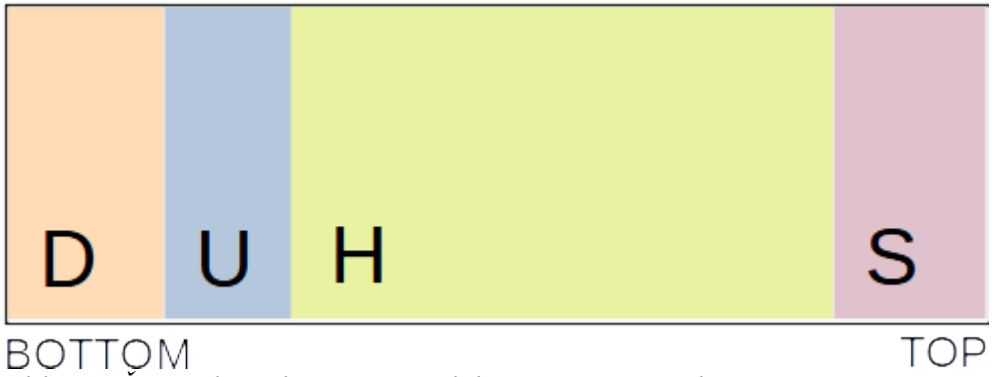
VEŽBA 1

Programi namenjeni za izvršavanje na računarskim sistemima imaju tipičan razmeštaj objekata unutar **radne memorije** koji je prikazan na slici 1. Ovo važi kako za velike računarske sisteme, tako i za emebeded softver koji se izvršava na mikrokontrolerima. Uz izvesna ograničenja, važi i za sisteme koji podržavaju izvršavanje više programa, **zadataka** (*engl. task*) ili **niti** (*engl. thread*) istovremeno sadržavajući pri tome jednu ili više **procesorskih jezgara** (*engl. CPU core*).



Slika 1 - Šematski prikaz memorijskih zona unutar radne memorije

D – inicijalizovani podaci (*initialized data*) – globalne promenljive programa kojima je dodeljena početna vrednost (inicijalizovani su) pre početka izvršavanja programa (funkcije `main` ako je program pisan u jeziku C).

U – neinicijalizovani podaci (*unitialized data*) – globalne promenljive kojima nije dodeljena početna vrednost (neinicijalizovani su). Može se računati da im je vrednost jednaka nuli pre početka izvršavanja programa.

S – stek (*engl. stack*). Popunjava se od najviše adrese naniže. Ovde se upisuju lokalne promenljive koje pripadaju potprogramima (funkcijama u jeziku C) kao i **povratne adrese** odakle se izvršavanje programa nastavlja nakon što se neki potprogram završi.

H – prostor na raspolaganju za **dinamičku dodelu memorije** (*engl. heap*). Obično se popunjava od nižih ka višim adresama.

Prostor na raspolaganju za D i U je obično određen pre početka izvršavanja programa i zavisi od broja korišćenih globalnih promenljivih. Granica između H i S može biti dinamička, tj. svaki od njih može da se širi dok se ove dve zone ne dodirnu. Kod nekih sistema veličina S zone može biti unapred određena.

Pomenuti razmeštaj ne uzima u razmatranje **izvršni kod** (*engl. executable code*) programa koji sadrži niz mikroinstrukcija koji sadrži instrukcije programa koji se na računarskom sistemu u datom trenutku izvršava. Ovaj kod je u mikrokontrolerima obično upisan u trajnu programsku memoriju (ROM ili Flash) i tu se upisuje tokom specijalnog postupka programiranja (tzv. spuštanje programa).

Funkcije za rad sa dinamičkom memorijom – heap-om

Funkcija koja se koristi za dodelu, odnosno zauzimanje memorije u H zoni je **`malloc`**:

```
void* malloc(size_t size);
```

Obratiti pažnju da parametar treba da bude količina memorije izražena u jedinicama veličine podatka tipa `char` (u bajtovima). U cilju određivanja ove veličine najsvrsishodnije je koristiti operaciju `sizeof` ugrađenu u jezik C. Funkcija vraća pokazivač na dodeljenu zonu memorije. Npr.

```
int* dyn_var;
// ...
dyn_var = malloc(sizeof(int));
*dyn_var = 100;  // koriscenje memorije
// ...
free(dyn_var);  // oslobadjanje memorije
```

Kako poslednji red primera pokazuje, oslobađanje prostora ranije zauzetog sa **`malloc`** postiže se pozivom funkcije **`free`**:

```
void free(void* ptr);
```

Zadaci

1. Napisati program koji dinamički zauzima memoriju iz H zone dovoljnu za unos jednog broja tipa `int`, traži od korisnika unos tog broja preko terminala[1], ispisuje uneti broj i na kraju oslobađa zauzetu memoriju. Obratiti pažnju na pristup memoriji preko pokazivača!
2. Napisati program koji sadrži funkciju `int arith_sum(int start, int stop)` koja izračunava sumu svih celih brojeva počev od `start` do `stop` uključujući i broja `stop`. Funkcija treba da poziva samu sebe unutar svog tela, odnosno da koristi tehniku poznatu kao **rekurzija**.

Dodatni neobavezni zadatak:

3. Sve lokalne promenljive kao i povratne adrese koje se koriste pri povratku iz funkcija zauzimaju mesto na steku, tj. S zoni. Napisati funkciju `check_stack` koja vraća količinu prostora na steku koja je zauzeta od početka izvršavanja programa. Stanje steka ispisivati pri ulasku (*enter*) u funkciju `arith_sum` i neposredno pre izlaska (*exit*). Koliko prostora na steku zauzme svaki ulazak u funkciju? Imajući u vidu da svaka promenljiva tipa `int` i svaka povratna adresa zauzima 32 bita (za 32-bitne programe na x86 sistemima), može li se rekonstruisati na šta se prostor na steku troši?

Pomoć za rešavanje zadataka

Sumiranje niza sukcesivnih brojeva

Za nizove sukcesivnih brojeva počev od a do b važi:

$$\sum_{i=a}^{b} i = a + \sum_{i=a+1}^{b} i \quad , \quad a<b$$

Ovu relaciju treba iskoristiti kao osnovu rekurzivnog postupka.

1 Za unos broja sa terminala koristiti funkciju **`scanf_s`** koja se koristi na isti način kao i tradicionalna funkcija **`scanf`**. Prednost ove varijante je u povećanoj imunosti na greške koje korisnik može da napravi.

Provera stanja steka

Funkcija `check_stack` pri svakom pozivu vraća udaljenost sledećeg slobodnog mesta na steku od prvog zauzetog podatka na istom tom steku. Funkcija main mora sadržati odgovarajuću inicijalizaciju globalne promenljive `tos` da bi rezultat bio relevantan.

```
static char* tos;

int check_stack()
{
     char test;
     return (&test) - tos;
}

// ovde dolazi ostatak programa

int main()
{
     char ref; // ova promenljiva mora biti prva definisana
     // ovde dolaze ostale promenljive funkcije main

     tos = &ref;    // ova operacija je kljucna za ispravan rad
                    // funkcije check_stack

     // ovde dolazi ostatak funkcije main

     return 0;
}
```