ВЕЖБА 1 – ДОДАТАК

Аргументи командне линије

Програмима, као и функцијама се могу прослеђивати параметри. Познато је, да је ***main*** функција главна функција програма, па се начином њене декларације одређује и да ли ће програм прихватати аргументе или не. Уколико **не** желимо да програм прихвата аргументе главну функцију најчешће дефинишемо на следећи начин:

```
void main(void)
{
    ...
    return;
}
```

Програми као што су конзонлне апликације (***Console Application***) најчешће захтевају да су конфигурабилне, односно да се могу покретати са различитим параметрима. За потребе писања оваквих програма (који су и предмет ове вежбе) у C стандарду описано је како треба дефинисати главну функцију програма. Ово се постиже на следећи начин:

```
int main (int argc, char* argv[])
{
    ...
    return 0;
}
```

Као што се може приметити, главна функција прихвата два параметра ***argc*** и ***argv***, који се програму преносе приликом покретања и тада добијају вредности:

- ***argc*** – број аргумената наведених у командној линију + 1 (за подразумевани, први, аргумент командне линије који је име програма са апсолутном путањом)
- ***argv*** – низ аргумената наведених у командној линију (у облику низа карактера). ***argv[0]*** је увек горенаведени подразумевани аргумент командне линије који представља име програма.

У документу ***vezba1.pdf***, одељак ***Додатни параметри пројекта*** на странама 11 и 12 описује на који начин је програму могуће проследити параметре Слика 15.

НАПОМЕНА: назив програма је подразумевани параметар, односно њега не треба уносити као аргумент командне линије!

Пример 1:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    int i;
    int n;

    if (argc < 2 || argc > 2)
    {
        printf("\nNiste uneli odgovarajuci broj argumenata!\n");
        exit(-1);
    }

    printf("\n1. Podrazumevani argument je naziv programa:\n%s\n", argv[0]);

    printf("\n2. Argument koji ste Vi prosledili je n = %s\n\n", argv[1]);

    n = atoi(argv[1]);

    for (i = 0; i < n; i++)
        printf("%d. Hello World!\n", i + 1);

    printf("\n");
    return 0;
}
```

За пример изнад неопходно је детаљно прочитати кратак увод о аргументима командне линије и уз помоћ ***vezba1.pdf*** подесити пројекат да би се програм успешно извршио.

- Уколико се програм покрене без подешавања, у конзоли ће бити исписана порука:

  "Niste uneli odgovarajuci broj argumenata!".

- Уколико се програм исправно покрене, програм ће у конзолу исписати:

  1) назив програма
  2) шта му је прослеђено као аргумент командне линије
  3) и онолико пута поруку ***"Hello World!".*** колико је корисник проследио преко аргумента командне линије

За успешно подешавање пројекта, унети као аргумент командне линије број који говори колико пута ће се ***"Hello World!"*** порука исписати. Овај аргумент ће се доделити целобројној променљивој ***n*.** На 19. линији кода:

```
n = atoi(argv[1]);
```

преводи се низ карактера аргумента командне линије у целобројни тип уз помоћ функције ***atoi(const char* str)***. У питању је функција из стандардне C библиотеке, и декласирана је у заглављу stdlib.h. Функција као аргумент прихвата показивач на низ карактера које треба да преведе у целобројни тип, а као повратну вредност враћа резултат превођења. Више информација о овој функцији може се добити њеним означавањем и притиском на тастер ***F1***.

Преглед техника контролисаног извршења програма са циљем отклањања грешака кроз примере

Пример 2 (случајна грешка):

```
#include <stdio.h>
#include <stdlib.h>

#define NUMROWS 20
#define NUMCOLS 30

void main(void)
{
    int i, j;
    int pixels = 0;

    for (i = 0; i < NUMROWS; i++)
        for (j = 0; j < NUMCOLS; j++);
            pixels++;

    printf("\npixels = %d\n", pixels);

    return;
}
```

Дати пример представља показно решење петље за пролазак кроз све тачке једне слике, где се само одређује редни број тренутне тачке и на крају, по изласку из петље, добија се укупан број тачака у слици.

Очекивано је да након изласка из петље укупан број тачака буде 20x30=600 и да у конзоли тако стоји и у испису.

Међутим, програм се не понаша као што је очекивано и укупан број тачака који програм налази је 1.

У случајевима када променљива мења вредност само на једном месту, једноставно је праћење њених промена постављањем тачке прекида на одговарајућу линију. Тако се одмах може закључити да се линија 14 (pixels++) извршава само једном, уместо 600 пута. Пажљивим гледањем кода закључујемо да је грешка у дефиницији петље и то је вишак тачка-зарез „;“ на крају претходне линије, односно друге (угњеждене) for петље.

Ово је врло чест тип - случајна грешка. Најчешће је последица залуталих знакова, веома се лако отклања, али се некада јако тешко проналази.

Пример 3 (недостајућа или неодговарајућа иницијализација)

