i Struktury Danych

Laboratorium
Stos i kolejka

Przygotowanie do wykonania zadania.

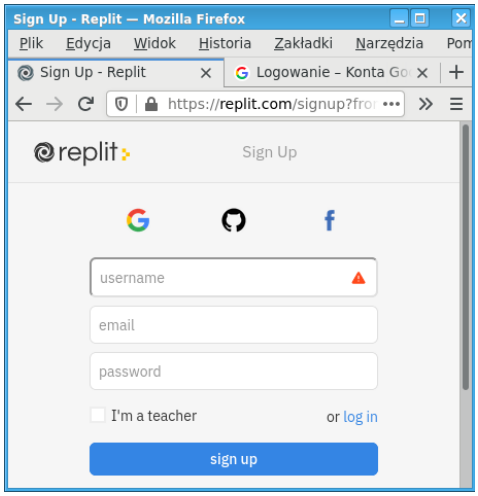
Rozpakuj archiwum z przykładowym projektem.

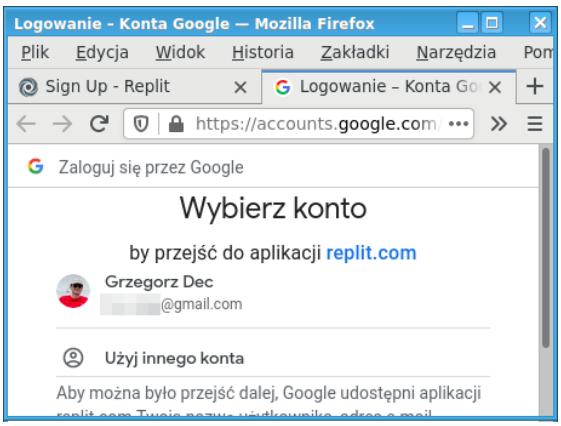
Wejdź na stronę `https://repl.it/`. Kliknij *Start coding*.



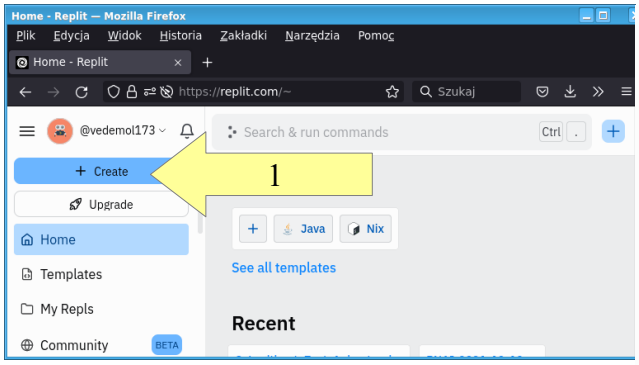
Zaloguj się kontem google albo facebook.

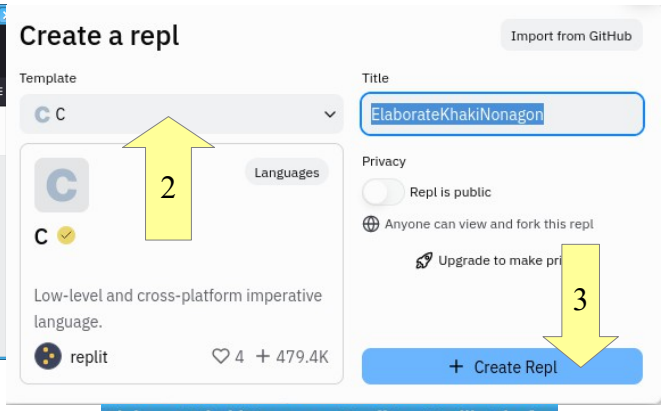
Możesz również utworzyć sobie tymczasowy e-mail (np. na stronie https://pl.emailfake.com/) i użyć go do zarejestrowania się na repl.it.



