29.03.2023 21:28:53

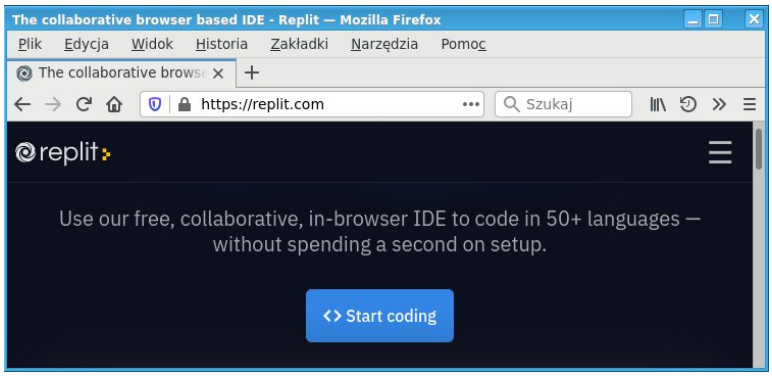
Algorytmy i Struktury Danych

Laboratorium
Listy

Przygotowanie do wykonania zadania.

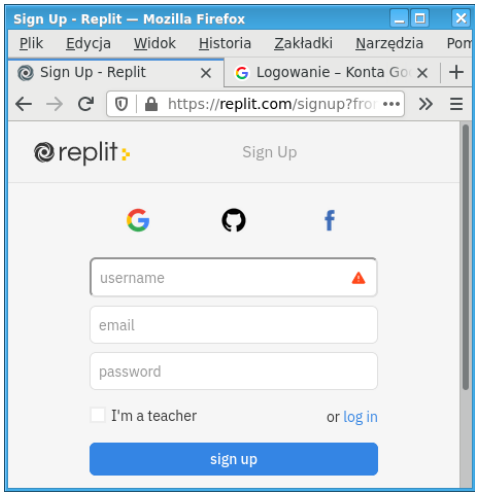
Rozpakuj archiwum z przykładowym projektem.

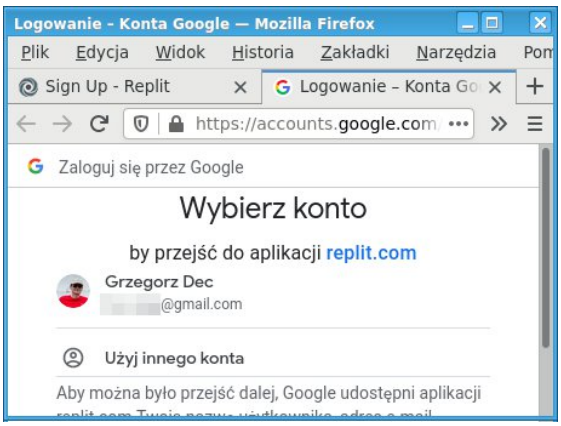
Wejdź na stronę `https://repl.it/`. Kliknij *Start coding*.



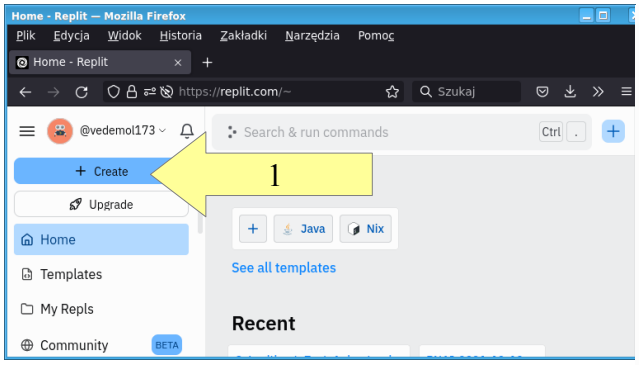
Zaloguj się kontem google albo facebook.