```
int minval(int* A, int n)
{
  int currmin;

  for (int i = 0; i < n; i++)
    if (A[i] < currmin)
      currmin = A[i];

  return currmin;
}
```

У примеру изнад користи се променљива *currmin*, која је код првог извршења линије *if (A[i] < currmin)* још неиницијализована. Компајлер овакав случај пријављује само као упозорење (енг. warning) „uninitialized local variable 'currmin' used“. Ако се то упозорење занемари, могу бити изазвани проблеми током извршења програма, када се приликом поређења *A[i]* и *currmin* користи неиницијализована променљива. Извршење програма се тада прекида са поруком „The variable 'currmin' is being used without being defined“. Ово ће се десити само током контролисаног извршавања програма преведеног у Debug конфигурацији. Ако је програм преведн у Release конфигурацији никаква се грешка неће пријавити већ ће се програм погрешно извршавати и чудно понашати.

Пример 4 (лексичка грешка)

```
int minval(int* A, int n)
{
    int currmin = INT_MAX;
    int i;

    for (i = 0; i < n; i++)
        if (A[i] > currmin)
            currmin = A[i];

    return currmin;
}
```

Проблем у овом примеру је што би у поређењу *A[i]* и *currmin* требала да стоји релација <, што потпуно мења логику и резултат извршења ове функције/програма.

Оваква грешка би се једноставно могла открити додавањем дела кода за проверу резултата извршења функције *minval* на једноставном примеру:

```
void main(void)
{
    int A[5] = {12, 25, 3, 9, 8};

    _ASSERTE(minval(A, 5) == 3);

    return;
}
```

Програм би приликом извршења избацио поруку која указује на неиспуњење услова *minval(A, 5)==3*, а каснијим позиционирањем на излазак из функције *minval* се може утврдити да је повратна вредност једнака *INT_MAX* те да почетна вредност промењљиве *currmin* никад није измењена.

Пример 5 (грешка у копирању кода)

```
switch (i)
{
    case 1:
        print(1);
        break;
    case 2:
        print(2);
        break;
    case 3:
        print(1);
        break;
    case 4:
        print(4);
        break;
    default:
        break;
}
```

Пример би требало да испише вредност сваке итерације у конзолу. Међутим приликом покретања се примећује да то није случај.

У овом случају приказана је грешка приликом копирања дела кода (случаја 1 у остале случајеве, у датом примеру). Овакве грешке се веома тешко откривају, па је избегавање копирања кода строга препорука иако је коришћење постојећег кода веома пожељно. У случају копирања дела кода потребна је добра концентрација да се што више смањи могућност грешке.

Оваква грешка, када лако можете сумњати у позив функције *do_something(3)*, може се открити постављањем тачке прекида у тој функцији или додавањем исписа који оначава да је одговарајућа функција извршена.

Пример 6 (случајна грешка)