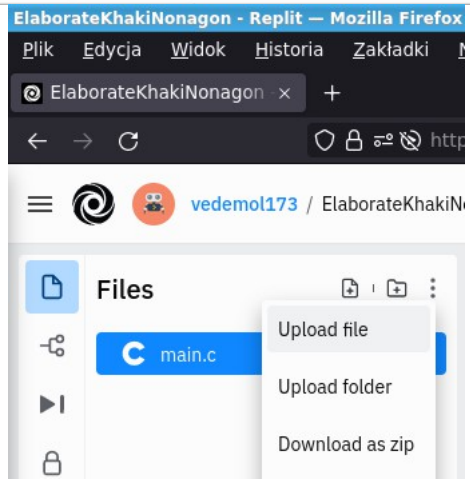


Utwórz nowy projekt w języku C.

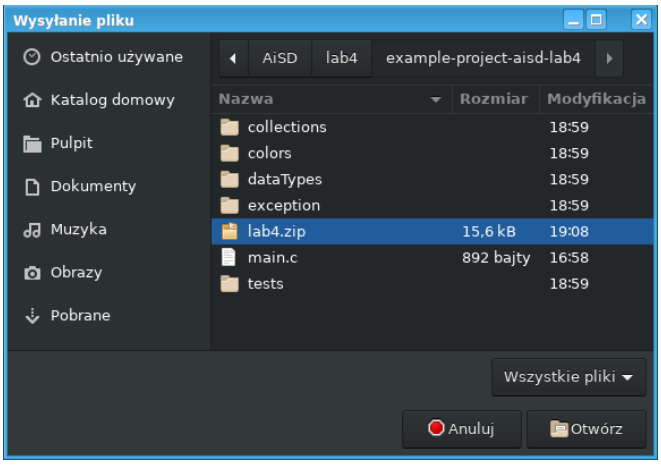




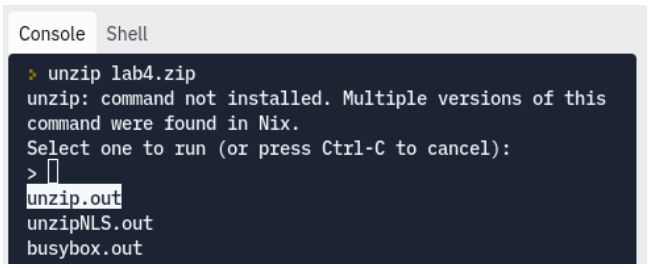
Kliknij w symbol ⋮ i wybierz *Upload file*.



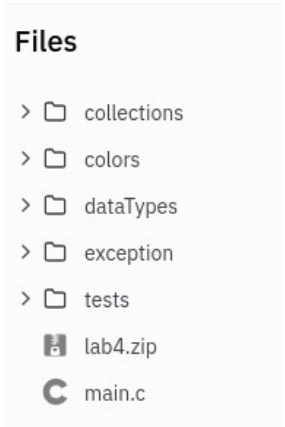
Wybierz plik `lab4.zip` z folderu, który zawiera rozpakowany przykładowy projekt.



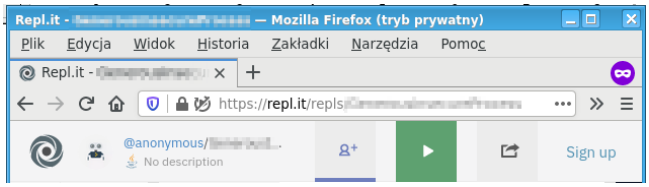
W oknie konsoli napisz polecenie:
`unzip lab4.zip`
Wybierz instalowanie programu `unzip.out` i naciśnij enter.
Jeżeli `unzip` zapyta się o zastąpienie plików, wybierz All.



Struktura projektu powinna być taka, jak pokazano obok.