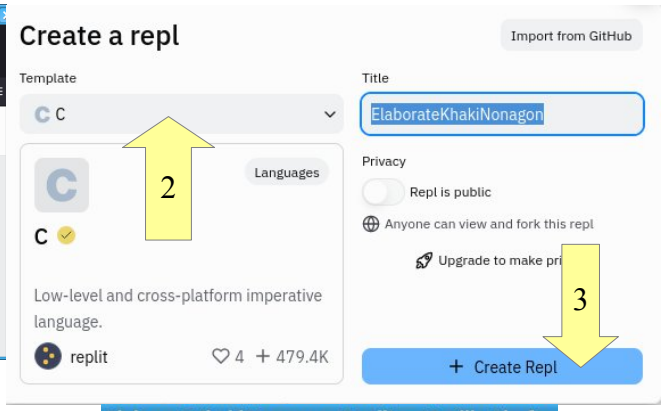
Możesz również utworzyć sobie tymczasowy e-mail (np. na stronie https://pl.emailfake.com/) i użyć go do zarejestrowania się na repl.it.



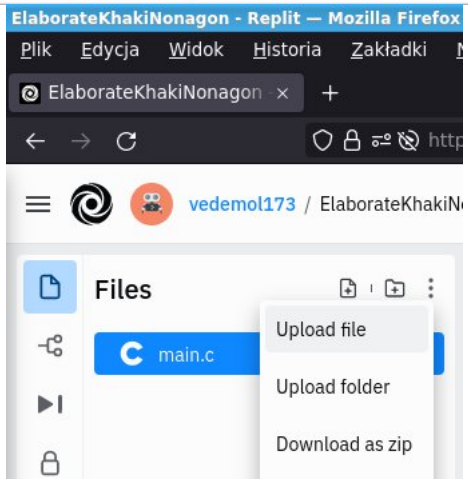


Utwórz nowy projekt w języku C.

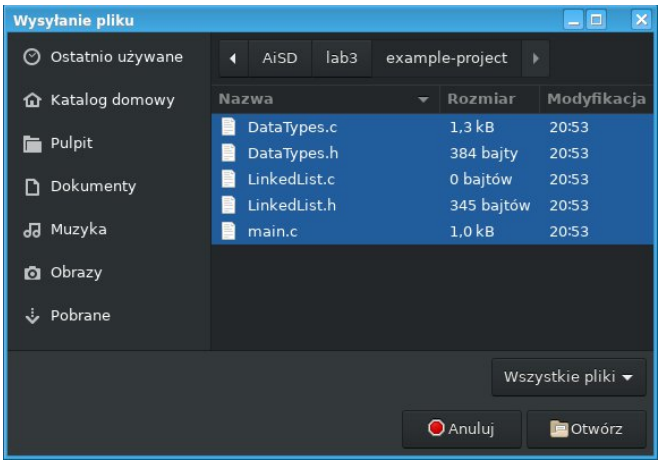




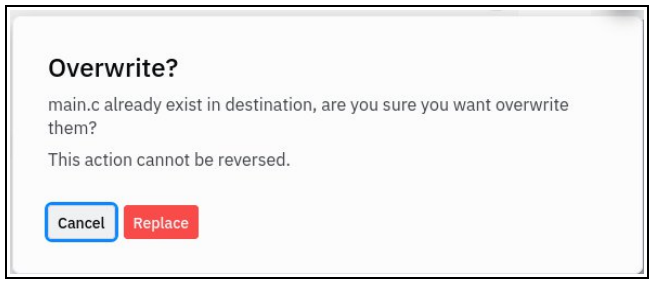
Kliknij w symbol ⋮ i wybierz *Upload file*.



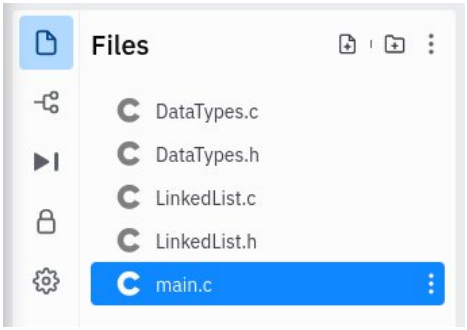
Wybierz wszystkie pliki z folderu, który zawiera rozpakowany przykładowy projekt.



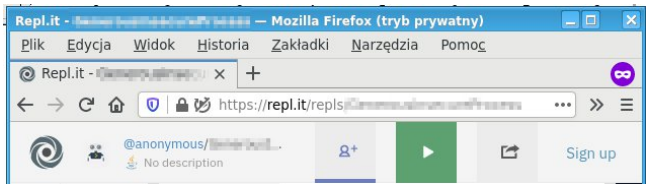
Potwierdź nadpisanie pliku `main.c` (kliknij *Replace*).