```
if (foo = 5)
    foo == 7;
```

Грешке у замени оператора за доделу и поређење се најчешће дешава случајно. Проблем је што прва грешка (*if (foo = 5)*) изазива увек истиниту повратну вредност (енг. true), док друга грешка (*foo == 7*) чини да тај исказ нема никаквог ефекта. Због синтаксне слабости језика C, ни једна од ове две грешке неће бити третирана као синтаксно нетачна.

Заустављањем на првој линији може се утврдити да иако је вредност промењиве *foo* различита од 5, услов је испуњен и следећа линија која се извршава је *foo == 7*. На тој линији може се видети да вредност промењиве више није иста, већ износи 5, што је такође последица прве грешке. Након извршења ове линије, уочава се да *foo* није 7, већ је и даље 5, што је последица друге грешке.

C претпроцесор

Јединствена карактеристика језика C је претпроцесор. Програм може да користи алатке које нуди претпроцесор да би био једноставан за читање, једноставан за мењање, преносив и ефикаснији.

Претпроцесор је програм који обрађује код пре него што код прође кроз компајлер. Он ради под контролом претпроцесорских командних линија и директива. Претпроцесорске директиве су смештене у изворни програм пре главне (енг. main) линије и пре него што изворни код прође кроз компајлер претражује га претпроцесор, тражећи било коју претпроцесорску директиву. Ако постоји било каква одговарајућа директива, предузимају се одговарајуће акције и након тога се изворни код предаје компајлеру. Претпроцесор заправо ствара нови код, на основу улазног изворног кода и претпроцесорских директива.

Претпроцесорске директиве прате специјалне синтаксна правила и почињу са симболом # и не захтевају тачка-зарез на крају. Скуп најчешће коришћених претпроцесорских директива:

Директива	Функција
#define	Дефинише макро за замену; у коду се свака појава идентификатора мења са дефиницијом
#undef	Поништава макро дефиницију
#include	Одређује датотеку коју треба укључити; садржај датотеке се практично копира, пре прослеђивања компајлеру
#ifdef	Провера постојања макро дефиниције; уколико дефиниција не постоји, код који је обухваћен директивом се изоставља
#ifndef	Провера постојања макро дефиниције; уколико дефиниција постоји, код који је обухваћен директивом се изоставља
#if	Провера сложенијег услова у време компајлирања; уколико услов није испуњен, код који је обухваћен директивом се изоставља
#else	Одређује алтернативе када услов за #if/#ifdef/#ifndef није испуњен
#endif	Одређује крај директиве #if/#ifdef/#ifndef

Tabela 1 Скуп најчешће коришћених претпроцесорских директива

Претпроцесорке директиве се могу поделити у три групе:

- група макро замене,
- група за укључивање датотеке и
- група контроле компајлера.

Макрои (макро замена)

Макро замена је процес у којем се идентификатор у програму мења са претходно дефинисаним низом карактера састављеним од једног или више токена који се могу користити у облику *#define <израз за замену>*.

Има следећи облик:

#define <идентификатор> <низ карактера за замену>

Претпроцесор мења сваку појаву *<идентификатора>* у изворном коду *<низом карактера за замену>*. Дефиниција почиње кључном речи *#define* праћене идентификатором и низом карактера са бар једним размаком (енг. space) између њих. Низ карактера може бити било какав текст, а идентификатор мора бити исправно C име.

Постоје различите форме макро замена, а најчешће су:

- једноставна макро замена,
- макро замена са аргументима и
- угњеждена макро замена.

Једноставна макро замена

Једноставна макро замена се најчешће користи за дефинисање константи:

#define PI 3.1415926

Писање дефиниција макроа великим словима је конвенција, али не и правило. Макро дефиниција може да буде више од дефиниције једноставне константне вредности, може такође да садржи изразе.

Макрои са аргументима

Претпроцесор омогућава дефинисање сложенијих и кориснијих облика замена. Имају следећи облик:

#define <идентификатор>(<f1>,<f2>,<f3>.....<fn>) <низ карактера за замену>

Приметите да не постоји размак између идентификатора и леве, отворене заграде. Аргументи *f1,f2,f3.....fn* су аналогни формалним аргументима у дефиницији функција.

Постоји основна разлика између једноставне замене објашњене пре и макроа са аргументом, тзв. макро позива.

Једноставан пример макроа са аргументима:

#define CUBE(x) ((x)(x)*(x))*

Ако би се следећи исказ појавио у каснијем изворном коду:

volume = CUBE(side);

претпроцесор би развио исказ у

volume = ((side)(side)*(side));*

Угњеждени макрои

У дефиницији једног макроа, такође се може користити дефиниција другог макроа, тј. макро дефиниције могу бити угњеждене. Пример угњеждене макро дефиниције:

#define PI 3.1415926

#define SQUARE(x) ((x)(x))*

*#define CIRCLE_AREA(r) (PI*SQUARE(r))*

Рекурзивне макро дефиниције нису могуће (позив истог макроа у телу дефиниције односно низу карактера за замену) јер, једноставно, макро дефиниција не постоји пре своје прве дефиниције. Рекурзивни позиви макро дефиниције су могући:

#define MIN(m,n) ((m)<(n)?(m):(n))

x = MIN(10,MIN(7,11));

Поништавање макро дефиниције

Макро дефиниција може бити поништена исказом:

#undef <идентификатор>

Ово је корсино када је потребно ограничавање дефиниције на само један део програма.

Укључивање датотеке

Претпроцесорска директива

#include "име датотеке"

се може користити за укључивање било које датотеке у програм. Ако функције, променљиве и/или макрои постоје у спољашњој датотеци, могу се користити и у програму, уколико је та датотека укључена.

У директиви, *име датотеке* је обухваћено знацима навода и представља име датотеке која садржи потребне дефиниције и/или функције. Дефинисана у оваквом формату, директива означава претрагу датотеке на датој апсолутној путањи, или релативној, у односу на директоријум у којем се пројекат налази. Додатно директива може имати следећи облик:

#include <име датотеке>

У оваквој форми, без знакова навода, обавезно ограничено малим заградама <>, датотека се претражује само у стандардним дефинисаним директоријумима.

Контрола компајлера

Претпроцесор подржава две форме провере услова у време компајлирања.

#ifdef / *#ifndef <макро дефиниција>* омогућава проверу да ли макро дефиниција постоји, односно да ли је макро дефинисан. Може се користити у пару са *#else* за дефинисање алтернативе у случају да макро није дефинисан. Обавезно се завршава са *#endif* директивом.

Генералнију форму за проверу услова у тренутку компајлирања нуди директива

#if <константни услов>

уз коју се могу користити директиве *#else* и *#elif <константни услов>* за дефинисање алтернативе у случају неиспуњења услова. Обавезно се завршава са *#endif* директивом.

```
#if <константни услов 1>
        Исказ1;
        Исказ2;
#elif <константни услов 2>
        Исказ3;
        Исказ4;
#else
        Исказ5;
#endif
```

Пример 7 (грешка у макроу)

```
#define SQUARE(n) (n * n)

val = SQUARE(a + b);
```

Макро *SQUARE* би требало да враћа квадрат прослеђене вредности. Међутим, јавља се не баш очигледан проблем у случају да је n неки израз, а не константна вредност. Због природе макроа, након замене линија ће бити измењена у:

```
val = a * b + a + b;
```

уместо

```
val = (a + b) * (a + b);
```

па је резултат погрешан. Ово би се могло открити провером добијене вредности, додавањем линије:

```
_ASSERTE(val == (a + b) * (a + b));
```