Teraz uruchom projekt – kliknij zielony przycisk Run



U mnie działa. U ciebie też powinno działać:

```
 ==============  Testing stack ==============
--------- stack content ---------
     empty
-----------------
Testing stack underflow ...  FATAL ERROR: stack pop function not implemented
Testing push == pop ...  FATAL ERROR: pop or push not implemented
Testing push1, push2 == pop2, pop1 ...  FATAL ERROR: pop or push not implemented
Testing if stack is empty ...  Ok
 FATAL ERROR: pop or push not implemented
Testing stack overflow ...  FATAL ERROR: stack pop function not implemented
 ==============  Testing queue ==============
--------- queue content ---------
     empty
-----------------
Testing queue underflow ...  FATAL ERROR: queue dequeue function not implemented
Testing enqueue == dequeue ...  FATAL ERROR: queue enqueue or dequeue function not
implemented
Testing ENQ1, ENQ2 == ENQ1, ENQ2 ...  FATAL ERROR: queue enqueue or dequeue functio
n not implemented
Testing if queue is empty ...  Ok
 FATAL ERROR: queue enqueue function not implemented
Testing queue overflow ...  FATAL ERROR: queue enqueue function not implemented
```

Struktura programu

Wczytany program zawiera następujące elementy:

- w folderze `dataTypes` deklarację typu `Finger` – element zapisywany na stosie i w kolejce,
- w pliku `collections/Stack.h` deklarację typu stosu, prototypy funkcji operacji na stosie,
- w pliku `collections/Queue.h` deklarację typu kolejki, prototypy funkcji operacji na kolejce,
- w plikach źródłowych `Stack.c` i `Queue.c` w folderze `collections` wstępny kod funkcji, które należy samodzielnie napisać.

Stos

Wykonaj następujące zadanie:

1. W skrypcie do AiSD w sekcji 2.3.1 są podane algorytmy K2.4, K2.5. Zaimplementuj te algorytmy – napisz funkcje `push`, `pop` w pliku `collections/Stack.c`.

Kolejka

Wykonaj następujące zadanie:

1. W skrypcie do AiSD w sekcji 2.3.2 są podane algorytmy K2.7, K2.8. Zaimplementuj te algorytmy – napisz funkcje `enqueue`, `dequeue` w pliku `collections/Queue.c`.

Podpowiedzi

Zwracanie błędów w języku C

Załóżmy, że mamy funkcję o następującej deklaracji:

```
ZwracanyTyp funkcja(TypParametru parametr);
```

Musimy przyjąć takie założenie, że wykonywanie funkcji nie powiedzie się i konieczne będzie zwrócenie informacji o błędzie. Historyczny sposób zwracania błędów z funkcji jest taki:

```
TypKoduBłędu funkcja(TypParametru parametr, ZwracanyTyp * zwracanaWartość);
```

Tak się programowało w starożytnych językach programowania na starożytnych komputerach. Współczesny sposób zwracania błędów to użycie wyjątków. W języku C nie ma wyjątków, ale opracowano pewne standardy, z których można skorzystać. W tym ćwiczeniu korzystamy z kodu udostępnionego przez anonimowego programistę. Zasady są następujące:

W pliku nagłówkowym deklarujemy wyjątek:

```
EXC_DECLARE(stack_overflow);
```

W pliku źródłowym definiujemy wyjątek:

```
EXC_DEFINE(stack_overflow, EXC_NUM_USER_DEFINED + 2, "stack overflow");
```

Jeżeli chcę zwrócić z funkcji komunikat błędu:

```
if(stos_się_przepełnił == true) {  EXC_THROW(stack_overflow);  }
```

W programie, który wywołuje funkcję:

```
EXC_TRY {
       y1 = funckja1(parametr1);
       y2 = funckja2(parametr2);
       y3 = funckja2(parametr2);
} EXC_CATCH(wyjątek1) {
       // obsługa wyjątku  wyjątek1
} EXC_CATCH(wyjątek1) {
       // obsługa wyjątku  wyjątek2
}
EXC_END;
```

Stos – pop

Struktura algorytmu K.2.5 i odpowiadająca mu struktura programu w języku C są następujące:

Algorytm	Program
POP(S)	`pop(S)`
Jeśli stos-pusty(s)	`if(stos jest pusty) {`
to błąd „niedomiar”	`rzuć wyjątek stack underflow`
inaczej	`} else {`
top[S] ← top[S]-1	`Zmniejsz wierzchołek o 1`
zwróć S[top[S]+1]	`return element nad wierzchołkiem;`
	`}`

Typy poszczególnych zmiennych w algorytmie i typy w programie:

Algorytm		Program	
Zmienna	**Typ**	**Zmienna**	**Typ**
S	Stos	`S`	`Stack`
S[top[S]+1]	Struktura danych	`element`	`Finger`
Błąd „niedomiar”	wyjątek	`wyjątek`	`stack underflow`

Program ze zidentyfikowanymi typami:

```
Finger pop(Stack S)
  // kod tej funkcji napisany bez wskaźników
  // jest bezsensowny. Dlatego go nie ma.

  return element;
}
```

Przeanalizujmy tę funkcję.

1. Gdybyśmy przekazali do funkcji `pop` parametr `S` przez wartość, to funkcja otrzymałaby kopię stosu. Funkcja `pop` musi zmodyfikować wierzchołek w oryginalnym stosie, a nie w kopii. Dlatego przekazywanie kopii w tym przypadku jest co najmniej błędem. Optymalne jest przekazanie wskaźnika do struktury stosu.
2. Z funkcji `pop` zwracamy element stosu. Przy pokazanej wyżej deklaracji zostanie zwrócona kopia elementu. Zwracanie kopii nie ma uzasadnienia logicznego. Wystarczy, że zwrócimy informację o położeniu elementu na stosie (wskaźnik do elementu). Klient funkcji `pop` zrobi sobie kopię, jeżeli będzie taka potrzeba.

Uwzględniając powyższe rozważania, funkcja przyjmie postać:

```
Finger * pop(Stack * S)
  if(isStackEmpty(S){
    EXC_THROW(stack_underflow);
  } else {
    S->top--;
    return & S->buffer[S->top+1];
  }
}
```

Program z normalnymi nazwami zmiennych:

```
Finger * pop(Stack * stack)
  if(isStackEmpty(stack){
    EXC_THROW(stack_underflow);
  } else {
    stack->top--;
    return & stack->buffer[stack->top+1];
  }
}
```

Stos – pop – zabezpieczenia przed błędnym działaniem

Podane niżej fragmenty programów są przykładami pokazującymi, w jaki sposób spowodować nieprawidłowe działanie programu i uzyskać nieautoryzowany dostęp do systemu. Prezentowany kod jest poprawny składniowo i jeżeli intencją programisty jest wyrządzenie szkód, to jest również poprawny semantycznie.

Dostęp do pamięci poza buforem stosu

W języku C nie ma możliwości ukrycia pola struktury. Można więc ustawić pole `top` stosu przed wywołaniem funkcji `pop` lub `push`. W pokazanym przykładzie uzyskamy dostęp do haseł i tajnych danych (funkcja `push` zmodyfikowałaby hasła i tajne dane). Zastanów się, jak zabezpieczyć swoje funkcje `pop` i `push` przed takim atakiem.

```
UserPassword passwords[10];
Stack stack;
SecretInformation secrets[10];
Finger * element;

stack.top = -5;
element = pop(& stack);
stack.top = STACK_CAPACITY+3;
element = pop(& stack);
```

Nieprawidłowa struktura danych przekazana jako parametr

W języku C można rzutować typy danych. Da się więc wywołać funkcję `pop` lub `push` w taki sposób, że przekażemy jej wskaźnik nie do stosu, ale do jakiejś innej struktury. W pokazanym przykładzie program zrobi coś dziwnego. Prawdopodobnie zostanie przerwany przez system operacyjny albo funkcja `pop` zwróci wskaźnik do obszaru pamięci zawierający bliżej nieokreślone dane.