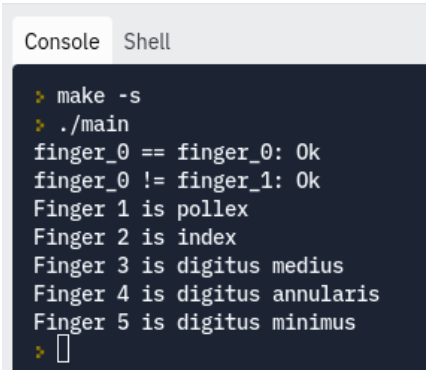
Struktura projektu powinna być taka, jak pokazano obok.



Teraz uruchom projekt – kliknij zielony przycisk Run



U mnie działa. U ciebie też powinno działać:



Lista dwukierunkowa

Wczytany program zawiera następujące elementy:

- deklarację typu `Finger`,
- deklarację typu listy i typu węzła listy,
- prototypy funkcji operacji na liście,
- przykładowy kod tworzący pięć palców i wyświetlający informacje o nich.

Wykonaj następujące zadanie:

1. W skrypcie do AiSD w sekcji 2.2.2 są podane algorytmy K2.1, K2.2, K2.3. Zaimplementuj te algorytmy – napisz funkcje `listFind`, `listInsert`, `listRemove`.
2. Do porównywania palców wykorzystaj funkcję `fingersEqual` z `DataTypes`.
3. W pliku `main.c` dopisz kod, który utworzy listę zawierającą pięć palców.
4. Wyświetl zawartość listy.
5. Wyszukaj dowolny palec funkcją `listFind` i wyświetl informację o nim (użyj funkcji `printFingerInfo`).
6. Dodaj do listy dowolny szósty palec i usuń go. Przed usunięciem i po usunięciu wyświetl zawartość listy.

Wskazówki.

1. W pliku `LinkedList.h` jest deklaracja typu listy, typu węzła listy i deklaracje funkcji K2.1, K2.2, K2.3.

2. Zaimplementuj te funkcje w pliku `LinkedList.c`.

Zadanie dla zdolnych studentek i studentów

Wykonaj poprzednie zadanie za pomocą listy jednokierunkowej. Trzeba samodzielnie napisać deklarację węzła tej listy i przerobić funkcje.

Zadanie dla mega zdolnych studentek i studentów

Wykonaj zadanie z listą dwukierunkową stosując tablicową implementację listy (strona 35 w skrypcie).

Podpowiedzi

Wyszukiwanie na liście

Struktura algorytmu K.2.1 i odpowiadająca mu struktura programu w języku C są następujące:

Algorytm	Program
LISTA-POSZ(L, k)	`listFind(L, k)`
x ← head[L]	`x = początek listy;`
dopóki *warunek* jest prawdą	`while(true == condition) {`
wykonuj x← next[x]	`x = następny węzeł listy;`
	`}`
zwróć x	`return x;`
	`}`

W języku C każda zmienna musi mieć określony typ. Zastanówmy się, jakie są typy poszczególnych zmiennych w algorytmie i jakie typy należy przypisać im w programie.

Algorytm		Program	
Zmienna	**Typ**	**Zmienna**	**Typ**
L	Lista	`L`	`DoubleLinkedList`
k	Struktura danych	`k`	`Finger`
x	Węzeł listy	`x`	`ListNode`

Uzupełnijmy program o zidentyfikowane typy:

```
ListNode listFind(DoubleLinkedList L, Finger k)
  ListNode x;

  // kod tego programu napisany bez użycia wskaźników jest bezsensowny

  return x;
}
```

W programie parametry `list` i `finger` funkcji `listFind` to struktury. Jeżeli przekazujemy jako parametr jakąś strukturę, to zastanówmy się nad następującymi kwestiami:

1. Czy muszę przekazać kopię struktury, czy wystarczy referencja do niej?
2. Czy funkcja będzie modyfikować parametry?

W naszym przypadku przekazywanie kopii nie ma uzasadnienia i funkcja nie modyfikuje parametrów. Zastosujemy wskaźniki. Program należy więc zmienić tak:

```
ListNode * listFind(const DoubleLinkedList * L, const Finger * k) {
  ListNode * x;

  x = L->head;  //początek listy
  while(condition) {
        x = x->next; następny węzeł listy
  }
  return x;
}
```