Zaproponuj zabezpieczenie funkcji `pop` przed takim błędem (da się to zrobić).

```
DoubleLinkedList list;
Finger * element;

element = pop((Stack *)& list);
```

Stos - push

Struktura algorytmu K.2.4 i odpowiadająca mu struktura programu w języku C są następujące:

Algorytm	Program
PUSH(S, x)	`push(S, x)`
Czy stos się przepełnił?	`if(stos przepełnony) { rzuć wyjątek stack overflow }`
top[S] ← top[S]+1	`zwiększ wierzchołek o 1`
S[top[S]] ← x	`Zapisz x na stos`
	`}`

Typy poszczególnych zmiennych w algorytmie i typy w programie:

Algorytm		Program	
Zmienna	**Typ**	**Zmienna**	**Typ**
S	Stos	`S`	`Stack`
x	Struktura danych	`x`	`Finger`
błąd	Przepełnienie stosu	`wyjątek`	`stack overflow`

Program ze zidentyfikowanymi typami:

```
void push(Stack S, Finger x) {
  // kod tej funkcji napisany bez wskaźników
  // jest bezsensowny. Dlatego go nie ma.
}
```

Podobnie jak poprzednio, parametry należy przekazać jako wskaźniki. Stos jest modyfikowany. Element też może być modyfikowany, gdy go zwróci funkcja `pop`. Dlatego nie używamy słowa `const`:

```
void push(Stack * S, Finger * x) {
   if(wierzchołek stosu jest w ostatnim rekordzie bufora) {
    EXC_THROW(stack_overflow);
   }
   S->top++;
   S->buffer[S->top] =  * x; // musi być dereferencja wskaźnika
}
```

Ponieważ struktura stosu ma statyczny bufor, to zapisując `x` musimy zrobić dereferencję wskaźnika. Dereferencja spowoduje, że na stosie zostanie zapisana kopia `x`.
Program z normalnymi nazwami zmiennych:

```
void push(Stack * stack, Finger * element) {
   if(wierzchołek stosu jest w ostatnim rekordzie bufora) {
    EXC_THROW(stack_overflow);
   }
   stack->top++;
   stack->buffer[stack->top] =  * element; // musi być dereferencja wskaźnika
}
```

Warunek w instrukcji `if` napisz samodzielnie. Bufor ma rozmiar `STACK_CAPACITY` elementów, a indeks ostatniego dostępnego elementu to `STACK_LAST_ELEMENT_INDEX`.

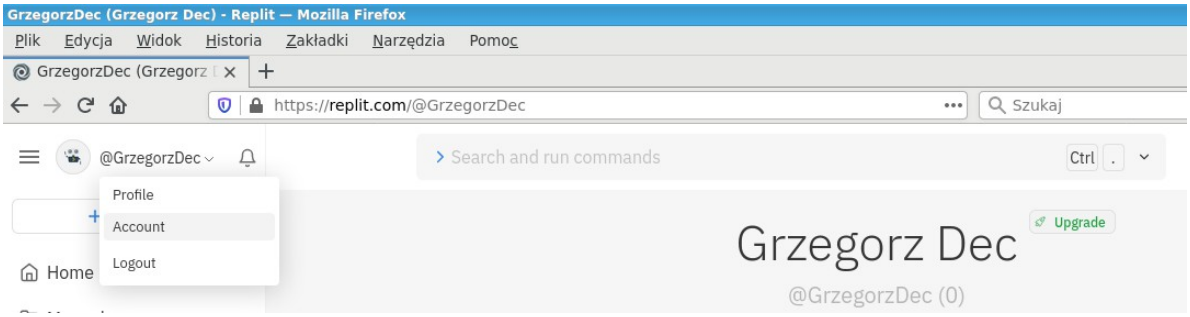
Kolejka – enqueue i dequeue

Napisz te funkcje samodzielnie. To nic trudnego. W funkcji `enqueue` pamiętaj o dereferencji wskaźnika na element, który ma być zapisany w kolejce.