Mamy rok 2022. Język C powstał w roku 1972, kiedy firma Intel oferowała mikroprocesor 8008 taktowany zegarem 800 kHz. Żeby program skompilował się w skończonym czasie na komputerze z takim mikroprocesorem, nazwy zmiennych i funkcji pisało się krótkie. Dzisiaj ważne jest, żeby program był czytelny, debugowalny i mógł być rozwijany przez inną osobę, niż autor programu. Jednoliterowe nazwy zmiennych i funkcji są więc zaszłością historyczną i nie używamy ich. Zmodyfikujmy program, dołączając niezbędne pliki nagłówkowe:

```
#include "LinkedList.h"

ListNode * listFind(const DoubleLinkedList * list, const Finger * finger) {
  ListNode * listNode;

  listNode = list->head;
  while(condition) {
        listNode  = listNode->next;
  }
  return listNode;
}
```

Zapiszmy wyrażenie warunkowe występujące w pętli `while`. Sprawdzamy, czy osiągnięty został koniec listy i czy dane na liście są równe przekazanemu parametrowi `finger`. W języku C operatory porównania `==` i `!=` nie działają na typach złożonych. Musimy użyć funkcji, która porównuje dane z węzła listy z parametrem: `fingersEqual`. Warunek wygląda więc tak:

```
listNode != NULL && false == fingersEqual(finger, & listNode->data)
```

Funkcja `fingersEqual` spodziewa się dwóch parametrów typu `Finger *`:

```
bool fingersEqual(const Finger * fingerOne, const Finger * fingerTwo);
```

Zmienna `finger` ma odpowiedni typ, a `listNode->data` jest typu `Finger`. Żeby zmienić ten typ na wskaźnik, używamy operatora `&`.

```
#include "LinkedList.h"

ListNode * listFind(const DoubleLinkedList * list, const Finger * finger) {
  ListNode * listNode;

  listNode = list->head;
  while(listNode != NULL && false == fingersEqual(finger, & listNode->data)) {
        listNode  = listNode->next;
  }
  return listNode;
}
```

Pisząc program powinno się unikać tworzenia skomplikowanych wyrażeń, ponieważ ich debugowanie jest na ogół niemożliwe. Lepiej napisać więcej linijek czytelnego kodu, niż jednoliniową funkcję, której nikt nie rozumie. Dlatego kod funkcji `listFind` można zmodyfikować:

```
#include "LinkedList.h"

ListNode * listFind(const DoubleLinkedList * list, const Finger * finger) {
  ListNode * listNode;
  bool       endOfList, dataElementFound;

  endOfList        = false;
  dataElementFound = false;
  listNode         = list->head;

  while(1) {
        if(NULL == listNode) {
             endOfList = true;
        } else {
             dataElementFound = fingersEqual(finger, & listNode->data);
        }
        if(endOfList || dataElementFound) {
            break;
        }
        listNode  = listNode->next;
  }
  return listNode;
}
```

Dodawanie do listy

Struktura algorytmu K.2.2 i odpowiadająca mu struktura programu w języku C są następujące:

Algorytm	**Program**
LISTA-WSTAW(L, x)	`listInsert(L, x)`
next[x] ← head[L]	`następny węzeł po x = początek listy;`
jeśli head[L] not NILL	`if(NULL != początek listy) {`
to prev[head[L]]←x	`węzeł przed początkowym = x;`
	`}`
head[L] ← x	`początek listy = x;`
prev[x] ← NILL	`węzeł przed węzłem x = NULL;`
	`}`

Typy poszczególnych zmiennych w algorytmie i typy w programie:

Algorytm		**Program**	
Zmienna	**Typ**	**Zmienna**	**Typ**
L	Lista	`L`	`DoubleLinkedList`
x	Węzeł listy	`x`	`ListNode`

Program ze zidentyfikowanymi typami:

```
ListNode listInsert(DoubleLinkedList L, ListNode x) {
  // kod tej funkcji napisany bez wskaźników
  // jest zawiły i bezsensowny. Dlatego go nie ma.

  return x;
}
```

Podobnie jak poprzednio, parametry należy przekazać jako wskaźniki. Lecz tym razem modyfikujemy listę `L` i nowy węzeł `x`, więc nie będzie słowa `const`:

```
ListNode * listInsert(DoubleLinkedList * L, ListNode * x) {
  x->next = L->head;        //  następny węzeł po x = początek listy
  if(NULL != L->head) {
    L->head->previous = x; //węzeł przed początkowym = x
  }
  L->head = x;              //początek listy = x
  x->previous = NULL        //węzeł przed węzłem x = NULL
  return x;
}
```

Program z normalnymi nazwami zmiennych:

```
#include "LinkedList.h"

ListNode * listInsert(DoubleLinkedList * list, ListNode * newNode) {
  newNode->next = list->head;        //  następny węzeł po x = początek listy
  if(NULL != list->head) {
    list->head->previous = newNode; //węzeł przed początkowym = x
  }
  list->head = newNode;              //początek listy = x
  newNode->previous = NULL           //węzeł przed węzłem x = NULL
  return newNode;
}
```

Usuwanie z listy

Napisz samodzielnie, postępując wg schematu:

1. Utwórz strukturę funkcji.
2. Określ typy zmiennych i zadeklaruj potrzebne zmienne lokalne.
3. Wprowadź wskaźniki.
4. Wprowadź czytelne nazwy zmiennych.
5. Napisz program.

Operacje na liście

Pustą listę utworzymy deklarując zmienną typu `DoubleLinkedList`. Węzły listy możemy zadeklarować jako pojedyncze zmienne albo tablicę z węzłami:

```
#include "LinkedList.h"

DoubleLinkedList list;                          // pusta lista
ListNode node1, node2, node3, node4, node5;  // węzły ...
ListNode nodes[5];                              // ... albo tablica z węzłami
ListNode *node;                                 // wskaźnik na węzeł

list.head=NULL;
node1.previous = node1.next = ... reszta węzłów ... = NULL;
{int i; for(i=0; i<5; i++) nodes[i].previous = nodes[i].next = NULL; }
```

Pamiętamy, że zmienne lokalne w języku C nie są inicjalizowane automatycznie. Stąd potrzeba zerowania wskaźników.

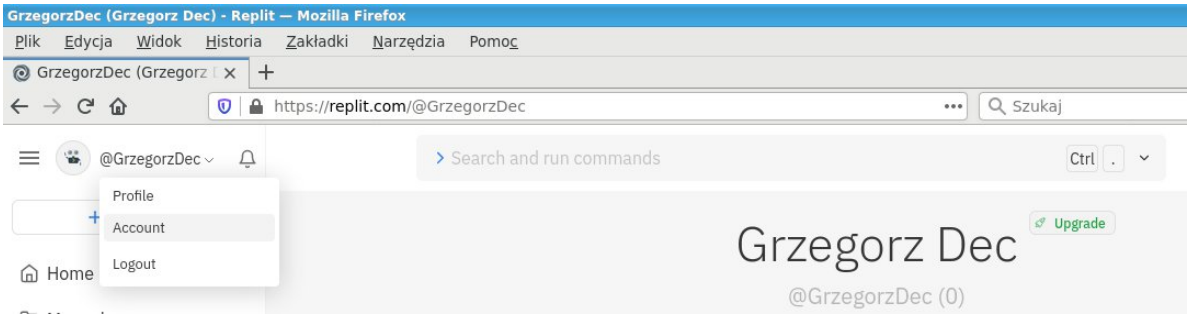
W przykładowym kodzie mamy tablicę `fingers` z danymi, które można wstawić na listę. Prosty program może wyglądać tak:

```
node1.data = fingers[0];
node2.data = fingers[1];
listInsert(& list, & node1); //funkcja oczekuje wskaźników, używamy operatora &
node = listFind(& list, & node2.data); // powinno być: node == NULL
if(NULL != node) {
  puts("node2.data found but is not on the list\n");
  exit(-1);
}
node = listFind(& list, & node1.data); // powinno być: node == & finger[0]
if(false == fingersEqual(& node->data, & fingers[0])) {
  puts("node1.data not found but is on the list\n");
  exit(-1);
}
printFingerInfo(& node->data); // ta funkcja chce Finger *, musimy użyć &
listRemove(& list, node);    //node jest typu ListNode *, nie trzeba operatora &
```