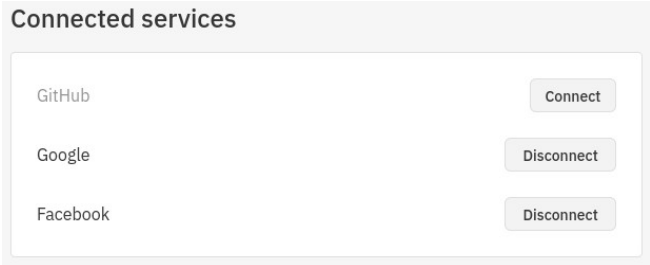
Usunięcie konta repl.it

Konto repl.it będzie używane w kolejnych ćwiczeniach. Jeżeli nie chcesz zostawiać swoich danych firmie Replit, Inc., usuń konto.

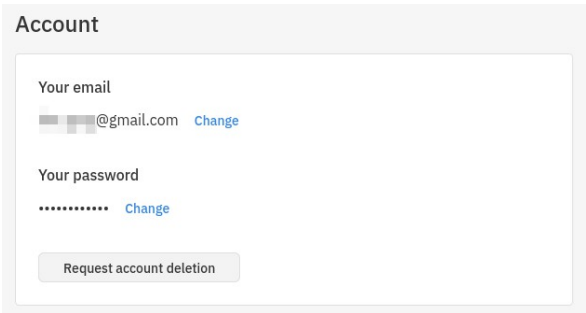
Przejdź na swoje konto repl.it.



U dołu strony jest lista połączonych usług. Odłącz się od nich:

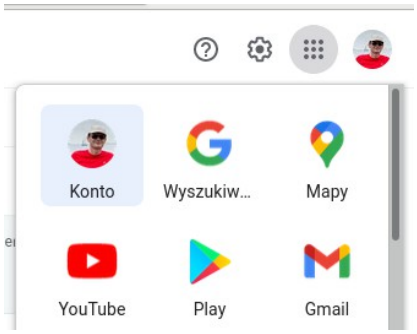


Usuń konto:

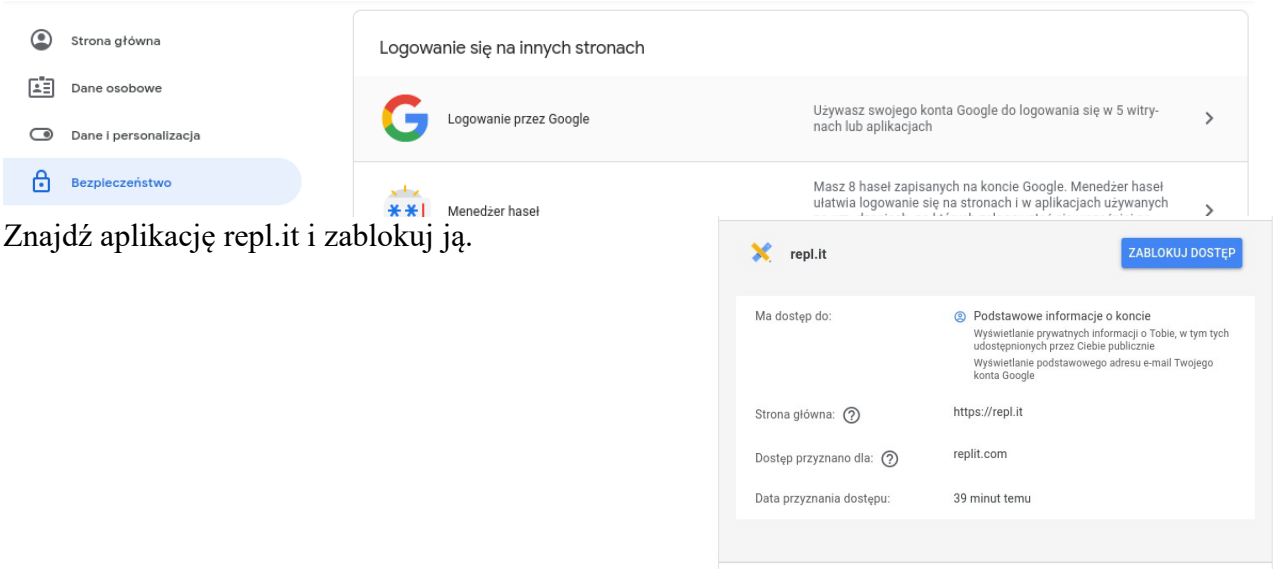


Usunięcie logowania w repl.it kontem google

Wejdź na swój profil w google. Z listy usług wybierz *Konto*.



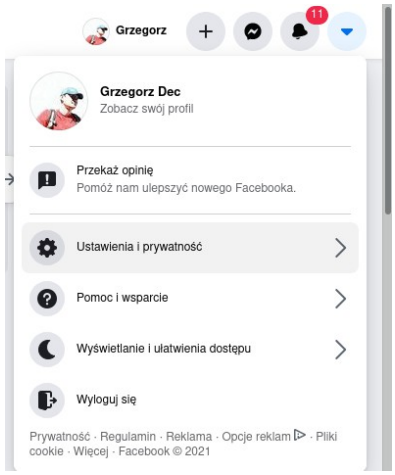
Wybierz grupę usług *bezpieczeństwo* i przewiń do funkcji *Logowanie się na innych stronach*.



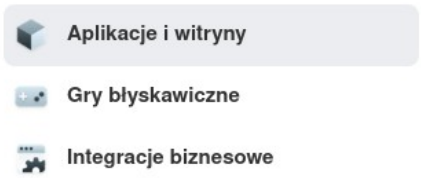
Znajdź aplikację repl.it i zablokuj ją.

Usunięcie logowania w repl.it kontem facebook

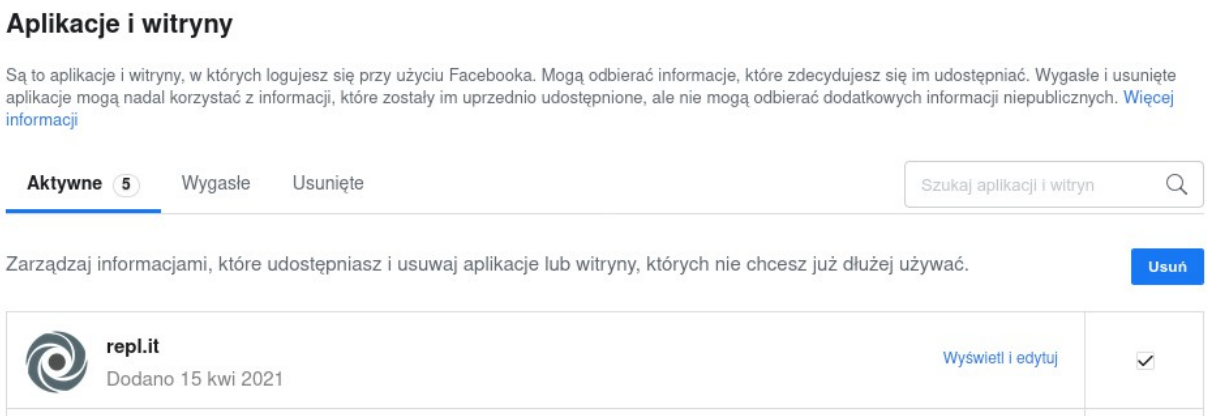
Wejdź na swój profil w Facebook'u. Z listy usług wybierz *Ustawienia i prywatność*.



Wybierz grupę usług *Aplikacje i witryny*.



Usuń repl.it z listy aktywnych aplikacji.



Literatura

1. Eric S. Roberts *Implementing Exceptions in C*, Research Report #40, Digital Equipment Corporation Systems Research Center, Palo Alto, California, March 1989.