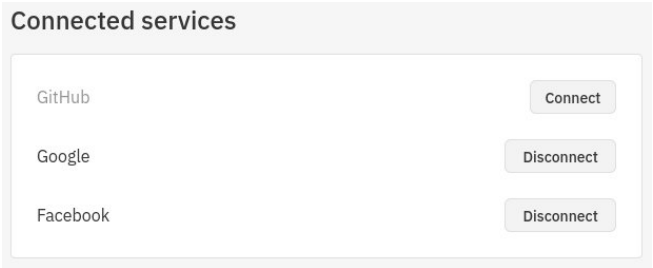
Usunięcie konta repl.it

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

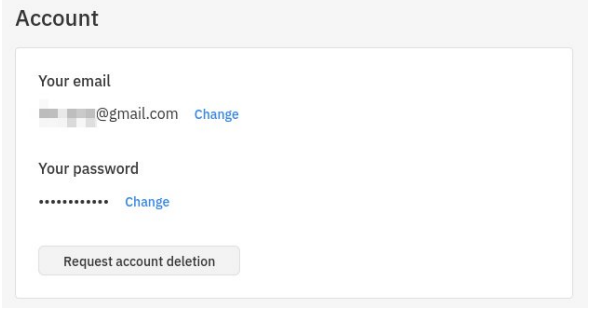
Przejdź na swoje konto repl.it.



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

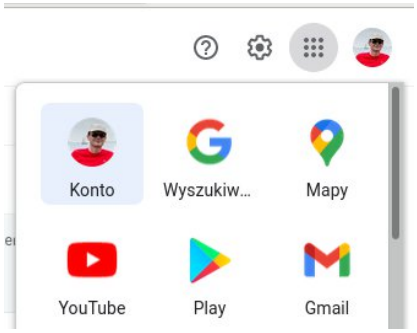


Usuń konto:

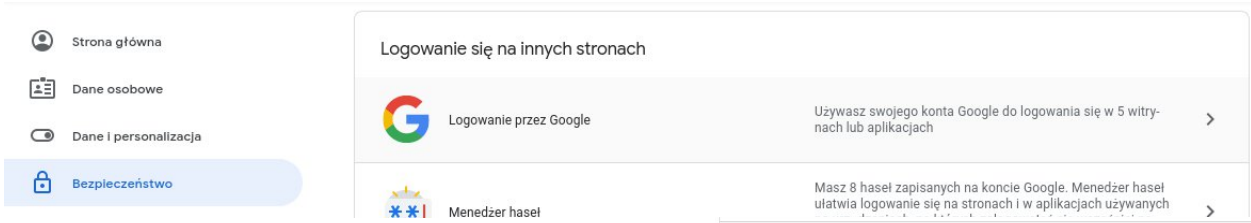


Usunięcie logowania w repl.it kontem google

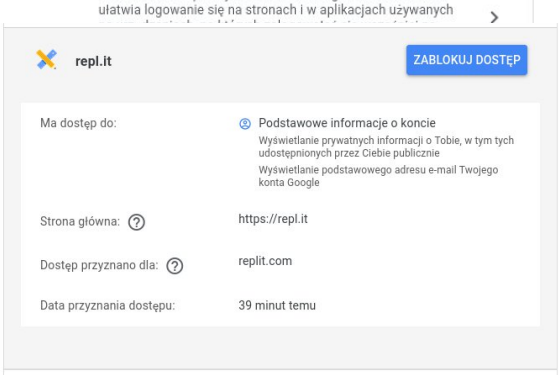
Wejdź na swój profil w google. Z listy usług wybierz *Konto*.



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

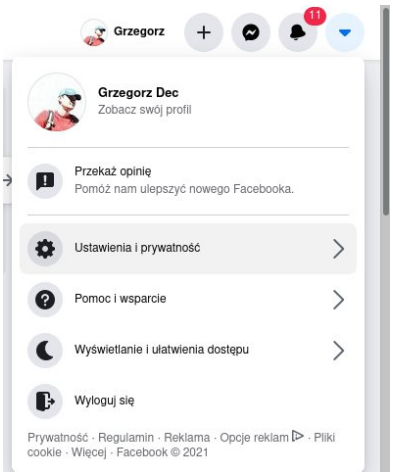


Znajdź aplikację repl.it i zablokuj ją.

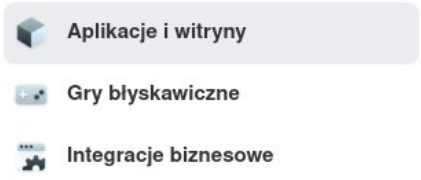


Usunięcie logowania w repl.it kontem facebook

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.